Scheduling with Many Shared Resources

1st Max A. Deppert  
Department of Computer Science  
Kiel University  
Kiel, Germany  
0000-0003-3083-7998

2nd Klaus Jansen  
Department of Computer Science  
Kiel University  
Kiel, Germany  
0000-0001-8358-6796

3rd Marten Maack  
Heinz Nixdorf Institute and  
Department of Computer Science  
Paderborn University  
Paderborn, Germany  
0000-0001-7918-6642

4th Simon Pukrop  
Heinz Nixdorf Institute and  
Department of Computer Science  
Paderborn University  
Paderborn, Germany  
0000-0002-4473-5215

5th Malin Rau  
Department of Computer Science  
Universität Hamburg  
Hamburg, Germany  
0000-0002-5710-560X

Abstract—Consider the many shared resources scheduling problem where jobs have to be scheduled on identical parallel machines with the goal of minimizing the makespan. However, each job needs exactly one additional shared resource in order to be executed and hence prevents the execution of jobs that need the same resource while being processed. Previously, an approximation ratio of asymptotically 2 was the best known result for this problem. Furthermore, a 6/5-approximation for the case with only two machines was known as well as a PTAS for the case with a constant number of machines. We present a simple and fast 5/3-approximation and a much more involved but still reasonable 1.5-approximation. Furthermore, we provide a PTAS for the case with only a constant number of machines, which is arguably simpler and faster than the previously known one, as well as a PTAS with resource augmentation for the general case. The approximation schemes make use of the N-fold integer programming machinery, which has found more and more applications in the field of scheduling recently. It is plausible that the latter results can be improved and extended to more general cases. Lastly, we give an inapproximability result for the natural problem extension where each job may need up to a constant number of different resources, namely 3, ruling out better than 5/4 approximations for that case.

Index Terms—Scheduling, Approximation, Parallel Identical Machines, Resource Constraints, Conflicts

I. INTRODUCTION

We consider the problem of makespan minimization on identical parallel machines with many shared resources or many shared resources scheduling (MSRS) for short. In this problem, we are given m identical machines, a set J of n jobs, and a processing time or size p_j \in \mathbb{N}_{>0} for each job j \in J. Furthermore, each job needs exactly one additional shared resource in order to be executed and no other job needing the same resource can be processed at the same time. Hence, the jobs are partitioned into (non-empty) classes C, i.e., \bigcup_{c \in C} c = J, such that each class corresponds to one of the resources. A schedule (\sigma, t) maps each job to a machine \sigma : J \rightarrow \{1, \ldots, m\} and a starting time t : J \rightarrow \mathbb{N}_{\geq 0}. It is called valid if no two jobs overlap on one machine and no two jobs of the same class are processed in parallel, i.e.,

\begin{itemize}
  \item \forall j, j' \in J, j \neq j' with \sigma(j) = \sigma(j'): t(j) + p_j \leq t(j') \text{ or } t(j') + p_j' \leq t(j)
  \item \forall c \in C : j, j' \in c, j \neq j': t(j) + p_j \leq t(j') \text{ or } t(j') + p_j' \leq t(j)
\end{itemize}

The makespan C_{\text{max}} of a schedule is defined as \max_{j \in J} t(j) + p_j and the goal is to find a schedule with minimum makespan. Note that MSRS also models the case in which some jobs do not need a resource since in this case private resources can be introduced.

A. State of the Art and Motivation

The study of scheduling problems with additional resources has a long and rich tradition. Already in 1983, Blazewicz et al. [1] provided a classification for such problems along with basic hardness results and several additional surveys have been published since then [2–4]. The MSRS problem, in particular, was introduced by Hebrard et al. [5] who considered the scheduling of download plans for Earth observation satellites and provided a (2m/(m+1))-approximation for the problem. In this application scenario a satellite has a communication link with a ground station only for very limited time during which files stored on several memory banks should be downloaded via several download channels. The download time of a file is proportional to its size, but only one file from each memory bank can be downloaded at the same time. Hence, the download channels correspond to the parallel machines, the files to the jobs, and the memory banks to the resources. Strusevich [6] revisited MSRS and presented an additional application in human resource management, where the additional resources correspond to experts that have to join teams (machines) for certain projects (jobs). Moreover,
he provided a faster, alternative \((2m/(m+1))\)-approximation that is claimed to be simpler as well, and a \(6/5\)-approximation for the case with only two machines. The work also extends the three field notation for scheduling problems based on the convention for additional resources introduced in [1] to encompass the problem at hand. In particular, MSRS is denoted as \(P^{\text{res}}|111|C_{\text{max}}\) in this notation. The most recent result regarding MSRS is due to Dôsa et al. [7] who provided an efficient polynomial time approximation scheme (EPTAS) for MSRS with a constant number of machines. In fact, the EPTAS even works for a more general setting where each job \(j\) additionally may only be assigned to a machine belonging to a given set \(M(j)\) of eligible machines.

We employ standard notation regarding approximation schemes: A *polynomial time approximation scheme* (PTAS) provides a polynomial time \((1 + \varepsilon)\)-approximation for each \(\varepsilon > 0\). It is called *efficient*, or EPTAS, if its running time is of the form \(f(1/\varepsilon)\text{poly}(|I|)\) where \(f\) is some function and \(|I|\) the encoding length of the instance \(I\). Moreover, an EPTAS is called *fully polynomial time approximation scheme* (FPTAS) if the function \(f\) is a polynomial.

Since MSRS includes makespan minimization on identical machines (without resource constraints) as a subproblem, it is NP-hard already on two machines and strongly NP-hard if the number of machines is part of the input due to straightforward reductions from the partition and 3-partition problem, respectively. Hence, approximation schemes are essentially the best we can hope for.

The MSRS problem has also been considered with regard to the total completion time objective [8, 9]. The study of this variant is motivated by a scheduling problem in the semiconductor industry. On one hand, the authors show NP-hardness for generalizations of the problem, and on the other, they argue that the approach yielding a polynomial time algorithm for total completion time minimization in the absence of resource constraints leads to a \((2 - 1/m)\)-approximation for the considered problem.

Another way of looking at MSRS is to consider it as variant of scheduling with conflicts, where a conflict graph is given in which the jobs are the vertices and no two jobs connected by an edge may be processed at the same time. This problem was introduced for unit processing times by Baker and Coffman in 1996 [10]. It is known to be APX-hard [11] already on two machines with job sizes at most 4 and a bipartite agreement graph, i.e., the complement of the conflict graph. There are many positive and negative results for variants of this problem (see, e.g., [10, 11] and the references therein). For instance, the problem is NP-hard on cographs with unit-size jobs but polynomial time solvable if the number of machines is constant [12]. Note that in the case of MSRS, we have a particularly simple cograph, i.e., a collection of disjoint cliques.

**B. Results**

We present a \(5/3\)-approximation in Section II, a \(3/2\)-approximation in Section III, approximation schemes in Section IV, and inapproximability results in Section V. Note that the \(5/3\)- and \(3/2\)-approximation have better approximation ratios than the previously known \((2m/(m+1))\)-approximation already for 6 and 4 machines, respectively.

The \(5/3\)-approximation is a simple and fast algorithm that is based on placing full classes of jobs taking special care of classes containing jobs with particularly big sizes and of classes with large processing time overall. While the \(3/2\)-approximation reuses some of the ideas and observations of the first result, it is much more involved. To achieve the second result, we first design a \(3/2\)-approximation for the instances in which jobs cannot be too large relative to the optimal makespan. Next, we design an algorithm that carefully places classes containing such large jobs and uses the first algorithm as a subroutine for the placement of the remaining classes.

