Mathematical Programming (2022) 195:367–401
https://doi.org/10.1007/s10107-021-01694-3

FULL LENGTH PAPER
Series A

Empowering the configuration-IP: new PTAS results for scheduling with setup times

Klaus Jansen[1] · Kim-Manuel Klein[1] · Marten Maack[1] · Malin Rau[2]

Received: 13 August 2019 / Accepted: 17 June 2021 / Published online: 11 August 2021

Abstract
Integer linear programs of configurations, or configuration IPs, are a classical tool in the design of algorithms for scheduling and packing problems where a set of items has to be placed in multiple target locations. Herein, a configuration describes a possible placement on one of the target locations, and the IP is used to choose suitable configurations covering the items. We give an augmented IP formulation, which we call the module configuration IP. It can be described within the framework of n-fold integer programming and, therefore, be solved efficiently. As an application, we consider scheduling problems with setup times in which a set of jobs has to be scheduled on a set of identical machines with the objective of minimizing the makespan. For instance, we investigate the case that jobs can be split and scheduled on multiple machines. However, before a part of a job can be processed, an uninterrupted setup depending on the job has to be paid. For both of the variants that jobs can be executed in parallel or not, we obtain an efficient polynomial time approximation scheme (EPTAS) of running time $f(1/\varepsilon) \cdot \mathrm{poly}(|I|)$. Previously, only constant factor approximations of $5/3$ and $4/3 + \varepsilon$, respectively, were known. Furthermore, we present an EPTAS for

This research was partially funded by German Research Foundation (DFG) Project JA 612/20-1. There is a published conference version [18] as well as a preprint version of this work [17].

✉ Marten Maack
mmaa@informatik.uni-kiel.de

Klaus Jansen
kj@informatik.uni-kiel.de

Kim-Manuel Klein
kmk@informatik.uni-kiel.de

Malin Rau
Malin.Rau@inria.fr

[1] Department of Computer Science, University of Kiel, 24118 Kiel, Germany

[2] Department of Computer Science, University of Grenoble Alpes, 38401 Saint Martin, d'Héres cedex, France

a problem where classes of (non-splittable) jobs are given, and a setup has to be paid for each class of jobs being executed on one machine.

Keywords Parallel machines · Setup time · EPTAS · n-fold integer programming

Mathematics Subject Classification 68W25 · 90C10

1 Introduction

In this paper, we present an augmented formulation of the classical integer linear program of configurations (configuration IP) and demonstrate its use in the design of efficient polynomial time approximation schemes for scheduling problems with setup times. Configuration IPs are widely used in the context of scheduling or packing problems in which items have to be distributed to multiple target locations. The configurations describe possible placements on a single location, and the integer linear program (IP) is used to choose a proper selection covering all items. Two fundamental problems, for which configuration IPs have prominently been used, are bin packing and minimum makespan scheduling on identical parallel machines, or machine scheduling for short. For bin packing, the configuration IP was introduced as early as 1961 by Gilmore and Gomory [13], and the recent results for both problems typically use configuration IPs as a core technique, see, e.g., [14,19]. In the present work, we consider scheduling problems and therefore introduce the configuration IP in more detail using the example of machine scheduling.

Configuration IP for Machine Scheduling In the problem of machine scheduling, a set $\mathcal{J}$ of n jobs is given together with processing times p_j for each job j and a number m of identical machines. The objective is to find a schedule $\sigma : \mathcal{J} \rightarrow [m]$ such that the makespan is minimized, that is, the latest finishing time of any job $C_{\max}(\sigma) = \max_{i\in[m]} \sum_{j\in\sigma^{-1}(i)} p_j$. For a given makespan bound, the configurations may be defined as multiplicity vectors indexed by the occurring processing times such that the overall length of the chosen processing times does not violate the bound. The configuration IP is then given by variables x_C for each configuration C; constraints ensuring that there is a machine for each configuration, i.e., $\sum_C x_C = m$; and further constraints due to which the jobs are covered, i.e., $\sum_C C_p x_C = |\{j \in \mathcal{J} \mid p_j = p\}|$ for each processing time p. In combination with certain simplification techniques, this type of IP is often used in the design of *polynomial time approximation schemes* (PTAS). A PTAS is a procedure that, for any fixed accuracy parameter $\varepsilon > 0$, returns a solution with approximation guarantee $(1+\varepsilon)$, that is, a solution whose objective value lies within a factor of $(1+\varepsilon)$ of the optimum. In the context of machine scheduling, the aforementioned simplification techniques can be used to guess the target makespan T of the given instance; to upper bound the cardinality of the set of processing times P by a constant (depending in $1/\varepsilon$); and to lower bound the processing times in size such that they are within a constant factor of the makespan T (see, e.g., [4,19]). Hence, only a constant number of configurations is needed, which leads to an integer program with a constant number of variables. Integer programs of that kind can be efficiently solved using the classical algorithm by Lenstra and Kannan [22,27], yielding a PTAS for machine scheduling. Here, the error of $(1+\varepsilon)$ in the quality of the solution is due to the simplification steps, and the scheme has a running time of the form $f(1/\varepsilon)\cdot\mathrm{poly}(|I|)$, where $|I|$ denotes the input size and f some computable function. A PTAS with this property is called *efficient* (EPTAS). Note that for a regular PTAS a running time of the form $|I|^{f(1/\epsilon)}$ is allowed. It is well-known that machine scheduling is strongly NP-hard, and therefore it admits no optimal polynomial time algorithm, unless P=NP. Moreover, a so-called *fully polynomial* PTAS (FPTAS)—which is an EPTAS with a polynomial function f—cannot be hoped for either.

Machine Scheduling with Classes The configuration IP is used in a wide variety of approximation schemes for machine scheduling problems [4,19]. However, for scheduling problems where the jobs have to meet some additional requirements, such as class dependencies, the approach often ceases to work. A problem emerging, in this case, is that the additional requirements have to be represented in the configurations, resulting in a super-constant number of variables in the IP. We elaborate on this using a concrete example: Consider the variant of machine scheduling in which the jobs are partitioned into K setup classes. For each job j, a class k_j is given; and for each class k, a setup time s_k has to be paid on a machine if a job belonging to that class is scheduled on it, i.e., $C_{\max}(\sigma) = \max_{i\in[m]}\big(\sum_{j\in\sigma^{-1}(i)} p_j + \sum_{k\in\{k_j \mid j\in\sigma-1(i)\}} s_k\big)$. With some effort, simplification steps similar to the ones for machine scheduling can be applied. In the course of this, the setup times as well can be suitably bounded in number and guaranteed to be sufficiently big (see [20]). However, it is not obvious how the configuration IP should be extended without losing the property that it can be solved efficiently. For instance, extending the configurations with multiplicities of setup times creates a need to encode class information into the configurations or to introduce other class dependent variables. This leads to a super-constant number of variables and constraints.

Module Configuration IP Our approach to deal with the class dependencies of the jobs is to cover the job classes with so-called modules and cover the modules in turn with configurations in an augmented IP, called the module configuration IP (MCIP). In the setup class model, for instance, the modules may be defined as combinations of setup times and multiplicity vectors of processing times, and the configurations, in turn, as multiplicity vectors of module sizes. The number of both the modules and the configurations will typically be bounded by a constant. To cover the classes by modules, each class is provided with its own set of modules, that is, there are variables for each pair of class and module. Since the number of classes is part of the input, the number of variables in the resulting MCIP is super-constant, and therefore the algorithm by Lenstra and Kannan [22,27] is not the proper tool for the solving of the MCIP. However, the MCIP has a certain simple structure: The mentioned variables are partitioned into uniform classes each corresponding to the set of modules, and for each class, the modules have to do essentially the same, that is, cover the jobs of the class. Utilizing these properties, we can formulate the MCIP in the framework of n-fold integer programs—a class of IPs whose variables and constraints fulfill certain uniformity requirements. In 2013 Hemmecke et al. [15] presented the first

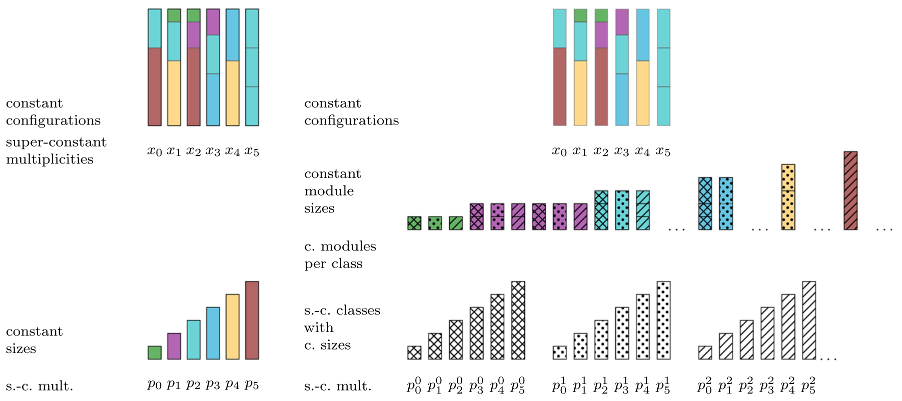
constant configurations
super-constant multiplicities
x_0 x_1 x_2 x_3 x_4 x_5
constant sizes
s.-c. mult.
p_0 p_1 p_2 p_3 p_4 p_5
constant configurations
x_0 x_1 x_2 x_3 x_4 x_5
constant module sizes
c. modules per class
s.-c. classes with c. sizes
s.-c. mult.
p_0^0 p_1^0 p_2^0 p_3^0 p_4^0 p_5^0
p_0^1 p_1^1 p_2^1 p_3^1 p_4^1 p_5^1
p_0^2 p_1^2 p_2^2 p_3^2 p_4^2 p_5^2

Fig. 1 On the left, there is a schematic representation of the configuration IP. There is a constant number of different sizes each occurring a super-constant number of times. The sizes are directly mapped to configurations. On the right, there is a schematic representation of the MCIP. There is a super-constant number of classes each containing a constant number of sizes which have super-constant multiplicities. The elements from the class are mapped to a constant number of different modules, which have a constant number of sizes. These module sizes are mapped to configurations

fixed parameter tractable (FPT) algorithm for n-fold IPs, that is, an algorithm with a running time $f(k)\cdot \mathrm{poly}(|I|)$ where k is some parameter (or a sequence of parameters) depending in the instance. In the MCIP the corresponding parameters can be properly bounded which enables the present result. For a more detailed description of n-fold IPs and the MCIP, the reader is referred to Sects. 2 and 3, respectively. In Fig. 1, the basic idea of the MCIP is visualized.

Using the MCIP, we are able to formulate an EPTAS for machine scheduling in the setup class model described above. Before, only a regular PTAS with running time $nm^{\mathcal{O}(1/\varepsilon^5)}$ was known [20]. To the best of our knowledge, this is the first use of n-fold integer programing in the context of approximation algorithms.

Results and Methodology To show the conceptual power of the MCIP, we utilize it for two more problems: The *splittable* and the *preemptive* setup model of machine scheduling. In both variants, for each job j, a setup time s_j is given. Each job may be partitioned into multiple parts that can be assigned to different machines, but before any part of the job can be processed the setup time has to be paid. In the splittable model, job parts belonging to the same job can be processed in parallel, and therefore it suffices to find a partition of the jobs and an assignment of the job parts to machines. This is not the case for the preemptive model, in which additionally a starting time for each job part has to be found, and two parts of the same job may not be processed in parallel. In 1999, Schuurman and Woeginger [33] presented a polynomial time algorithm for the preemptive model with approximation guarantee $4/3+\varepsilon$, and for the splittable case, a guarantee of $5/3$ was achieved by Chen et al. [6]. These are the best known approximation guarantees for the problems at hand. We show that solutions arbitrarily close to the optimum can be found in polynomial time:

Theorem 1 *There is an efficient PTAS with running time* $2^{f(1/\varepsilon)}\mathrm{poly}(|I|)$ *for minimum makespan scheduling on identical parallel machines in the setup-class model, as well as in the preemptive and splittable setup models.*

More precisely, we get a running time of $2^{\mathcal{O}(1/\varepsilon^3(\log 1/\varepsilon)^4)}K^{1+o(1)}nm$ in the setup class model, $2^{\mathcal{O}(1/\varepsilon^2(\log 1/\varepsilon)^3)}n^{2+o(1)}$ in the splittable, and $2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}}n^{1+o(1)}m$ in the preemptive model. Note that all three problems are strongly NP-hard, due to trivial reductions from machine scheduling, and hence FPTAS results cannot be hoped for.

Summing up, the main achievement of this work is the development of the module configuration IP and its application in the design of approximation schemes. Up to now, EPTAS or even PTAS results seemed out of reach for the considered problems, and for the preemptive model, we provide the first improvement in 20 years. The simplification techniques developed for the splittable and preemptive model in order to employ the MCIP are original and in the latter case quite sophisticated and therefore interesting by themselves. Furthermore, we expect the MCIP to be applicable to other packing and scheduling problems as well, in particular for variants of machine scheduling and bin packing with additional class dependent constraints. On a more conceptual level, we have presented a first demonstration of the potential of n-fold integer programming in the theory of approximation algorithms and hope to inspire further studies in this direction.

We conclude this paragraph with a more detailed overview of our results and their presentation. For all three EPTAS results, we employ the classical dual approximation framework by Hochbaum and Shmoys [16] to get a guess of the makespan T. This approach is introduced in Sect. 2 together with n-fold IPs and formal definitions of the problems. In the following section, we develop the module configuration IP and argue that it is indeed an n-fold IP. The EPTAS results follow the same basic approach described above for machine scheduling: We find a schedule for a simplified instance via the MCIP and transform it into a schedule for the original one. The simplification steps typically include rounding of the processing and setup times using standard techniques, as well as the removal of certain jobs which later can be reinserted via carefully selected greedy procedures. For the splittable and preemptive model, we additionally have to prove that schedules with a certain simple structure exist, and in the preemptive model, the MCIP has to be extended. In Sect. 4 the basic versions of the EPTAS are presented, and in Sect. 5 some improvements of the running time for the splittable and the setup class model are discussed.

Related work For an overview on n-fold IPs and their applications, we refer to the following works [12,28,31]. The first FPT algorithm for n-fold IPs was presented by Hemmecke et al. [15] in 2013, and it has a running time with a cubic dependence in n. In 2018, Eisenbrand et al. [11] and independently Koutecký et al. [26] developed algorithms with running times with near quadratic dependence in n and improved dependencies in the parameters. Then, in 2019, a near linear dependence in n was achieved by Jansen et al. [21] as well as Eisenbrand et al. [12]. Finally, in 2021, Cslovjecsek et al. [10] further improved and parallelized the result. For an overview on recent results on n-fold IPs and related topics we refer to [12].

There have been recent applications of n-fold integer programming to scheduling problems in the context of parameterized algorithms: Knop and Koutecký [23] showed, among other things, that the problem of makespan minimization on unrelated parallel machines where the processing times are dependent on both jobs and machines is fixed-parameter tractable with respect to the maximum processing time and the number of distinct machine types. This was generalized to the parameters maximum processing time and rank of the processing time matrix by Chen et al. [7]. Furthermore, Knop et al. [25] provided an improved algorithm for a special type of n-fold IPs, yielding improved running times for several applications of n-fold IPs including results for scheduling problems. In a recent result [24], published after the present work, the configuration IP is strongly generalized. The resulting problem is modeled as an n-fold IP and shown to catch several allocation problems.