We provide an EPTAS for the variant of MSRS where the number of machines is constant and an EPTAS with resource augmentation for the general case. In particular, we need \([(1 + \varepsilon)m]\) many machines in the latter result. Both results make use of the basic framework introduced in [13] which in turn utilizes relatively recent algorithmic results for integer programs (IPs) of a particular form – so-called \(N\)-fold IPs. Compared to the mentioned work by Dôsa et al. [7] – which provides an EPTAS for the case with a constant number of machines as well – our result is arguably simpler and faster (going from at least triply exponential in \(m/\varepsilon\) to doubly exponential). We also provide the result with resource augmentation for the general case, which may be refined in the future to work without resource augmentation as well. Moreover, it seems plausible that the use of \(N\)-fold IPs in the context of scheduling with additional resources may lead to further results in the future, which do not have to be limited to approximation schemes.

Finally, we provide inapproximability results for variants of MSRS where each job may need more than one resource. In particular, we show that there is no better than \(5/4\)-approximation for the variant of MSRS with multiple resources per job, unless \(P = NP\), even if no job needs more than three resources and all jobs have processing time 1, 2, or 3. Previously, the APX-hardness result due to Even et al. [11] for scheduling with conflicts was known, which did focus on a different context and in particular does not provide bounds regarding the number of resources a job may require.

In the present version of the paper, many details and proofs in particular are omitted due to space constraints and are available in a long version [14].

**C. Further Related Work**

As mentioned above, there exists extensive research regarding scheduling with additional resources and we refer to the
surveys [1–4] for an overview. For instance, the variant with only one additional shared renewable resource where each job needs some fraction of the resource capacity has received a lot of attention (see [15–18] for some relatively recent examples). Interestingly, Hebrard [5] pointed out that this basic setting is more closely related MSRS than it first appears: Consider the case that we have dedicated machines, i.e., each job is already assigned to a machine and we only have to choose the starting times, each job needs one unit of the single additional shared resource, and the shared resource has some integer capacity. This problem is equivalent to MSRS if the multiple resources taken on the roles of the machines and the machines take the role of the single resource. Hence, results for variants of this setting translate to MSRS as well. For instance, MSRS can be solved in polynomial time if at most two classes include more than one job [19] and [20] yields a (3 + ε)-approximation.

Scheduling with conflicts has also been studied from the orthogonal perspective, where jobs that are in conflict may not be processed on the same machines. This problem was already studied in the 1990’s (see e.g. [12, 21]), and there has been a series of recent results [22–24] regarding the setting corresponding to MSRS where the conflict graph is a collection of disjoint cliques.

D. Preliminaries

We introduce some additional notation and a first observation that will be used throughout the sections. For any set of jobs X let \( p(X) = \sum_{j \in X} p_j \) denote its total processing time. Also let \( p(j) = p_j \) for all jobs \( j \in J \). While creating or discussing a schedule, for any machine \( m \) denote by \( p(m) \) the (current) total load of jobs on that machine \( m \). Subsequently, for a set of machines \( M \), \( p(M) = \sum_{m \in M} p(m) \). For any combination of a set \( X \subseteq \{ J, C \} \), a relation \( \in \subseteq \{<, \leq, \geq, >\} \), and a number \( \lambda \), we define \( X_{\lambda} = \{ x \in X \mid p(x) \in \lambda \} \). Furthermore, given an interval \( [x,y) \), we define \( X_{[x,y)} = \{ x \in X \mid p(x) \in [x,y) \} \). For example it holds that \( J_{>1/2} = \{ j \in J \mid p(j) > 1/2 \} \) and \( C_{[1/2, 3/4]} = \{ c \in C \mid p(c) \in (1/2, 3/4) \} \). It holds that \( \text{OPT} \geq \max \{ \frac{p(J)}{m}, \max_{c \in C} p(c) \} \). Hence, we assume that \( m < |C| \) as otherwise, there is a trivial schedule with one machine per class.

II. A 5/3-APPROXIMATION

In this section, we introduce a simple algorithm that gives some intuition on the problem that will be used more cleverly in the next section. We start by lower bounding the makespan \( T \) of an optimal schedule and construct a schedule with makespan at most \( \frac{5}{3} T \). The algorithm works by placing full classes of jobs in a specific order. More precisely, first classes that contain a job of size at least \( \frac{1}{2} T \), then classes with total processing time larger than \( \frac{2}{3} T \), and lastly, all residual classes get placed.

Theorem 1. There exists an algorithm that, for any instance \( I \) of MSRS, finds a schedule with makespan bounded by \( \frac{5}{3} T \) in \( O(|I|) \) steps, where for the jobs \( j_m \) and \( j_{m+1} \) with \( m \)-th and \( (m + 1)\)-st largest processing time we define \( T := \max \{ \frac{1}{2} p(J), \max_{c \in C} p(c), p(j_m), p(j_{m+1}) \} \).

As noted earlier, \( T \) denotes a lower bound on the makespan. We scale each job by \( 1/T \). As a consequence all jobs have a processing time in \( (0,1) \) and the total load is bounded by \( m \). Denote by \( C_{B} := \{ c \in C \mid |c \cap J_{>1/2}| = 1 \} \) all classes containing a job of size greater than \( 1/2 \). We aim to find a schedule with makespan in \([1, 5/3]\). The following observation is directly implied by the definition of \( T \). In the algorithm we use the following observation.

Lemma 1. Each class \( c \in C \) holds \( |c \cap J_{>1/2}| \leq 1 \), it holds that \( |C_{B}| = |J_{>1/2}| \leq m \), and each class \( c \in C_{>2/3} \setminus C_{B} \) can be partitioned into parts \( c_1 \) and \( c_2 = c \setminus c_1 \) such that \( 1/3 \leq p(c_1) \leq 2/3 \) and \( p(c_2) \leq 2/3 \). This partition can be found in time \( O(|c|) \).

Proof. The partition of classes \( c \in C_{>2/3} \setminus C_{B} \) works as follows: If there exists a job \( j \in c \) with \( p(j) > 1/3 \), we define \( c_1 = \{ j \} \) and \( c_2 = c \setminus c_1 \). Note that \( c \) does not contain a job with processing time larger than \( 1/2 \) and hence, \( p(c_1) \in (1/3, 1/2] \) and \( p(c_2) = \min(\{p(c) \mid c \subseteq C \}) \leq 1 - 1/3 = 2/3 \).

Otherwise, greedily add jobs from \( c \) to an empty set \( c_1 \) until \( p(c_1) \geq 1/3 \) and set \( c_2 = c \setminus c_1 \). Since all the jobs of \( c \) have processing time at most \( 1/3 \), it holds that \( p(c_1) \in [1/3, 2/3] \). Consequently, it holds that \( p(c_2) \leq 2/3 \) as well. \( \square \)

The algorithm works as follows:

A. Algorithm: Algorithm_5/3

Step 1. Consider all classes containing a job with processing time larger than \( 1/2 \), \( C_{B} \). Each of these classes is assigned to an individual machine, and all jobs from such a class are scheduled consecutively, see Fig. 1a.

Step 2. Consider all remaining classes with total processing time larger than \( 2/3 \), \( C_{>2/3} \setminus C_{B} \). Try to add these classes on the machines filled with the classes \( C_{B} \) and afterward proceeds to empty machines, see Fig. 1b. If the considered machine has load in \([1, 5/3]\), close the machine and no longer attempt to place any other job on it. Note that after placing the classes \( C_{B} \) all machines remained open. Let \( m_i \) be the machine we try to place class \( c \in C_{>2/3} \setminus C_{B} \). On \( m_i \) has load \( p(m_i) \leq 5/3 - p(c) \), place the entire class on this machine and close it. Otherwise, partition the class \( c \) in two parts \( c_1 \) and \( c_2 \) such that \( p(c_2) \leq p(c_1) \leq 2/3 \) (cf. Lemma 1). Place the larger part \( c_1 \) on the current machine starting at \( 5/3 - p(c_1) \) and close it, moving to the next machine. All jobs on this machine are delayed such that the first job starts at \( p(c_2) \). All jobs from \( c_2 \) are scheduled between \( 0 \) and \( p(c_2) \) on this machine. If it has load of at least \( 1 \), this machine is closed as well.