There is extensive literature concerning scheduling problems with setup times. We highlight a few closely related results and otherwise refer to the surveys [1–3]. In the following, we use the term α-approximation as an abbreviation for polynomial time algorithms with approximation guarantee α. The setup class model was first considered by Mäcker et al. [29] in the special case that all classes have the same setup time. They designed a 2-approximation and additionally a $(3/2 + \varepsilon)$-approximation for the case that the overall length of the jobs from each class is bounded. Jansen and Land [20] presented a simple 3-approximation with linear running time, a $(2 + \varepsilon)$-approximation, and the aforementioned PTAS for the general setup class model. As indicated before, Chen et al. [6] developed a $5/3$-approximation for the splittable model. A generalization of this, in which both setup and processing times are job and machine dependent, has been considered by Correa et al. [8]. They achieve a $(1+\phi)$-approximation where ϕ denotes the golden ratio, using a newly designed linear programming formulation. Moreover, there are recent results concerning machine scheduling in the splittable model considering the sum of (weighted) completion times as the objective function, e.g., [9,32]. For the preemptive model, a PTAS for the special case that all jobs have the same setup time has been developed by Schuurman and Woeginger [33]. The mentioned $(4/3 + \varepsilon)$-approximation for the general case [33] follows the same approach. Furthermore, a combination of the setup class and the preemptive model has been considered in which the jobs are scheduled preemptively, but the setup times are class dependent. Monma and Potts [30] presented, among other things, a $(2 - 1/(\lfloor m/2 \rfloor + 1))$-approximation for this model, and later Chen [5] achieved improvements for some special cases.

2 Preliminaries

In the following, we establish some concepts and notations, formally define the considered problems, and outline the dual approximation approach by Hochbaum and Shmoys [16], as well as n-fold integer programs.

For any integer n, we denote the set $\{1, \dots, n\}$ by $[n]$; we write $\log(\cdot)$ for the logarithm with basis 2; and we will usually assume that some instance I of the problem considered in the respective context is given together with an accuracy parameter $\varepsilon \in (0, 1)$ such that $1/\varepsilon$ is an integer. Furthermore, for any two sets X, Y, we write

Y^X for the set of functions $f : X \to Y$. If X is finite, we say that Y is indexed by X and sometimes denote the function value of f for the argument $x \in X$ by f_x.

Problems For all three of the considered problems, a set $\mathcal{J}$ of n jobs with processing times $p_j \in \mathbb{Q}_{>0}$ for each job $j \in \mathcal{J}$ and a number of machines m is given. In the preemptive and the splittable model, the input additionally includes a setup time $s_j \in \mathbb{Q}_{>0}$ for each job $j \in \mathcal{J}$; while in the setup class model, it includes a number K of setup classes, a setup class $k_j \in [K]$ for each job $j \in \mathcal{J}$, as well as setup times $s_k \in \mathbb{Q}_{>0}$ for each $k \in [K]$.

We take a closer look at the definition of a schedule in the preemptive model. The jobs may be split. Therefore, partition sizes $\kappa : \mathcal{J} \to \mathbb{Z}_{>0}$, together with processing time fractions $\lambda_j : [\kappa(j)] \to (0, 1]$ such that $\sum_{k\in[\kappa(j)]} \lambda_j(k) = 1$ have to be found, meaning that job j is split into $\kappa(j)$ many parts and the k-th part for $k \in [\kappa(j)]$ has processing time $\lambda_j(k)p_j$. This given, we define $\mathcal{J}' = \{(j,k) \mid j \in \mathcal{J}, k \in [\kappa(j)]\}$ to be the set of job parts. Now, an assignment $\sigma : \mathcal{J}' \to [m]$ along with starting times $\xi : \mathcal{J}' \to \mathbb{Q}_{>0}$ has to be determined such that any two job parts assigned to the same machine or belonging to the same job do not overlap. More precisely, we have to assure that for each two job parts $(j,k), (j',k') \in \mathcal{J}'$ with $\sigma(j,k) = \sigma(j',k')$ or $j = j'$, we have $\xi(j,k) + s_j + \lambda_j(k)p_j \le \xi(j',k')$ or vice versa. A schedule is given by $(\kappa, \lambda, \sigma, \xi)$, and the makespan can be defined as $C_{\max} = \max_{(j,k)\in\mathcal{J}'}(\xi(j,k) + s_j + \lambda_j(k)p_j)$. Note that the variant of the problem in which overlap between a job part and setup of the same job is allowed is equivalent to the one presented above. This was pointed out by Schuurmann and Woeginger [33] and can be seen with a simple swapping argument.

In the splittable model, it is not necessary to determine starting times for the job parts because, given the assignment σ, the job parts assigned to each machine can be scheduled as soon as possible in arbitrary order without gaps. Hence, in this case, the output is of the form $(\kappa, \lambda, \sigma)$, and the makespan can be defined as $C_{\max} = \max_{i\in[m]} \sum_{(j,k)\in\sigma^{-1}(i)}(s_j + \lambda_j(k)p_j)$.

Lastly, in the setup class model, the jobs are not split, and the jobs assigned to each machine can be scheduled in batches comprised of the jobs of the same class assigned to the machine without overlaps and gaps. The output is therefore just an assignment $\sigma : \mathcal{J} \to [m]$, and the makespan is given by $C_{\max} = \max_{i\in[m]} \sum_{j\in\sigma^{-1}(i)} p_j + \sum_{k\in\{k_j \mid j\in\sigma^{-1}(i)\}} s_k$.

Note that in the preemptive and the setup class model, we can assume that the number of machines is bounded by the number of jobs: If there are more machines than jobs, placing each job on a private machine yields an optimal schedule in both models, and the remaining machines can be ignored. This, however, is not the case in the splittable model, which causes a minor problem in the following.

Dual Approximation All of the presented algorithms follow the dual approximation framework introduced by Hochbaum and Shmoys [16]: Instead of solving the minimization version of a problem directly, it suffices to find a procedure that for a given bound T on the objective value either correctly reports that there is no solution with value T, or returns a solution with value at most $(1 + a\varepsilon)T$ for some constant a. If we have some initial upper bound B for the optimal makespan OPT with $B \le b$OPT for some b, we can define a PTAS by trying different values T from the interval $[B/b, B]$ in a binary search fashion, and find a value $T^* \leq (1+\mathcal{O}(\varepsilon))\text{OPT}$ after $\mathcal{O}(\log b/\varepsilon)$ iterations. Note that for all of the considered problems, constant approximation algorithms are known, and the sum of all processing and setup times is a trivial m-approximation. Hence, we always assume that a target makespan T is given. Furthermore, we assume that the setup times and in the preemptive and setup class cases also the processing times are bounded by T because otherwise we can reject T immediately.

n-fold Integer Programs We briefly define n-fold integer programs (IPs) following the notation of [15] and [23], and state the main algorithmic result needed in the following. Let $n, r, s, t \in \mathbb{Z}_{>0}$ be integers and A be an integer $((r + ns) \cdot nt)$-matrix of the following form:

$$A = \begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}$$

The matrix A is the so-called n-fold product of the bimatrix $\binom{A_1}{A_2}$, with A_1 an $r \times t$ and A_2 an $s \times t$ matrix. Furthermore, let $w, \ell, u \in \mathbb{Z}^{nt}$ and $b \in \mathbb{Z}^{r+ns}$. Then the n-fold integer programming problem is given by:

$$\min\{wx \mid Ax = b, \ell \leq x \leq u, x \in \mathbb{Z}^{nt}\}$$

We set Δ to be the maximum absolute value occurring in A. There are several algorithms for solving n-fold IPs. We use the most recent result by Cslovjecsek et al. [21]:

Theorem 2 *The n-fold integer programming problem can be solved in time* $2^{\mathcal{O}(rs^2)} (rs\Delta)^{\mathcal{O}(r^2s+s^2)}(nt)^{1+o(1)}$.

The variables x can naturally be partitioned into *bricks* $x^{(q)}$ of dimension t for each $q \in [n]$ such that $x = (x^{(1)}, \dots x^{(n)})$. Furthermore, we denote the constraints corresponding to A_1 as *globally uniform* and the ones corresponding to A_2 as *locally uniform*. Hence, r is the number of globally and s the number of locally uniform constraints (ignoring their n-fold duplication), t the *brick size* and n the *brick number*.