Step 3 (Greedy). Finally, place the classes \( C \setminus C_{B} \), see Fig. 1c. Consider the residual machines one after another and add each class \( c \in C \setminus C_{B} \) entirely to the considered
machine. As soon as the load of a machine exceeds 1 close it and move to the next.

Proof of Theorem 1. We show the correctness and approximation ratio of Algorithm_5/3, and therefore Theorem 1, by proving the following three statements.

1) All jobs can be scheduled.
2) The processing times of two jobs from the same class never overlap.
3) The latest completion time of a job is given by 5/3.

The algorithm closes only machines with total load of at least 1. Since the total load of all jobs is bounded by \( m \), when attempting to schedule the last class, there has to exist a non closed machine. The algorithm only potentially closes a machine with load less than 1 in step 2 when a class \( C_{b_2} \) is split into two parts. Let \( C_{b_2} \) be the class already on the machine and \( c_1 \) and \( c_2 \) be the parts of the class the algorithm tries to schedule in this step, such that \( p(c_1) \geq p(c_2) \). Since the class was split in two by the algorithm it holds that \( p(c_{b_2}) + p(c_1) + p(c_2) > 5/3 \). Furthermore, since \( p(c_1) + p(c_2) \leq 1 \) and \( p(c_1) \geq p(c_2) \) it holds that \( p(c_2) \leq 1/2 \) and hence \( p(c_{b_2}) + p(c_1) > 7/6 \). Hence that closed machine has a load of at least 1.

2) Again, the only time one class is scheduled on more than one machine is step 2. When placing the two parts, they never overlap, since they have a processing time of at most 1 and one of the parts starts at 0 while the other ends at 5/3. The algorithm does not generate any overlapping by shifting jobs already on the machine, since those have to originate from classes in \( C_{b_2} \), which each got placed on an individual machine.

3) After step 1 all machines have a load of at most 1, since each class has a total processing time of at most 1. In step 2, we only add an entire class if the total load is bounded by 5/3. If a class is split, the part that is added has a total processing time of at most 2/3. Since before adding this part the machine had a load of at most 1, the load of the closed machine is bounded by 5/3.

The existence and correctness of the algorithm proves Theorem 1.

III. A 3/2-APPROXIMATION

In this section we introduce the more involved algorithm hinted at earlier. While the general idea is similar, finding a lower bound \( T \) for the makespan and then placing classes depending on included big jobs and total processing time, the steps are a lot more granular. We first give a 3/2-approximation algorithm for instances without jobs of size bigger than \( \frac{3}{4}T \). After that we introduce a second 3/2-approximation algorithm that places classes with jobs of size bigger than \( \frac{3}{4}T \) on distinct machines and fills them with other jobs in a clever way such that we can reuse the first algorithm for the remaining classes.

Theorem 2. There exists an algorithm that for any given instance \( I \) of MSRS finds a schedule with makespan bounded by \( 2OPT \) in \( O(n + m \log(m)) \) steps.

In the following let us assume that we have scaled the instance such that \( OPT = 1 \). In order to provide a 3/2-approximation algorithm, we consider four different types of jobs. We split the jobs of a given instance into huge jobs \( J_H = \{ j \in J \mid p_j > 3/4 \} \), big jobs \( J_B = \{ j \in J \mid p_j \in [1/2, 3/4] \} \), medium jobs \( J_M = \{ j \in J \mid p_j \in [1/4, 1/2] \} \), and all residual jobs (with a processing time of at most 1/4) which we refer to as small jobs. Furthermore, turning to the classes \( C \) we define the subset \( C_H = \{ c \in C : \lfloor J_H \cap c \rfloor = 1 \} \) of all classes containing a huge job, the subset \( C_B = \{ c \in C : \lfloor J_B \cap c \rfloor = 1 \} \) of all classes containing a big job, the subset \( C_{2/3/4} = \{ c \in C : p(c) \geq 3/4 \} \) of all classes with a total processing time of at least 3/4, and the subset \( C_{1/2/3/4} = \{ c \in C : p(c) \in (1/2, 3/4) \} \) of all classes with a total processing time in \( (1/2, 3/4) \).

Lemma 2. For any normalized optimal schedule and the corresponding partition of \( C \) into \( C_H, C_B, C_{2/3/4} \setminus (C_H \cup C_B) \) and \( C \setminus C_{2/3/4} \) it holds that

\[
|C_H| + \max \{ |C_B|, \lfloor \frac{1}{2} (|C_B| + |C_{2/3/4} \setminus (C_H \cup C_B)) \rfloor \} \leq m.
\]

Furthermore in \( O(n + m \log(m)) \) steps it is possible to find the smallest value \( T \) with

\[
\max \{ \frac{1}{n} p(J), \max_{c \in C} p(c), p(j_m) + p(j_{m+1}) \} \leq T \leq OPT
\]

such that the instance scaled by \( 1/T \) fulfills these properties (\( j_m \) and \( j_{m+1} \) are the jobs with \( m \)-th and \( (m+1) \)-st largest processing time).

The intuition for the proof of this Lemma is to study the time corridor of “height” \( 1/2 \) between time \( 1/4 \) and \( 3/4 \) over all \( m \) machines; apparently, each class in \( C_H \cup C_{2/3/4} \setminus (C_H \cup C_B) \) contributes with a load of at least \( 1/4 \) to the total load which is scheduled inside of this corridor.

Before presenting the algorithms we give two Lemmas stating the possibility to partition some classes into two parts that will be scheduled on two different machines.

Lemma 3. Let \( c \in C_{2/3/4} \) and \( \max_{j \in c} p_j \leq 3/4 \). Then \( c \) can be partitioned into two parts \( \hat{c} \) and \( \check{c} \) with \( p(\hat{c}) \leq 1/2 \) and \( p(\check{c}) \leq 3/4 \) and \( p(\hat{c}) \leq p(\check{c}) \). Furthermore, if \( \max_{j \in c} p_j \leq 1/2 \), it holds that \( p(\check{c}) \in (1/4, 1/2] \) or \( p(\hat{c}) \in (1/4, 1/2] \).

Lemma 4. Let \( c \in C \) with \( p(c) \in (1/2, 3/4) \) and \( \max_{j \in c} p_j \leq 1/2 \). Then \( c \) can be partitioned into two parts \( \hat{c} \) and \( \check{c} \) with \( p(\hat{c}) \leq p(\check{c}) \leq 1/2 \) and \( 1/4 < p(\check{c}) \).

In the following, we will present two algorithms. The first can handle instances with classes that do not possess an item with processing time larger than \( 3/4 \). This algorithm will be used as a subroutine for the second algorithm, which can handle all instances.
A. Algorithm for Instances without Huge Jobs

Remark 1. Define a simplified instance as follows: Iterate all classes \( c \in \mathcal{C} \)

- If \( p(c) > 3/4 \) partition it into parts \( \hat{c} \) and \( \check{c} \) with \( p(\hat{c}) \leq p(\check{c}) \leq 3/4 \) and \( p(\hat{c}) \leq 1/2 \). Introduce for each part a new job with processing time \( p(\check{c}) \) and \( p(\hat{c}) \).
- If \( p(c) \leq 3/4 \) introduce one job of size \( p(c) \).

Now each class contains at most two jobs.

Here we give an algorithm for instances with \( |\mathcal{C}_H| = 0 \). We assume that the instance was scaled by a value \( 1/T \) and the classes are categorized as described earlier. The main idea is to repeatedly take combinations of classes with specific parameters which conveniently fulfill one, two or three machines, without opening additional ones. Fill in this case means that the average load of full machines is in \([1, 3/2]\). At some point in the algorithm we reach a state where all residual classes have total load at most 1/2. Those can be scheduled greedily, by placing full classes on residual machines, until a machine has load at least 1.

Since we repeatedly have to refer to the jobs which have not been scheduled, we introduce the notation of \( \mathcal{C}_X \subset \mathcal{C}_X \) to denote the subset of classes that have not been scheduled at the described step for any class specifier \( X \). Note that in the beginning of the algorithm, we have \( \mathcal{C}_X = \mathcal{C}_X \) for all the sets. Furthermore, the algorithm will close some machines during the construction of the schedule and will not add jobs to closed machines. We denote the set of closed machines as \( M_c \).

B. Algorithm: Algorithm_no_huge

Step 1. Partition every class \( c \in \mathcal{C}_{>3/4} \) into \( \hat{c}, \check{c} \subseteq c \) with \( p(\hat{c}) \leq p(\check{c}) \leq 3/4 \) and \( p(\check{c}) \leq 1/2 \).

Step 2. While \( |\mathcal{C}_{(1/2,3/4)}| \geq 2 \) Take \( c_1, c_2 \in \mathcal{C}_{(1/2,3/4)} \). Schedule \( c_1 \) and \( c_2 \) on one machine.

Claim. The load of each machine closed in this step is in \((1, 3/2)\). After this step it holds that \( |\mathcal{C}_{(1/2,3/4)}| \leq 1 \), the partial schedule is feasible, and the total load of closed machines \( M_c \) is at least \( |M_c| \).

Step 3. While \( |\mathcal{C}_{(3/4)}| \geq 4 \) Take \( c_1, c_2, c_3, c_4 \in \mathcal{C}_{(3/4)} \). On the first machine schedule \( \hat{c}_1 \) and \( \check{c}_2 \), such that \( \hat{c}_1 \) starts at 0 and \( \check{c}_2 \) ends at 3/2. On the second machine schedule \( \hat{c}_1 \) and \( c_3 \), such that \( \hat{c}_1 \) ends at 3/2 and starts after 1. On the third machine schedule \( \check{c}_2 \) and \( c_4 \), such that \( \check{c}_2 \) starts at 0 and ends before 1/2 followed by \( c_4 \), see Fig. 2b for an example. Close all three machines.

Claim. After this step \( |\mathcal{C}_{(1/2,3/4)}| \leq 1 \) and \( |\mathcal{C}_{(3/4)}| \leq 3 \), the partial schedule is feasible, and the total load of closed machines \( M_c \) is at least \( |M_c| \). Furthermore, all scheduled jobs are finished by 3/2.

Step 4. If \( |\mathcal{C}_{(3/4)}| \geq 2 \) and \( |\mathcal{C}_{(1/2,3/4)}| = 1 \): Take \( c_1, c_2 \in \mathcal{C}_{(3/4)} \) and \( c_3 \in \mathcal{C}_{(1/2,3/4)} \). Schedule \( c_3 \) on the first machine, followed by \( \hat{c}_1 \) such that it ends at 3/2. Schedule \( c_1 \) on the second machine followed by the jobs from \( c_2 \) and close both machines, see Fig. 2c for an example.

Claim. After this step it holds that \( |\mathcal{C}_{(1/2,3/4)}| = 0 \) and \( |\mathcal{C}_{(3/4)}| \leq 3 \) or it holds that \( |\mathcal{C}_{(1/2,3/4)}| = 1 \) and \( |\mathcal{C}_{(3/4)}| \leq 1 \). This implies that \( |\mathcal{C}_{(1/2)}| \leq 3 \) after this step and that \( \mathcal{C}_{(1/2)} \) contains at most one class with total processing time less than 3/4. Furthermore, the partial schedule is feasible, the total load of closed machines \( M_c \) is at least \( |M_c| \), and no scheduled job finishes after 3/2.

Depending on the size of \( |\mathcal{C}_{(1/2)}| \) the algorithm chooses one of three procedures:

Step 5. If \( |\mathcal{C}_{(1/2)}| \leq 1 \): Place this class \( c \) on one machine. Fill this machine and the residual machines greedily with the residual classes in \( \mathcal{C}_{\leq 1/2} \).

Claim. After this step it either holds that \( 2 \leq |\mathcal{C}_{(1/2)}| \leq 3 \) or all jobs have been scheduled feasibly with no job finishing after 3/2.

Step 6. If \( |\mathcal{C}_{(1/2)}| = 2 \): Let \( \mathcal{C}_{(1/2)} = \{ c_1, c_2 \} \) with \( p(c_1) \geq p(c_2) \). We know that \( p(c_1) \geq 3/4 \).

1) If \( p(c_2) \leq 3/4 \):
   a) If \( p(c_1) + p(c_2) \leq 3/2 \): Schedule both on one machine (with \( c_1 \) starting at 0 and \( c_2 \) ending at 3/2), close it, and continue greedily with the residual jobs.
   b) If \( p(c_1) + p(c_2) > 3/2 \): Place \( c_2 \) on one machine followed by \( \check{c}_1 \) and close the machine. Place \( \hat{c}_1 \) on the
next machine and continue greedily with the residual jobs in $C_{<1/2}$.

2) If $p(c_2) \geq 3/4$:
   
a) If $p(c_1) + p(c_2) \leq 1$: Schedule $c_2$ followed by $c_1$ on one machine and close it. Start $c_1$ at 0 on the next machine and continue greedily with the residual jobs in $C_{<1/2}$.
   
b) If $p(c_1) + p(c_2) > 1$: Then place $c_1$ and $c_2$ on one machine such that $c_1$ starts at 0 and $c_2$ ends at $3/2$. Place $c_2$ at the bottom and $c_1$ at the top of the next machine. Continue greedily with the residual classes $C_{<1/2}$. Start placing them between $c_2$ and $c_1$ until the load of that machine is at least 1 and then continue with the empty machines.

Claim. After this step it either holds that $|C_{>1/2}| = 3$ or all jobs have been scheduled feasibly with no job finishing after $3/2$.

Step 7. If $|C_{>1/2}| = 3$: Then $C_{>1/2} = C_{2/3/4}$. Let $C_{2/3/4} = \{c_1, c_2, c_3\}$.

1) If there exists an $i \in \{1, 2, 3\}$ such that $c_i \leq 1/2$: Let w.l.o.g. $c_1 \leq 1/2$. On the first machine schedule $c_1$ followed by all the jobs from $c_2$. On the next machine schedule all the jobs from $c_3$ and the job $c_1$ such that it ends at $3/2$ and close both machines. Greedily schedule the jobs in $C_{<1/2}$ on the non-closed machines.

2) If $c_i > 1/2$ for all $i \in \{1, 2, 3\}$: Place $c_1$ and $c_2$ on one machine such that $c_1$ starts at 0 and $c_2$ ends at $3/2$.

   a) If $p(c_1) + p(c_2) + p(c_3) \leq 3/2$: On the next machine place $c_2$ followed by $c_3$ and $c_1$ and let $c_1$ end at $3/2$. Close both machines.

   b) If $p(c_1) + p(c_2) + p(c_3) > 3/2$: Then w.l.o.g. $p(c_1) > 1/4$ and we place $c_1$ and $c_2$ on the next machine, such that $c_1$ ends at $3/2$. Close both machines. On the next machine place $c_2$ such that it starts at 0.

Greedily schedule the jobs in $C_{<1/2}$ on the non-closed machines.

Claim. After this step, all scheduled jobs are finished by $3/2$ and the schedule is feasible.

C. Algorithm for the General Case

Now we present the above-mentioned algorithm that can handle any instance of the problem and uses the previous algorithm in a subroutine. More specifically, this algorithm places all classes which contain a huge job on a separate machine and fills those machines with jobs from other classes. This is done by working through different combinations of classes until we reach a point where we can handle the remaining classes and machines as a separate problem instance, at which point the previous algorithm is used. As before we assume that the instance is scaled by a value $1/T$ and the classes are categorized as described earlier.

We denote the set of machines that received a huge class as $M_H$ and the subset of open machines that received a huge class as $M_{op}$. Open machines are all machines which were not explicitly closed. Moreover, the set of unused machines is denoted as $M_u$. We keep the following invariant of the remaining instance over the whole algorithm.