3 Module configuration IP

In this section, we state the configuration IP for machine scheduling; introduce a basic version of the module configuration IP (MCIP) that is already sufficiently general to work for both the splittable and setup class model; and lastly show that the configuration IP can be expressed by the MCIP in multiple ways. Before that, however, we formally introduce the concept of *configurations*.

Given a set of objects A, such as jobs, a configuration C of these objects is a vector of multiplicities indexed by the objects, i.e., $C \in \mathbb{Z}_{\geq 0}^A$. For given sizes $\Lambda(a)$ of the objects $a \in A$, the size $\Lambda(C)$ of a configuration C is defined as $\sum_{a \in A} C_a \Lambda(a)$. Moreover, for a given upper bound B, we define $\mathcal{C}_A(B)$ to be the set of configurations of A that are bounded in size by B, that is, $\mathcal{C}_A(B) = \{C \in \mathbb{Z}_{\geq 0}^A \mid \Lambda(C) \leq B\}$.

Configuration IP We provide a recollection of the configuration IP for machine scheduling. Let P be the set of distinct processing times for some instance I with multiplicities n_p for each $p \in P$, meaning, I includes exactly n_p jobs with processing time p. The size $\Lambda(p)$ of a processing time p is the processing time itself, that is, $\Lambda(p) = p$. Furthermore, let T be a guess of the optimal makespan. The configuration IP for I and T is given by variables $x_C \geq 0$ for each $C \in \mathcal{C}_P(T)$ and the following constraints:

$$\sum_{C \in \mathcal{C}_P(T)} x_C = m \tag{1}$$

$$\sum_{C \in \mathcal{C}_P(T)} C_p x_C = n_p \qquad \forall p \in P \tag{2}$$

Due to constraint (1), exactly one configuration is chosen for each machine; while (2) ensures that the correct number of jobs or job sizes is covered.

Module Configuration IP Let $\mathcal{B}$ be a set of basic objects (e.g., jobs or setup classes) and let there be D integer values $B_1, \ldots, B_D$ for each basic object $B \in \mathcal{B}$ (e.g., processing time or numbers of different kinds of jobs). Our approach is to cover the basic objects with so-called *modules* and, in turn, cover the modules with configurations. Depending on the context, modules correspond to batches of jobs or job piece sizes together with a setup time and can also encompass additional information like a starting time. Let $\mathcal{M}$ be a set of such modules. In order to cover the basic objects, each module $M \in \mathcal{M}$ also has D integer values $M_1, \ldots, M_D$. Furthermore, each module M has a size $\Lambda(M)$ and a set of eligible basic objects $\mathcal{B}(M)$. The latter is needed because not all modules are compatible with all basic objects, e.g., because they do not have the right setup times. The configurations are used to cover the modules, however, it typically does not matter which module exactly is covered, but rather which size the module has. Let H be the set of distinct module sizes, i.e., $H = \{\Lambda(M) \mid M \in \mathcal{M}\}$, and for each module size $h \in H$ let $\mathcal{M}(h)$ be the set of modules with size h. We consider the set $\mathcal{C}$ of configurations of module sizes which are bounded in size by a guess of the makespan T, i.e., $\mathcal{C} = \mathcal{C}_H(T)$. In the preemptive case, configurations need to additionally encompass information about starting times of modules, and therefore the definition of configurations will be slightly more complicated in that case.

Since we want to choose configurations for each machine, we have variables x_C for each $C \in \mathcal{C}$ and constraints corresponding to (1). Furthermore, we choose modules with variables y_M for each $M \in \mathcal{M}$, and because we want to cover the chosen modules with configurations, we have some analogue of constraint (2), say $\sum_{C \in \mathcal{C}(T)} C_h x_C = \sum_{M \in \mathcal{M}(h)} y_M$ for each module size $h \in H$. However, while a module M may be used to cover multiple basic objects, each instance of M should only be used for one of them. Hence, it makes sense to introduce the variables y_M for each basic object, and this is were n-fold IPs come into play. The variables stated so far form a brick of the variables of the n-fold IP, and there is one brick for each basic object, that is, we have, for each $B \in \mathcal{B}$, variables $x_C^{(B)}$ for each $C \in \mathcal{C}$, and $y_M^{(B)}$ for each $M \in \mathcal{M}$. Using the upper bounds of the n-fold model, variables $y_M^{(B)}$ are set to zero, if B is not eligible for M; and we set the lower bounds of all variables to zero. Sensible upper bounds for the remaining variables will be typically clear from context. Besides that, the module configuration integer program MCIP (for $\mathcal{B}$, $\mathcal{M}$ and $\mathcal{C}$) is given by:

$$\sum_{B\in\mathcal{B}}\sum_{C\in\mathcal{C}} x_C^{(B)} = m \tag{3}$$

$$\sum_{B\in\mathcal{B}}\sum_{C\in\mathcal{C}(T)} C_h x_C^{(B)} = \sum_{B\in\mathcal{B}}\sum_{M\in\mathcal{M}(h)} y_M^{(B)} \qquad \forall h \in H \tag{4}$$

$$\sum_{M\in\mathcal{M}} M_d y_M^{(B)} = B_d \qquad \forall B \in \mathcal{B}, d \in [D] \tag{5}$$

It is easy to see that the constraints (3) and (4) are globally uniform. They are the mentioned adaptations of (1) and (2). The constraint (5), on the other hand, is locally uniform and ensures that the basic objects are covered.

Note that, while the duplication of the configuration variables does not carry meaning, it also does not upset the model: Consider the modified MCIP that is given by not duplicating the configuration variables. A solution $(\tilde{x}, \tilde{y})$ for this IP gives a solution (x, y) for the MCIP by fixing some basic object B^*, setting $x_C^{(B^*)} = \tilde{x}_C$ for each configuration C, setting the remaining configuration variables to 0, and copying the remaining variables. Given a solution (x, y) for the MCIP, on the other hand, gives a solution for the modified version $(\tilde{x}, \tilde{y})$ by setting $\tilde{x}_C = \sum_{B\in\mathcal{B}} x_C^B$ for each configuration C. Summarizing we get:

Observation 1 The MCIP is an n-fold IP with brick-size $t = |\mathcal{M}| + |\mathcal{C}|$, brick number $n = |\mathcal{B}|$, $r = |H| + 1$ globally uniform and $s = D$ locally uniform constraints.

Moreover, in all of the considered applications, we will minimize the overall size of the configurations, i.e., $\sum_{B\in\mathcal{B}}\sum_{C\in\mathcal{C}} \Lambda(C) x_C^{(B)}$. This will be required because in the simplification steps of our algorithms some jobs are removed and have to be reinserted later, and we therefore have to make sure that no space is wasted.

First Example We conclude the section by pointing out several different ways to replace the classical configuration IP for machine scheduling with the MCIP, thereby giving some intuition for the model. The first possibility is to consider the jobs as the basic objects and their processing times as their single value ($\mathcal{B} = \mathcal{J}$, $D = 1$); the modules are the processing times ($\mathcal{M} = P$), and a job is eligible for a module, if its processing time matches; and the configurations are all the configurations bounded in size by T. Another option is to choose the processing times as basic objects keeping all the other definitions essentially like before. Lastly, we could consider the whole set of jobs or the whole set of processing times as a single basic object with $D = |P|$ different values. In this case, we can define the set of modules as the set of configurations of processing times bounded by T.

4 EPTAS results

In this section, we present approximation schemes for each of the three considered problems. Each of the results follows the same approach: The instance is carefully simplified, a schedule for the simplified instance is found using the MCIP, and this schedule is transformed into a schedule for the original instance. The presentation of the result is also similar for each problem: We first discuss how the instance can be sensibly simplified and how a schedule for the simplified instance can be transformed into a schedule for the original one. Next, we discuss how a schedule for the simplified instance can be found using the MCIP, and, lastly, we summarize and analyze the taken steps.