Invariant 1. The total load of unscheduled jobs and jobs placed on open machines is upper bounded by the number of open machines, i.e., $p(M_H) + p(C) \leq |M_u| + |M_{op}|$, and in each step the cardinality of the set of unused machines $M_u$ is bounded analogous to Lemma 2: $|M_u| \geq \max\{|C_B|, |(C_B)_{>2/3} M_{<1/2}\}_{/2}\}$.

The algorithm for the general case works as follows:

D. Algorithm: Algorithm_3/2

Step 1. Combine specific jobs of the same class into one job. The simplification is done as follows: Iterate all classes $c \in C$

   - If $c \in C_H$ combine all jobs in $c$ to one huge job.

   - Else if $p(c) > 3/4$ partition it into parts $c$ and $c$ with $p(\hat{c}) = p(\check{c}) \leq 3/4$ and $p(\hat{c}) \leq 1/2$. Introduce for each part a new job with processing time $p(\hat{c})$ and $p(\check{c})$, see Lemma 3.

   - Else if $c \in C_{(1/2,3/4)} \cap C_B$ partition it into $\check{c}$ and $\hat{c}$, such that $\hat{c}$ contains the largest job and $\hat{c}$ contains the rest.

   - Else if $c \in C_{(1/2,3/4)} \setminus C_B$ partition it into $\check{c}$ and $\hat{c}$ with $p(\check{c}) \leq p(\hat{c}) \leq 1/2$, see Lemma 4.

   - Else if $p(c) \leq 1/2$ introduce one job of size $p(c)$.
This partition is feasible and every solution for this simplified instance will still be a solution for the original instance.

Step 2. For each $c \in C_H$: Open one new machine and assign class $c$ to it. As stated above, we denote the set of these machines containing a class from $C_H$ as $M_H$ and the subset of currently open machines as $\bar{M}_H$. Close all the machines that have load exactly 1.

Claim. After this step, there are $|M_H|$ open machines with load in $(3/4, 1)$, $|C_H| = 0$, and Invariant 1 holds.

Step 3. Assign classes $c_i$ with $p(c_i) \leq 1/2$ greedily to machines $\bar{M}_H$ and close each machine with load at least 1. Continue until either no machines in $\bar{M}_H$ with load less than 1 is left, or no class with load at most 1/2 is left. If $|M_H| = 0$, continue with Algorithm_no_huge on the residual instance.

Claim. After this step either all jobs are scheduled feasibly or it holds that $|M_H| \geq 1$ and $|C_{\leq 1/2}| = 0$. Furthermore, the partial schedule is feasible, all scheduled jobs are finished by 3/2, and Invariant 1 holds.

Step 4. While $|M_H| \geq 2$ and $|C_{(1/2, 3/4)} \setminus C_B| \geq 1$: Take $m_1, m_2 \in M_H$, $c \in C_{(1/2, 3/4)} \setminus C_B$. Shift the huge job on $m_2$ up such that it ends at 3/2 and starts at or after 1/2. Schedule $c$ on $m_1$ such that it ends at 3/2, schedule $\bar{c}$ on $m_2$ starting at 0 and close both machines. If $|M_H| = 0$, continue with Algorithm_no_huge on the residual instance.

Claim. After this step either all jobs are scheduled feasibly or one of the following two conditions holds: $|M_H| = 1$ and $|C_{\leq 1/2}| = 0$, or $|M_H| \geq 2$ and $|C \setminus (C_B \cup C_{\geq 3/4})| = 0$. Furthermore, the partial schedule is feasible, all scheduled jobs are finished by 3/2 and all machines not in $\bar{M}_H$ are either closed or empty, and Invariant 1 holds.

Step 5. If $|M_H| = 1$:

- If there exists $c \in \bar{C} \setminus C_B$: Choose $c' \in \{\bar{c}, c\}$ with $c' \in (1/4, 1/2]$. Schedule $c'$ on the last open machine denoted by $m_0$. Use Algorithm_no_huge to schedule the residual instance, including the job $c' \in c \setminus c'$.

"Rotate" the load on $m_0$, such that $c'$ does not overlap with $c''$.

- If $\bar{C} \setminus C_B$ is empty: Assign all the residual classes to an individual machine.

Claim. After this step all jobs have been scheduled feasibly or $|M_H| \geq 2$ and $|C \setminus (C_B \cup C_{\geq 3/4})| = 0$. Additionally the partial schedule is feasible, all scheduled jobs are finished by 3/2, and Invariant 1 holds.

Step 6. While $|M_H| = 1$, $|C_{(1/2, 3/4)} \cap C_B| \geq 1$, and $|C_{\geq 3/4}| = 1$: Take $m_1 \in M_H$, $b \in C_{(1/2, 3/4)} \cap C_B$ and $c \in \bar{C}_{\geq 3/4}$. Open one new machine $m_2$. Schedule $\bar{c}$ on $m_1$ such that it ends at 3/2. Schedule $\bar{c}$ on $m_2$ such that it starts at 0 and ends before 3/4. Schedule $b$ at $m_2$ such that it ends at 3/2. Close both machines. If $|M_H| = 0$, continue with Algorithm_no_huge on the residual instance.

Claim. After this step all jobs are scheduled feasibly or $|M_H| \geq 1$ and $|C \setminus (C_{(1/2, 3/4)} \cap C_B)| = 0$ or $|C \setminus C_{\geq 3/4}| = 0$. Furthermore, all jobs are scheduled feasibly in this step, all scheduled jobs are finished by 3/2, and Invariant 1 holds.

Step 7. If $|C_{(1/2, 3/4)} \cap C_B| \neq 0$, open one machine for each of these classes.

Claim. After this step all jobs are feasibly scheduled or it holds that $|M_H| \geq 1$ and all residual classes have a total processing time of at least 3/4, all scheduled jobs are finished by 3/2, and Invariant 1 holds.

Step 8. While $|M_H| \geq 2$ and $|C_{\geq 3/4}| \geq 2$: Take $m_1, m_2 \in M_H$, $c_1, c_2 \in C_{\geq 3/4}$ starting with the classes in $C_B$. Shift all jobs on $m_2$ to the top, such that the last job ends at 3/2. Schedule $\bar{c}_1$ on $m_1$ as one block that ends at 3/2 and all the jobs from $\bar{c}_2$ as one block on $m_2$ that starts at 0. Open one more machine $m_3$ where we start the jobs from $\bar{c}_1$ at 0 and let the last job from $\bar{c}_2$ end at 3/2. Close all three machines $m_1, m_2, m_3$. If $|M_H| = 0$, continue with Algorithm_no_huge on the residual instance.

Claim. After this step all jobs are scheduled or it holds that either $|M_H| = 1$ or $|C_{\geq 3/4}| \leq 1$. Furthermore $|C \setminus C_{\geq 3/4}| = 0$ and in each iteration the partial schedule is feasible, all scheduled jobs are finished by 3/2, and Invariant 1 holds.
Step 9. If $|\hat{M}_H| \geq 2$ or $|\hat{C} \setminus C_B| = 0$, open one machine for each of the remaining classes.

Claim. After this step either all jobs are scheduled or it holds that $|\hat{M}_H| = 1$, $|\hat{C} \setminus \hat{C}_{0/4}| = 0$, $\hat{C} \setminus C_B \neq \emptyset$, the partial schedule is feasible, all scheduled jobs are finished by $3/2$, and Invariant 1 holds.

Step 10. If $|\hat{M}_H| = 1$, take $c \in \hat{C} \cap C_B$. It holds that $p(c) \geq 3/4$ and there exists $c' \in \{\acute{c}, \grave{c}\}$ with $p(c') \in (1/4, 1/2]$. Place $c'$ on $m_0 \in \hat{M}_H$. Continue with Algorithm_no_huge to schedule the residual jobs including the job $c'' \in \hat{C} \setminus \{c'\}$. Rotate the load on $m_0$ such that $c''$ does not overlap with $c'$.

Claim. After this step all jobs are scheduled feasibly and all scheduled jobs are finished by $3/2$.