For the sake of clarity, we have given rather formal definitions for the problems at hand in Sect. 2. In the following, however, we will use the terms in a more intuitive fashion for the most part, and we will, for instance, often take a geometric rather than a temporal view on schedules and talk about the *length* or the *space* taken up by jobs and setups on machines rather than time. In particular, given a schedule for an instance of any one of the three problems together with an upper bound for the makespan T, the *free space* with respect to T on a machine is defined as the summed up lengths of time intervals between 0 and T in which the machine is idle. The free space (with respect to T) is the summed up free space of all the machines. For bounds T and L for the makespan and the free space, respectively, we say that a schedule is a (T, L)-schedule if its makespan is at most T and the free space with respect to T is at least L.

When transforming the instance, we will increase or decrease processing and setup times and fill in or remove extra jobs. Consider a (T', L')-schedule where T' and L' denote some arbitrary makespan or free space bounds. If we fill in extra jobs or increase processing or setup times, but can bound the increase on each machine by some bound b, we end up with a $(T' + b, L')$-schedule for the transformed instance. In particular, we have the same bound for the free space because we properly increased the makespan bound. If, on the other hand, jobs are removed or setup times decreased, we obviously still have a (T', L')-schedule for the transformed instance. This will be used frequently in the following.

4.1 Setup class model

We start with the setup class model. In this case, we can essentially reuse the simplification steps that were developed by Jansen and Land [20] for their PTAS. The main difference between the two procedures is that we solve the simplified instance via the MCIP, while they used a dynamic program. For the sake of self-containment, we include our own simplification steps, but remark that they are strongly inspired by

Table 1 Overview on the job classifications

s_{k_j}	p_j		
	$< \varepsilon^4 T$	$< \varepsilon T$	$\geq \varepsilon T$
$< \varepsilon^3 T$	$\mathcal{J}^{\text{sst}}_{\text{tiny}}$	$\mathcal{J}^{\text{sst}}_{\text{small}}$	$\mathcal{J}^{\text{sst}}_{\text{large}}$
$\geq \varepsilon^3 T$	$\mathcal{J}^{\text{bst}}_{\text{tiny}}$	$\mathcal{J}^{\text{bst}}_{\text{small}}$	$\mathcal{J}^{\text{bst}}_{\text{large}}$

those from [20]. In Sect. 5, we present a more elaborate rounding procedure resulting in an improved running time.

Simplification of the Instance In the following, we distinguish *big setup* jobs j belonging to classes k with setup times $s_k \geq \varepsilon^3 T$ and *small setup* jobs with $s_k < \varepsilon^3 T$. We denote the corresponding subsets of jobs by $\mathcal{J}^{\text{bst}}$ and $\mathcal{J}^{\text{sst}}$, respectively. Furthermore, we call a job *tiny* or *small*, if its processing time is smaller than $\varepsilon^4 T$ or εT, respectively, and *big* or *large* otherwise. For any given set of jobs J, we denote the subset of tiny jobs from J with J_{tiny} and the small, big, and large jobs analogously (see Table 1 for an overview).

We simplify the instance in four steps, aiming for an instance that exclusively includes big jobs with big setup times and additionally only a constant number of distinct processing and setup times. For technical reasons, we assume $\varepsilon \leq 1/2$.

We proceed with the first simplification step. Let I_1 be the instance given by the job set $\mathcal{J} \setminus \mathcal{J}^{\text{sst}}_{\text{small}}$ and Q the set of setup classes completely contained in $\mathcal{J}^{\text{sst}}_{\text{small}}$, i.e., $Q = \{k \mid \forall j \in \mathcal{J} : k_j = k \Rightarrow j \in \mathcal{J}^{\text{sst}}_{\text{small}}\}$. An obvious lower bound on the space taken up by the jobs from $\mathcal{J}^{\text{sst}}_{\text{small}}$ in any schedule is given by $L = \sum_{j \in \mathcal{J}^{\text{sst}}_{\text{small}}} p_j + \sum_{k \in Q} s_k$. Note that the instance I_1 may include a reduced number K' of setup classes.

Lemma 1 *A schedule for I with makespan T induces a (T, L)-schedule for I_1, that is, a schedule with makespan T and free space at least L; and any (T', L)-schedule for I_1 can be transformed into a schedule for I with makespan at most $(1+\varepsilon)T' + \varepsilon T + 2\varepsilon^3 T$.*

Proof The first claim is obvious and we therefore assume that we have a (T', L)-schedule for I_1. We group the jobs from $\mathcal{J}^{\text{sst}}_{\text{small}}$ by setup classes and first consider the groups with summed up processing time at most $\varepsilon^2 T$. For each of these groups, we check whether the respective setup class contains a large job. If this is the case, we schedule the complete group on a machine on which such a large job is already scheduled if possible using up free space. Since the large jobs have a length of at least εT, there are at most $T'/(\varepsilon T)$ many large jobs on each machine, and therefore the schedule on the respective machine has length at most $(1+\varepsilon)T'$, or there is free space with respect to T' left. If, on the other hand, the respective class does not contain a large job and is therefore fully contained in $\mathcal{J}^{\text{sst}}_{\text{small}}$, we create a container including the whole class and its setup time. Note that the overall length of the container is at most $(\varepsilon^2 + \varepsilon^3)T \leq \varepsilon T$ (using $\varepsilon \leq 1/2$). Next, we create a sequence containing the containers and the remaining jobs ordered by setup class. We insert the items from this sequence greedily into the remaining free space in a next-fit fashion exceeding T' on each machine by at most one item from the sequence, thereby creating an error of at most εT. This can be done because we had a free space of at least L, and the inserted objects had an overall length of at most L. To make the resulting schedule feasible, we have to insert some setup times. However, because the overall length of the jobs from each class in need of a setup is at least $\varepsilon^2 T$, and the sequence was ordered by classes, there are at most $T'/(\varepsilon^2 T) + 2$ distinct classes without a setup time on each machine. Inserting the missing setup times will therefore increase the makespan by at most $(T'/(\varepsilon^2 T) + 2)\varepsilon^3 T = \varepsilon T' + 2\varepsilon^3 T$. □

Next, we deal with the remaining (large) jobs with small setup times $j \in \mathcal{J}^{\text{sst}}_{\text{large}}$. Let I_2 be the instance we get by increasing the setup times of the classes with small setup times to $\varepsilon^3 T$. We denote the setup time of class $k \in [K']$ for I_2 by s'_k. Note that there are no small setup jobs in I_2.

Lemma 2 *A (T', L')-schedule I_1 induces a $((1 + \varepsilon^2)T', L')$-schedule for I_2, and a (T', L')-schedule for I_2 is also a (T', L')-schedule for I_1.*

Proof The first claim is true because in a schedule with makespan at most T' there can be at most $T'/(\varepsilon T)$ many large jobs on any machine, and the second claim is obvious. □

Let I_3 be the instance we get by replacing the jobs from $\mathcal{J}^{\text{bst}}_{\text{tiny}}$ with placeholders of size $\varepsilon^4 T$. More precisely, we remove $\mathcal{J}^{\text{bst}}_{\text{tiny}}$, and, for each class $k \in [K]$, we introduce $\lceil (\sum_{j \in \mathcal{J}^{\text{bst}}_{\text{tiny}}, k_j = k} p_j)/(\varepsilon^4 T) \rceil$ many jobs with processing time $\varepsilon^4 T$ and class k. We denote the job set of I_3 by $\mathcal{J}'$ and the processing time of a job $j \in \mathcal{J}'$ by p'_j. Note that I_3 exclusively contains big jobs with big setup times.

Lemma 3 *If there is a (T', L')-schedule for I_2, there is also a $((1+\varepsilon)T', L')$-schedule; and if there is a (T', L')-schedule for I_3, there is also a $((1 + \varepsilon)T', L')$-schedule for I_2.*

Proof Note that for any (T', L')-schedule for I_2 or I_3, there are at most $T'/(\varepsilon^3 T)$ many distinct big setup classes scheduled on any machine. Hence, when considering such a schedule for I_2, we can remove the tiny jobs belonging to $\mathcal{J}^{\text{bst}}_{\text{tiny}}$ from the machines and instead fill in the placeholders, such that each machine for each class receives at most as much length from that class, as was removed, rounded up to the next multiple of $\varepsilon^4 T$. All placeholders can be placed like this and the makespan is increased by at most $(T'/(\varepsilon^3 T))\varepsilon^4 T = \varepsilon T'$. If, on the other hand, we consider such a schedule for I_3, we can remove the placeholders and instead fill in the respective tiny jobs, again overfilling by at most one job. This yields a $((1 + \varepsilon)T', L')$-schedule for I_2 with the same argument. □

Lastly, we perform both a geometric and an arithmetic rounding step for the processing and setup times. The geometric rounding is needed to suitably bound the number of distinct processing and setup times, and due to the arithmetic rounding, we will be able to guarantee integral coefficients in the IP. More precisely, we set $\tilde{p}_j = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} p'_j/(\varepsilon^4 T) \rceil} \varepsilon^4 T$ and $\bar{p}_j = \lceil \tilde{p}_j/\varepsilon^5 T \rceil \varepsilon^5 T$ for each $j \in \mathcal{J}'$, as well as $\tilde{s}_j = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} s'_j/(\varepsilon^3 T) \rceil} \varepsilon^3 T$ and $\bar{s}_k = \lceil \tilde{s}_j/\varepsilon^5 T \rceil \varepsilon^5 T$ for each setup class $k \in [K']$. The resulting instance is called I_4.

Lemma 4 *A (T', L')-schedule for I_3 induces a $((1+3\varepsilon)T', L')$-schedule for I_4, and any (T', L')-schedule for I_4 can be turned into a (T', L')-schedule for I_3.*

Proof For the first claim, we first stretch a given schedule by $(1+\varepsilon)$. This enables us to use the processing and setup times due to the geometric rounding step. Now, using the ones due to the second step increases the schedule by at most $2\varepsilon T'$, because there where at most $T'/(\varepsilon^4 T)$ many big jobs on any machine to begin with. The second claim is obvious. □

Based on the rounding steps, we define two makespan bounds $\bar{T}$ and $\breve{T}$: Let $\bar{T}$ be the makespan bound that is obtained from T by the application of the Lemmata 1–4 in sequence, i.e., $\bar{T} = (1+\varepsilon^2)(1+\varepsilon)(1+3\varepsilon)T = (1+\mathcal{O}(\varepsilon))T$. We will find a $(\bar{T}, L)$-schedule for I_4 utilizing the MCIP and afterward apply the Lemmata 1–4 backwards to get a schedule with makespan $\breve{T} = (1+\varepsilon)^2\bar{T} + \varepsilon T + 2\varepsilon^3 T = (1+\mathcal{O}(\varepsilon))T$.

Let P and S be the sets of distinct occurring processing and setup times for instance I_4. Because of the rounding, the minimum and maximum lengths of the setup and processing times, and $\varepsilon < 1$, we can bound $|P|$ and $|S|$ by $\mathcal{O}(\log_{1+\varepsilon} 1/\varepsilon) = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$.

Utilization of the MCIP At this point, we can employ the module configuration IP. The basic objects in this context are the setup classes, i.e., $\mathcal{B} = [K']$, and the different values are the numbers of jobs with a certain processing time, i.e., $D = |P|$. We set $n_{k,p}$ to be the number of jobs from setup class $k \in [K']$ with processing time $p \in P$. The modules correspond to batches of jobs together with a setup time. Batches of jobs can be modeled as configurations of processing times, that is, multiplicity vectors indexed by the processing times. Hence, we define the set of modules $\mathcal{M}$ to be the set of pairs of configurations of processing times and setup times with a summed up size bounded by $\bar{T}$, i.e., $\mathcal{M} = \{(C, s) \mid C \in \mathcal{C}_P(\bar{T}), s \in S, s + \Lambda(C) \le \bar{T}\}$, and write $M_p = C_p$ and $s_M = s$ for each module $M = (C, s) \in \mathcal{M}$. The values of a module M are given by the numbers M_p and its size $\Lambda(M)$ by $s_M + \sum_{p\in P} M_p p$. Remember that the configurations $\mathcal{C}$ are the configurations of module sizes H that are bounded in size by $\bar{T}$, i.e., $\mathcal{C} = \mathcal{C}_H(\bar{T})$. A setup class is eligible for a module if the setup times fit, i.e., $\mathcal{B}_M = \{k \in [K'] \mid s_k = s_M\}$. Lastly, we establish $\varepsilon^5 T = 1$ by scaling.

For the sake of readability, we state the resulting constraints of the MCIP with adapted notation and without duplication of the configuration variables:

$$\sum_{C\in\mathcal{C}} x_C = m \tag{6}$$

$$\sum_{C\in\mathcal{C}} C_h x_C = \sum_{k\in[K']} \sum_{M\in\mathcal{M}(h)} y_M^{(k)} \qquad \forall h \in H \tag{7}$$

$$\sum_{M\in\mathcal{M}} M_p y_M^{(k)} = n_{k,p} \qquad \forall k \in [K'], p \in P \tag{8}$$

Note that the coefficients are all integral and this includes those of the objective function, i.e., $\sum_C \Lambda(C)x_C$, because of the scaling step.

Lemma 5 *With the above definitions, there is a* $(\bar{T}, L)$*-schedule for* I_4 *if and only if the* MCIP *has a solution with objective value at most* $m\bar{T} - L$*.*

Proof Let there be a $(\bar{T}, L)$-schedule for I_4. Then the schedule on a given machine corresponds to a distinct configuration C that can be determined by counting for each possible module size h the batches of jobs from the same class whose length together with the setup time adds up to an overall length of h. Note that the length of this configuration is equal to the used up space on that machine. We fix an arbitrary setup class k and set the variables $x_C^{(k)}$ accordingly (and $x_C^{(k')} = 0$ for $k' \neq k$ and $C \in \mathcal{C}$). By this setting, we get an objective value of at most $m\bar{T} - L$ because there was at least L free space in the schedule. For each class k and module M, we count the number of machines on which there are exactly M_p jobs with processing time p from class k for each $p \in P$ and set $y_M^{(k)}$ accordingly. It is easy to see that the constraints are satisfied by these definitions.

Given a solution (x, y) of the MCIP, we define a corresponding schedule: Because of (6), we can match the machines to configurations such that each machine is matched to exactly one configuration. If machine i is matched to C, for each module size h, we create C_h slots of length h on i. Next, we divide the setup classes into batches. For each class k and module M, we create $y_M^{(k)}$ batches of jobs from class k with M_p jobs with processing time p for each $p \in P$ and place the batch together with the corresponding setup time into a fitting slot on some machine. Because of (8) and (7), all jobs can be placed by this process. Note that the used space equals the overall size of the configurations, and we therefore have free space of at least L. □

Result Using the above results, we can formulate and analyze the following procedure:

Algorithm 1 1. Generate the modified instance I_4:

- Remove the small jobs with small setup times.
- Increase the setup times of the remaining classes with small setup times.
- Replace the tiny jobs with big setup times.
- Round up the resulting processing and setup times.

2. Build and solve the MCIP for I_4.
3. If the MCIP is infeasible, or the objective value greater than $m\bar{T} - L$, report that I has no solution with makespan T.
4. Otherwise build the schedule with makespan $\bar{T}$ and free space at least L for I_4.
5. Transform the schedule into a schedule for I with makespan at most $\breve{T}$:

- Use the prerounding processing and setup times.
- Replace the placeholders by the tiny jobs with big setup times.
- Use the original setup times of the classes with small setup times.
- Insert the small jobs with small setup times into the free space.