IV. APPROXIMATION SCHEMES

Here we consider approximation schemes for the problem at hand. We present two results:

Theorem 3. There is an EPTAS for MSRS if either the number $m$ of machines is constant or $\lceil \varepsilon m \rceil$ additional machines may be used, i.e., resource augmentation is allowed.

To achieve these results, we follow a framework that was introduced in [13] and also used in [25]. In particular, we consider a simplified version of the problem and prove the existence of a certain well-structured solution with only bounded loss in the objective compared to an optimal solution. The problem of finding such a solution can then be formulated as an integer program (IP) of a particular form. This IP can be solved efficiently using N-fold integer programming algorithms. Furthermore, we guarantee that the solution for the simplified problem can be used to derive a solution for the original one with only little loss in the objective value. The main challenge lies in the design of the well-structured solution and the proof of its existence. This also causes the limitations of our result: A certain group of jobs may cause problems in the respective construction, and to deal with them we either use a more fine-grained approach, yielding a polynomial running time if $m$ is constant, or place the respective jobs on (few) additional machines using resource augmentation.

A. Simplification

We use the standard technique (see [26]) of applying a binary search framework to acquire a makespan guess $T$. The goal is then to either find a schedule of length $(1 + O(\varepsilon))T$ or correctly report that no schedule of length $T$ exists. We introduce parameters $\delta$ and $\mu$ and call a job $j$ big, medium, or small, if $p_j \in (\delta T, T)$, $p_j \in (\mu T, \delta T]$, or $p_j \in (0, \mu T]$, respectively. Furthermore, we assume $\varepsilon < 0.5$. We set $\mu = \varepsilon^2\delta$ and choose $\delta$ depending on the instance and on whether we consider the case with a constant number of machines or not.

1) Simplifications (i) for Fixed $m$ and (ii) $m$ Part of Input: We choose (i) $\delta \in \{\varepsilon, \varepsilon^2, \ldots, \varepsilon^{2m}/\varepsilon\}$ or (ii) $\delta \in \{\varepsilon, \varepsilon^2, \ldots, \varepsilon^{4}/\varepsilon\}$ such that:

1) The overall size of jobs $j$ with size $p_j \in (\mu T, \delta T]$ is at most (i) $\varepsilon T$ or (ii) $\varepsilon^2 m T$.

2) The overall size of jobs $j$ with size $p_j \leq \delta T$ from classes in which these jobs have overall size in $(\mu T, \delta T]$ is at most (i) $\varepsilon T$ or (ii) $\varepsilon^2 m T$.

Let $I$ be the input instance and $I_1$ the instance where (i) we remove all medium jobs or (ii) remove all medium jobs from classes including at most $\varepsilon T$ medium load and the entire classes containing at least $\varepsilon T$ medium load.

Lemma 5. Let (i) $m$ be a constant or (ii) $m$ be part of the input. If there is a schedule with makespan $T'$ for $I$, then there is also a schedule with makespan $T'$ for $I_1$; and if there is a schedule with makespan $T'$ for $I_1$, then there is also a schedule with makespan $T' + \varepsilon T$ for $I$ ((iii) using at most $\lceil \varepsilon m \rceil$ additional machines).

The first direction is obvious. For the other direction in the case of (i) note that the overall size of the medium jobs is upper bounded by $\varepsilon T$ and hence we can place all of them at the end of the schedule on some arbitrary machine. For the case of (ii), the proof idea is to use two greedy procedures placing the removed medium jobs from classes including at most $\varepsilon T$ medium load at the end of the schedule and the remaining removed medium jobs on the additional machines.

2) Layered Schedules: A $(T', L')$-schedule is a schedule with makespan at most $T'$ and at least $L'$ idle time throughout the schedule. Let $I_2$ be the instance we get if we remove all the small jobs from classes in which these jobs have overall size of at most $\delta T$ from $I_1$.

For some positive number $\xi$, we call a schedule $\xi$-layered, if the processing of each job starts at a multiple of $\xi$. The time between two such multiples is called a layer and the corresponding time on a single machine a slot. Let $I_3$ be the instance we get if we round the processing times of the big jobs in $I_2$ and replace the remaining small jobs with placeholders. In particular, let $p_j' = \lfloor p_j/(\xi \delta T) \rfloor \xi \delta T$ be the rounded size for each big job $j$. Furthermore, for each class $c \in C$ with $p_c(c) = \sum_{j \in c, p_j \leq \mu T} p_j > \delta T$, we remove the small jobs and introduce $\lceil p_c(c)/(\xi \delta T) \rceil$ new jobs with processing time $\varepsilon \delta T$ each. Let $L$ be the overall size of the jobs removed in this step.

Lemma 6. If there is a schedule with makespan $T'$ for $I_1$, then there is also an $\varepsilon \delta T$-layered $((1 + 2\varepsilon)T', L)$-schedule for $I_3$, and if there is an $\varepsilon \delta T$-layered $(T', L)$-schedule for $I_3$, then there is also a schedule with makespan $(1 + \varepsilon)T' + \varepsilon T$ for $I_1$.

This is the main technical lemma for the approximation schemes and we very briefly discuss the proof idea. For the first part, the schedule for $I_3$ is stretched and the big jobs can be rounded and placed in the layers using straightforward arguments. Regarding the small jobs a carefully crafted flow network is used that has a feasible flow due to the original placement of the jobs. Then flow integrality is utilized in order to place the placeholder small jobs in the layered schedule without conflicts. If there is a layered schedule for $I_3$, on the
other hand, constructing the schedule for $I_i$ boils down to a series of carefully designed greedy procedures used to place the removed or replaced jobs.

B. Integer Program

To find a layered schedule, we utilize an IP approach. The corresponding IP is essentially a module configuration IP as introduced in [13] but, for the sake of simplicity, we diverge from the notation in that paper. We set $T' = (1 + 2\varepsilon)T$ and search for an $\varepsilon T$-layered $(T', L)$-schedule for $I_3$. We introduce some notation. Let $\Xi = \{\ell \in \mathbb{Z}_{\geq 0} \mid (\ell - 1)\varepsilon T \leq (1 + 2\varepsilon)T\}$ be the set of layers, $P$ the set of distinct processing times in $I_3$, and $n_p^{(c)}$ the number of jobs of size $p$ in class $c$ for each $p \in P$ and $c \in C$. Furthermore, we define a (time) window as a pair $(\ell, p) \in \Xi \times P$ of a starting layer $\ell$ and a processing time $p$, and a configuration $K$ as a selection of windows $[0, 1]^W$ such that no two conflicting windows are chosen, i.e., $\sum_{(\ell, p) \in W_i} K_{\ell, p} \leq 1$ for each layer $\ell'$. The set of configurations is denoted as $\mathcal{K}$, the set of windows as $W$ and the set of windows intersecting layer $\ell$ as $W_\ell$. A window $(\ell, p)$ intersects $\forall\ell' \in \Xi \sum_{\ell' \in \Xi} W_{\ell'}$. A unique resource $p/\varepsilon T$ many succeeding layers starting with $\ell$.

**Remark 2.** $|P| \in O(1/(\varepsilon T)), |\Xi| \in O(1/(\varepsilon T)), |W| \in O(1/(\varepsilon T)), |\mathcal{K}| \in 2^{O(1/(\varepsilon T))}$.