The procedure is correct due to the above results. To analyze its running time, we first bound the parameters of the MCIP. We have $|\mathcal{B}| = K' \leq K$ and $D = |P|$ by definition and $|\mathcal{M}| = \mathcal{O}(|S|(1/\varepsilon^3)^{|P|}) = 2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$ because $|S|, |P| \in \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$. This is true due to the last rounding step which also implies $|H| = \mathcal{O}(1/\varepsilon^5)$ yielding $|\mathcal{C}| = |H|^{\mathcal{O}(1/\varepsilon^3)} = 2^{\mathcal{O}(1/\varepsilon^3 \log 1/\varepsilon)}$. According to Observation 1, this yields a brick size of $t = 2^{\mathcal{O}(1/\varepsilon^3 \log 1/\varepsilon)}$, a brick number of K, $r = \mathcal{O}(1/\varepsilon^5)$ globally, and $s = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$ locally uniform constraints for the MCIP. We have $\Delta = \mathcal{O}(1/\varepsilon^5)$ because all occurring values in the processing time matrix are bounded in $\bar{T}$, and we have $\bar{T} = \mathcal{O}(1/\varepsilon^5)$ due to the scaling.

By Theorem 2 and some arithmetic, the MCIP can be solved in time:

$$2^{\mathcal{O}(rs^2)}(rs\Delta)^{\mathcal{O}(r^2s+s^2)}(nt)^{1+o(1)} = 2^{\mathcal{O}(1/\varepsilon^{11}(\log 1/\varepsilon)^2)}K^{1+o(1)}$$

When building the actual schedule, we iterate through the jobs and machines like indicated in the proof of Lemma 5 yielding the following:

Theorem 3 *The algorithm for the setup class model finds a schedule with makespan $(1 + \mathcal{O}(\varepsilon))T$ or correctly determines that there is no schedule with makespan T in time $2^{\mathcal{O}(1/\varepsilon^{11}(\log 1/\varepsilon)^2)}K^{1+o(1)}nm$.*

4.2 Splittable model

The approximation scheme for the splittable model presented in this section is probably the easiest one discussed in this work. There is, however, one problem concerning this procedure: Its running time is polynomial in the number of machines which might be exponential in the input size in this case, since the input only includes the number of machines m with encoding length $\mathcal{O}(\log m)$. For the other two problems, we can assume that we have at most as many machines as jobs (see Sect. 2) and hence this is not an issue. But in the splittable case we may have many more machines than jobs. Note that is not an issue for the other two problems (see Sect. 2). In Sect. 5, we show how this problem can be overcome and further improve the running time.

Simplification of the Instance In this context, the set of big setup jobs $\mathcal{J}^{\text{bst}}$ is given by the jobs with setup times at least εT and the small setup jobs $\mathcal{J}^{\text{sst}}$ are all the others. Let $L = \sum_{j\in\mathcal{J}^{\text{sst}}}(s_j + p_j)$. Because every job has to be scheduled and every setup has to be paid at least once, L is a lower bound on the summed up space due to small jobs in any schedule. Let I_1 be the instance that we get by removing all the small setup jobs from the given instance I.

Lemma 6 *A schedule with makespan T for I induces a (T, L)-schedule for I_1; and any (T', L)-schedule for I_1 can be transformed into a schedule for I with makespan at most $T' + \varepsilon T$.*

Proof The first claim is obvious. Hence, consider a sequence consisting of the jobs from $\mathcal{J}^{\text{sst}}$ together with their setup times where the setup time of a job is the direct predecessor of the job. We insert the setup times and jobs from this sequence greedily into the schedule in a next-fit fashion: Given a machine, we keep inserting the items from the sequence on to the machine at the end of the schedule until the taken up space reaches T'. If the current item does not fit exactly, we cut it such that the used space on the machine is exactly T'. Then we continue with the next machine (without the insertion of an additional setup time). We can place the whole sequence like this without exceeding the makespan T', because we have free space of at least L which is the summed up length of the items in the sequence. Next, we remove each setup time that was placed only partly on a machine together with those that were placed at the end of the schedule Furthermore, we insert a fitting setup time for the jobs that were scheduled without one, which can happen only once for each machine. This yields a feasible schedule whose makespan is increased by at most εT. □

Next, we round up the processing and setup times of I_1 to the next multiple of $\varepsilon^2 T$, that is, for each job $j \in \mathcal{J}$, we set $\bar{p}_j = \lceil p_j/(\varepsilon^2 T)\rceil \varepsilon^2 T$ and $\bar{s}_j = \lceil s_j/(\varepsilon^2 T)\rceil \varepsilon^2 T$. We call the resulting instance I_2 and denote its job set by $\mathcal{J}'$.

Lemma 7 *If there is a (T, L')-schedule for I_1, then there is also a $((1 + 2\varepsilon)T, L')$-schedule for I_2 in which the length of each job part is a multiple of $\varepsilon^2 T$; and any (T', L')-schedule for I_2 yields a (T', L')-schedule for I_1.*

Proof Consider a (T, L)-schedule for I_1. There are at most $1/\varepsilon$ jobs scheduled on each machine since each setup time has a length of at least εT. On each machine, we extend each occurring setup time and the processing time of each occurring job part by at most $\varepsilon^2 T$ to round it to a multiple of $\varepsilon^2 T$. This step extends the makespan by at most $2\varepsilon T$. Since now each job part is a multiple of $\varepsilon^2 T$, the total processing time of the job is a multiple of $\varepsilon^2 T$ too. However, its overall length might be greater than its rounded processing time, and we simply discard some processing time in this case. The second claim is obvious. □

Based on the two Lemmata, we define two makespan bounds $\bar{T} = (1 + 2\varepsilon)T$ and $\breve{T} = \bar{T} + \varepsilon T = (1 + 3\varepsilon)T$. We will use the MCIP to find a $(\bar{T}, L)$-schedule for I_2 in which the length of each job part is a multiple of $\varepsilon^2 T$. Using the two Lemmata, this will yield a schedule with makespan at most $\breve{T}$ for the original instance I.

Utilization of the MCIP The basic objects, in this context, are the (big setup) jobs, i.e., $\mathcal{B} = \mathcal{J}^{\text{bst}} = \mathcal{J}'$, and they have only one value ($D = 1$), namely, their processing time. Moreover, the modules are defined as the set of pairs of job piece sizes and setup times, i.e., $\mathcal{M} = \{[\,|\,\}](q, s)s, q \in \{x\varepsilon^2 T \,|\, x \in \mathbb{Z}, 0 < x \leq 1/\varepsilon^2\}, s \geq \varepsilon T$, and we write $s_M = s$ and $q_M = q$ for each module $M = (q, s) \in \mathcal{M}$. Corresponding to the value of the basic objects, the value of a module M is q_M, and its size $\Lambda(M)$ is given by $q_M + s_M$. A job is eligible for a module if the setup times fit, i.e., $\mathcal{B}_M = \{j \in \mathcal{J}' \,|\, s_j = s_M\}$. In order to ensure integral values, we establish $\varepsilon^2 T = 1$ via a simple scaling step. The set of configurations $\mathcal{C}$ is comprised of all configurations of module sizes H that are bounded in size by $\bar{T}$, i.e., $\mathcal{C} = \mathcal{C}_H(\bar{T})$. We state the constraints of the MCIP for the above definitions with adapted notation and without duplication of the configuration variables:

$$\sum_{C\in\mathcal{C}} x_C = m \tag{9}$$

$$\sum_{C\in\mathcal{C}} C_h x_C = \sum_{j\in\mathcal{J}'} \sum_{M\in\mathcal{M}(h)} y_M^{(j)} \qquad \forall h \in H \tag{10}$$

$$\sum_{M \in \mathcal{M}} q_M y_M^{(j)} = p_j \qquad \forall j \in \mathcal{J}' \qquad (11)$$

Note that we additionally minimize the summed up size of the configurations via the objective function $\sum_C \Lambda(C)x_C$.

Lemma 8 *With the above definitions, there is a* $(\bar{T}, L)$*-schedule for* I_2 *in which the length of each job piece is a multiple of* $\varepsilon^2 T$ *if and only if the* MCIP *has a solution with objective value at most* $m\bar{T} - L$.

Proof Given such a schedule for I_2, the schedule on each machine corresponds to exactly one configuration C that can be derived by counting the job pieces and setup times with the same summed up length h and setting C_h accordingly. This yields the values for the x variables. The size of the configuration C is equal to the used space on the respective machine. Hence, the objective value is bounded by $m\bar{T} - L$. Furthermore, for each job j and job part length q, we count the number of times a piece of j with length q is scheduled and set $y_{(q,s_j)}^{(j)}$ accordingly. It is easy to see that the constraints are satisfied.