In the IP, we have a variable $x_K \in \{0, \ldots, m\}$ for each $k \in \mathcal{K}$, a variable $y_{\ell, p}^{(c)} \in \{0, \ldots, n\}$ for each class $c \in C$ and window $(\ell, p) \in W$, as well as the following constraints:

\begin{align*}
\sum_{K \in \mathcal{K}} x_K &= m \quad (1) \\
\sum_{K \in \mathcal{K}} K_{\ell, p} x_K &= \sum_{c \in C} y_{\ell, p}^{(c)} \forall (\ell, p) \in W \quad (2) \\
\sum_{\ell \in \Xi} y_{\ell, p}^{(c)} &= n_p^{(c)} \forall c \in C, p \in P \quad (3) \\
\sum_{(\ell', p) \in W_{\ell'}} y_{\ell', p}^{(c)} &\leq 1 \forall c \in C, \ell \in \Xi \quad (4)
\end{align*}

The variables $y_{\ell, p}^{(c)}$ are used to reserve time windows for the placement of jobs belonging to class $c$. (3) guarantees that the correct number is chosen, and due to (4) placing the respective jobs in the windows will not create conflicts. Furthermore, the variables $x_K$ are used to chose $m$ configurations (due to (1)). Each such configuration corresponds to a scheduling pattern on one of the $m$ machines. In particular, a configuration is by definition a selection of non-overlapping time windows and in (2) we make sure that these configurations cover the windows selected for the placement of jobs. Hence, it is easy to construct a solution for the IP given a $\varepsilon T$-layered $(T', L)$-schedule and vice-versa yielding:

**Lemma 7.** There exists an $\varepsilon T$-layered $(T', L)$-schedule for $I_3$, if the above IP is feasible.

C. Algorithm and Analysis

Summing up, we use a binary search framework to get the makespan guess $T$, perform the simplification steps described in Section IV-A, formulate and solve the described IP, construct a schedule from the IP solution, and transform it into a schedule for the original instance. For the given makespan guess $T$, we thus find a schedule with makespan at most $(1 + \varepsilon)(1 + 2\varepsilon)T + 2\varepsilon T = (1 + O(\varepsilon))T$ or, if the IP is not feasible, correctly report that a schedule with makespan $T$ does not exists.

Regarding the runtime, it is easy to see that the critical step lies in solving the IP, since all the other ones mostly involve simple changes of the instance and fast greedy procedures that obviously run in polynomial time. Hence, we take a closer look at the IP and again essentially apply the approach introduced in [13], i.e., solving it via $N$-fold integer programming. The slightly restricuted IP can be solved in time $2^O(1/(\varepsilon T))$ doubly exponential in $1/\varepsilon$ where the former factor can be bounded by $2^O(1/\varepsilon^2 T)$ and $2^O(1/\varepsilon)$ for arbitrary $m$ and fixed $m$, respectively.

V. INAPPROXIMABILITY RESULTS

We consider the case in which each job may need more than a single resource. Let us assume that we have a set $R$ of resources and each job $j$ needs some subset $R(j)$ in order to be processed. The classes then correspond to subsets of resources $R \subseteq R$ with $J(R) = \{j \in J \mid R(j) = R\}$. We could adapt an APX-hardness result from [11] by recreating their conflict graph with resources. This is done by creating a resource $r_e$ per edge $e = \{u, v\}$ and letting jobs $u$ and $v$ require that resource. This reduction needs 2 machines, job sizes in 1, 2, 3, 4 but roughly as many distinct resources per job, as there are jobs. Subsequently, we give a new unrelated reduction for an instance of the problem with a constant bound on the number of distinct resources per job. 

**Theorem 4.** There is no $5/4 - \varepsilon$ approximation algorithm with $\varepsilon > 0$ for the MSRS with multiple resources per job if $P \neq \emptyset$. This holds true, even if no job needs more than 3 resources $(\forall j \in J : |R(j)| \leq 3)$ and all jobs have processing time 1, 2 or 3 $(\forall j \in J : |R(j)| \in \{1, 2, 3\})$. Furthermore, this also holds when the number of machines is unlimited.

We show this by giving a reduction from the NP-hard MONOTONE 3-SAT-(2,2) problem [27], which is a satisfiability problem with the following restrictions: The boolean formula is in 3CNF, each clause contains either only unnegated or negated variables and each literal appears in exactly 2 clauses (and every variable in exactly 4 clauses). Note here, that we only use the bounded occurrence of literals, not the monotony.

In the following we write that two (or more) jobs $j$ and $j'$ "share a resource $r"$, which means that $r \in R(j)$ and $r \in R(j')$ and for all other jobs $j^*, j^* \neq j$ and $j^* \neq j$, $r \notin R(j^*)$. Let $\phi$ be the given formula and $C, X$ the sets of clauses and variables in $\phi$, respectively. We start by creating a dummy structure that we can anchor jobs to by using shared resources (see Fig. 4a for the general structure). Create $|C|$ many pairs of dummy jobs $j^A_i, j^B_i$ with $p(j^A_i) = 3, p(j^B_i) = 1$, which share a unique resource $A_i$. Furthermore, $j^A_i, j^B_i$ share a unique resource $A_{i-1}+1$. Create $|X|$ many pairs of dummy jobs $j^A_i, j^B_i$ with $p(j^A_i) = p(j^B_i) = 2$, which share a unique resource $B_i$. Furthermore, $j^A_i, j^B_i$ share a unique resource $B_{i-1}+1$. Lastly, $j^A_i, j^B_i$ share a unique resource $A_{i-1}B_i$. For every $x_i \in X$
create three variable jobs \( j_x, j_{x'}, j_{x''} \) and \( j_{dx} \), which all share a resource \( X_{x'} \), moreover \( j_{dx} \) and \( j_{x'}^B \) share a resource \( B_{x'} \).

For every \( c_i \in C \) with \( c_i = \{ x_1^i, x_2^i, x_3^i \} \) create four clause jobs \( j_{x_1'}^C, j_{x_2'}^C, j_{x_3'}^C \) and \( j_{x''}^C \) which all share a resource \( C_{c_i} \). For every unnegated literal \( x_i \) find the two clauses \( c' \) and \( c'' \) in which it is contained. Let \( j_{x_i}^C \) and \( j_{x_i}^{C'} \) be the two corresponding clause jobs created for \( x_i \) in \( c' \) and \( c'' \) respectively, then \( j_x \) shares a unique resource with \( j_{x_i} \) and another unique resource with \( j_{x_i}^C \) (see connections from \( j_w \) to its two occurrences in clauses in Fig. 4b). Repeat this process for \( x_i \) and its negated occurrences. Furthermore \( j_{x_i}^C \) and \( j_{x_i}^{C'} \) shares a resource \( A_{c_i} \).

Finally, we set the number of machines to \( 2|C| + 2|X| \) (One could also give an unlimited number of machines, as the resources limit the number of concurrently usable machines either way).

**Lemma 8.** There is an optimal schedule with makespan 4 iff there is a satisfying assignment for MONOTONE 3-SAT-\((2,2)\). Otherwise the optimal schedule has a makespan of 5.

**Remark 3.** By following a similar construction one can show the same inapproximability result for unit jobs and 5 or less resources per job. Furthermore, it is possible to give a \( 4/3 - \epsilon \) inapproximability result for the problem by giving a reduction from the NAE-3SAT problem. That reduction uses unit jobs, but a non-constant number of resources per job.

**VI. CONCLUSION**

In this paper, we did greatly improve the state of the art regarding the approximability of MSRS. There are several interesting avenues emerging for further investigations. Firstly, there is the question of whether a PTAS for MSRS without resource augmentation can be achieved. It seems plausible that the approximation schemes results of the present work could be further refined to reach this goal. For the case with only a constant number of machines, on the other hand, an FPTAS is not ruled out at this point.

Moreover, it would be interesting to explore natural extensions of MSRS and, in particular, to investigate for which variants approximation schemes may or may not be feasible. From the negative perspective, we have already provided initial results in this paper. We would like to point out one further question in this direction: Note that MSRS can be seen as a special case of scheduling with conflicts where the conflict graph is a cograph. This problem is known to be NP-hard already for unit size jobs [12] and it would be interesting to explore inapproximability for arbitrary sizes. Regarding the design of approximation schemes, on the other hand, variants where the corresponding conflict graph is a particularly simple cograph may be interesting.

Finally, from a broader perspective, it seems interesting to explore the possibilities of \( N \)-fold IPs and related concepts [28] for scheduling with additional resources.

**REFERENCES**

[1] J. Blazewicz, J. K. Lenstra, and A. H. G. R. Kan, “Scheduling subject to resource constraints: classification and complexity,” Discret. Appl. Math., vol. 5, no. 1, pp. 11–24, 1983. [Online]. Available: https://doi.org/10.1016/0166-218X(83)90012-4

[2] E. B. Edis, C. Oguz, and I. Ozkaran, “Parallel machine scheduling with additional resources: Notation, classification, models and solution methods,” Eur. J. Oper. Res., vol. 230, no. 3, pp. 449–463, 2013. [Online]. Available: https://doi.org/10.1016/j.ejor.2013.02.042

[3] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, M. Sterna, and J. Weglarz, Scheduling under Resource Constraints. Cham: Springer International Publishing, 2019, pp. 475–525. [Online]. Available: https://doi.org/10.1007/978-3-319-99849-7_13

[4] J. Blazewicz, N. Brauner, and G. Finke, “Scheduling with discrete resource constraints,” in Handbook of Scheduling - Algorithms, Models, and Performance Analysis. Chapman and Hall/CRC, 2004. [Online]. Available: https://doi.org/10.1201/9780203489802

[5] E. Hebrard, M. Huguet, N. Jozefowiez, A. Maillard, C. Pralet, and G. Verfaillie, “Approximation of the parallel machine scheduling problem with additional unit resources,” Discret. Appl. Math., vol. 215, pp. 126–135, 2016. [Online]. Available: https://doi.org/10.1016/j.dam.2016.07.003

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
[6] V. A. Strusevich, “Approximation algorithms for makespan minimization on identical parallel machines under resource constraints,” Journal of the Operational Research Society, vol. 0, no. 0, pp. 1–12, 2020. [Online]. Available: https://doi.org/10.1080/01605682.2020.1772019

[7] G. Dósa, H. Kellerer, and Z. Tuza, “Restricted assignment scheduling with resource constraints,” Theor. Comput. Sci., vol. 760, pp. 72–87, 2019. [Online]. Available: https://doi.org/10.1016/j.tcs.2018.08.016

[8] T. Janssen, C. M. F. Swennenhuis, A. Bitar, T. Bosman, D. Gijswijt, L. van Iersel, S. Dauzère-Pérès, and C. Yugma, “Parallel machine scheduling with a single resource per job,” CoRR, vol. abs/1809.05009, 2018. [Online]. Available: http://arxiv.org/abs/1809.05009

[9] T. Janssen, “Optimization in the photolithography bay: Scheduling and the traveling salesman problem,” Ph.D. dissertation, Delft University of Technology, Netherlands, 2019. [Online]. Available: http://resolver.tudelft.nl/uuid:12961f87-eeff-41b5-8688-df28e0ad9860

[10] B. S. Baker and E. G. C. Jr., “Mutual exclusion scheduling,” Theor. Comput. Sci., vol. 162, no. 2, pp. 225–243, 1996. [Online]. Available: https://doi.org/10.1016/0304-3975(96)00031-X

[11] G. Even, M. M. Halldörsson, L. Kaplan, and D. Ron, “Scheduling with conflicts: online and offline algorithms,” J. Sched., vol. 12, no. 2, pp. 199–224, 2009. [Online]. Available: https://doi.org/10.1007/s10951-008-0089-1

[12] H. L. Bodlaender and K. Jansen, “On the complexity of scheduling incompatible jobs with unit-timess,” in Proc. MFCS 1993, ser. Lecture Notes in Computer Science, vol. 711. Springer, 1993, pp. 291–300. [Online]. Available: https://doi.org/10.1007/3-540-57182-5_21

[13] K. Jansen, K. Klein, M. Maack, and M. Rau, “Empowering the configuration-ip: new PTAS results for scheduling with setup times,” Math. Program., vol. 195, no. 1, pp. 367–401, 2022. [Online]. Available: https://doi.org/10.1007/s10107-021-01694-3

[14] M. A. Deppert, K. Jansen, M. Maack, S. Pukrop, and M. Rau, “Scheduling with many shared resources,” CoRR, vol. abs/2210.01523, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2210.01523

[15] P. Kling, A. Mäcker, S. Riechers, and A. Skopali, “Sharing is caring: Multiprocessor scheduling with a sharable resource,” in Proc. SPAA 2017. ACM, 2017, pp. 123–132. [Online]. Available: https://doi.org/10.1145/3087556.3087578

[16] K. Jansen and M. Rau, “Closing the gap for single resource constraint scheduling,” in 29th Annual European Symposium on Algorithms, ESA 2021, September 6–8, 2021, Lisbon, Portugal (Virtual Conference), ser. LIPIcs, vol. 204. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 53:1–53:15. [Online]. Available: https://doi.org/10.4230/LIPIcs.ESA.2021.53

[17] M. Niemeier and A. Wiese, “Scheduling with an orthogonal resource constraint,” Algorithmica, vol. 71, no. 4, pp. 837–858, 2015. [Online]. Available: https://doi.org/10.1007/s00453-013-9829-5

[18] K. Jansen, M. Maack, and M. Rau, “Approximation schemes for machine scheduling with resource (in-)dependent processing times,” ACM Trans. Algorithms, vol. 15, no. 3, pp. 31:1–31:28, 2019. [Online]. Available: https://doi.org/10.1145/3302250

[19] H. Kellerer and V. A. Strusevich, “Scheduling problems for parallel dedicated machines under multiple resource constraints,” Discret. Appl. Math., vol. 133, no. 1-3, pp. 45–68, 2003. [Online]. Available: https://doi.org/10.1016/S0166-218X(03)00433-5

[20] A. Grigoriev and M. Uetz, “Scheduling jobs with time-resource tradeoff via nonlinear programming,” Discret. Optim., vol. 6, no. 4, pp. 414–419, 2009. [Online]. Available: https://doi.org/10.1016/j.disopt.2009.05.002

[21] H. L. Bodlaender, K. Jansen, and G. J. Woeginger, “Scheduling with incompatible jobs,” Discret. Appl. Math., vol. 55, no. 3, pp. 219–232, 1994. [Online]. Available: https://doi.org/10.1016/0166-218X(94)90099-4

[22] S. Das and A. Wiese, “On minimizing the makespan when some jobs cannot be assigned on the same machine,” in Proc. ESA 2017, ser. LIPIcs, vol. 87. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 31:1–31:14. [Online]. Available: https://doi.org/10.4230/LIPIcs.ESA.2017.31

[23] K. Grage, K. Jansen, and K. Klein, “An EPTAS for machine scheduling with bag-constraints,” in Proc. SPAA 2019. ACM, 2019, pp. 135–144. [Online]. Available: https://doi.org/10.1145/3323165.3323192

[24] D. R. Page and R. Solis-Oba, “Makespan minimization on unrelated parallel machines with a few bags,” Theor. Comput. Sci., vol. 821, pp. 34–44, 2020. [Online]. Available: https://doi.org/10.1016/j.tcs.2020.03.013

[25] K. Jansen, A. Lassota, and M. Maack, “Approximation algorithms for scheduling with class constraints,” in SPAA ’20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020. ACM, 2020, pp. 349–357. [Online]. Available: https://doi.org/10.1145/3350755.3400247

[26] D. S. Hochbaum and D. B. Shmoys, “Using dual approximation algorithms for scheduling problems theoretical and practical results,” J. ACM, vol. 34, no. 1, pp. 144–162, 1987. [Online]. Available: https://doi.org/10.1145/7531.7535

[27] A. Darmann and J. Döcker, “On simplified np-complete variants of monotone3-sat,” Discret. Appl. Math., vol. 292, pp. 45–58, 2021. [Online]. Available: https://doi.org/10.1016/j.dam.2020.12.010

[28] F. Eisenbrand, C. Hunkenschröder, K. Klein, M. Kouteký, A. Levin, and S. Onn, “An algorithmic theory of integer programming,” CoRR, vol. abs/1904.01361, 2019. [Online]. Available: http://arxiv.org/abs/1904.01361