Now, let (x, y) be a solution to the MCIP with objective value at most $m\bar{T} - L$. We use the solution to construct a schedule: For each configuration C we reserve x_C machines. On each of these machines we create C_h slots of length h for each module size $h \in H$. Note that because of (9), there is the exact right number of machines for this. Next, consider each job j and possible job part length q and create $y_{(q,s_j)}^{(j)}$ split pieces of length q and place them together with a setup of s_j into a slot of length $s_j + q$ on any machine. Because of (11), the entire job is split up by this, and because of (10), there are enough slots for all the job pieces. Note that the used space in the created schedule is equal to the objective value of (x, y) and therefore there is at least L free space. □

Result Summing up, we can find a schedule of length at most $(1 + 3\varepsilon)T$ or correctly determine that there is no schedule of length T with the following procedure:

Algorithm 2 1. Generate the modified instance I_2:

- Remove the small setup jobs.
- Round the setup and processing times of the remaining jobs.

2. Build and solve the MCIP for this case.
3. If the IP is infeasible, or the objective value greater than $m\bar{T} - L$, report that I has no solution with makespan T.
4. Otherwise build the schedule with makespan $\bar{T}$ and free space at least L for $\bar{I}$.
5. Transform the schedule into a schedule for I with makespan at most $\check{T}$:
   - Use the original processing and setup times.
   - Greedily insert the small setup jobs.

To assess the running time of the procedure, we mainly need to bound the parameters of the MCIP, namely $|\mathcal{B}|$, $|H|$, $|\mathcal{M}|$, $|\mathcal{C}|$ and D. By definition, we have $|\mathcal{B}| = |\mathcal{J}'| \leq n$ and $D = 1$. Since all setup times and job piece lengths are multiples of $\varepsilon^2 T$ and bounded by T, we have $|\mathcal{M}| = \mathcal{O}(1/\varepsilon^4)$ and $|H| = \mathcal{O}(1/\varepsilon^2)$. This yields $|\mathcal{C}| \leq |H|^{\mathcal{O}(1/\varepsilon)} = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ because the size of each module is at least εT and the size of the configurations bounded by $(1 + 2\varepsilon)T$.

According to Observation 1, we now have brick-size $t = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$, brick number $|\mathcal{B}| = n$, $r = |H| + 1 = \mathcal{O}(1/\varepsilon^2)$ globally uniform and $s = D = 1$ locally uniform constraints. Because of the scaling step, all occurring numbers in the constraint matrix of the MCIP are bounded by $\mathcal{O}(1/\varepsilon^2)$, and therefore we have $\Delta = \mathcal{O}(1/\varepsilon^2)$. Hence, the MCIP can be solved in time:

$$2^{\mathcal{O}(rs^2)}(rs\Delta)^{\mathcal{O}(r^2 s+s^2)}(nt)^{1+o(1)} = 2^{\mathcal{O}(1/\varepsilon^4 \log 1/\varepsilon)}n^{1+o(1)}$$

While the first step of the procedure is obviously dominated by the above, this is not the case for the remaining ones. In particular, building the schedule from the IP solution has linear costs in both n and m if the procedure described in the proof of Lemma 8 is realized in a straight-forward fashion. Note that the number of machines m could be exponential in the number of jobs, and therefore the described procedure is a PTAS only for the special case of $m = \mathrm{poly}(n)$. However, this limitation can be overcome with a little extra effort, as we discuss in Sect. 5.

Theorem 4 *The algorithm for the splittable model finds a schedule with makespan at most* $(1 + 3\varepsilon)T$ *or correctly determines that there is no schedule with makespan* T *in time* $2^{\mathcal{O}(1/\varepsilon^4 \log 1/\varepsilon)}n^{1+o(1)}m$.

4.3 Preemptive model

In the preemptive model, we have to actually consider the timeline of the schedule on each machine, instead of just the assignment of the jobs or job pieces, and this causes some difficulties. For instance, we will have to argue that it suffices to look for a schedule with few possible starting points, and we will have to introduce additional constraints in the IP in order to ensure that pieces of the same job do not overlap. Our first step in dealing with this extra difficulty is to introduce some concepts and notation: For a given schedule with a makespan bound T, we call a job piece together with its setup a *block*, and we call the schedule X-layered, for some value X, if each block starts at a multiple of X. Corresponding to this, we call the time in the schedule between two directly succeeding multiples of X a *layer* and the corresponding time on a single machine a *slot*. We number the layers bottom to top and identify them with their number, that is, the set of layers Ξ is given by $\{\ell \in \mathbb{Z}_{>0} \mid (\ell - 1)X \leq T\}$. Note that in an X-layered schedule there is at most one block in each slot, and for each layer there can be at most one block of each job present. Furthermore, we slightly alter the definition of free space for X-layered schedules: We solely count the space from slots that are completely free. If in such a schedule for each job there is at most one slot occupied by this job but not fully filled, we additionally call the schedule *layer-compliant*.

Table 2 Overview on the job classifications

p_j	s_j		
	$< \mu T$	$\geq \mu T, < \delta T$	$\geq \delta T$
$< \varepsilon T$	$\mathcal{J}^{\text{sst}}_{\text{small}}$	$\mathcal{J}^{\text{mst}}_{\text{small}}$	$\mathcal{J}^{\text{bst}}_{\text{small}}$
$\geq \varepsilon T$	$\mathcal{J}^{\text{sst}}_{\text{big}}$	$\mathcal{J}^{\text{mst}}_{\text{big}}$	$\mathcal{J}^{\text{bst}}_{\text{big}}$

Simplification of the instance

In the preemptive model, we distinguish *big*, *medium* and *small* setup jobs using two parameters δ and μ: The big setup jobs $\mathcal{J}^{\text{bst}}$ are those with setup time at least δT, the small $\mathcal{J}^{\text{sst}}$ have a setup time smaller than μT, and the medium $\mathcal{J}^{\text{mst}}$ are the ones in between. We set $\mu = \varepsilon^2 \delta$ and choose $\delta \in \{\varepsilon^1, \ldots, \varepsilon^{2/\varepsilon^2}\}$ such that the summed up processing time together with the summed up setup time of the medium setup jobs is upper bounded by $m\varepsilon T$, i.e., $\sum_{j \in \mathcal{J}^{\text{mst}}} (s_j + p_j) \leq m\varepsilon T$. If there is a schedule with makespan T, such a choice is possible because of the pidgeon hole principle and because the setup time of each job has to occur at least once in any schedule. Similar arguments are widely used, e.g., in the context of geometrical packing algorithms. Furthermore, we distinguish the jobs by processing times calling those with processing time at least εT *big* and the others *small*. For a given set of jobs J, we call the subsets of big or small jobs J_{big} or J_{small}, respectively. An overview of the job classification is provided in Table 2. We perform three simplification steps, aiming for an instance in which the small and medium setup jobs are big; small setup jobs have setup time 0; and for which an $\varepsilon\delta T$-layered, layer-compliant schedule exists. The rationale behind the above approach will only become clear step by step in the following, and we kindly ask the reader to be patient. In particular, we moved a particularly complicated proof to the end of this part.

Let I_1 be the instance we get by removing the small jobs with medium setup times $\mathcal{J}^{\text{mst}}_{\text{small}}$ from the given instance I.

Lemma 9 *If there is a schedule with makespan at most T for I, then there is also such a schedule for I_1; and if there is a schedule with makespan at most T' for I_1, then there is a schedule with makespan at most $T' + (\varepsilon + \delta)T$ for I.*

Proof The first claim is obvious. For the second, we create a sequence containing the jobs from $\mathcal{J}^{\text{mst}}_{\text{small}}$ each directly preceded by its setup time. Recall that the overall length of the objects in this sequence is at most $m\varepsilon T$, and the length of each job is bounded by εT. We greedily insert the objects from the sequence considering each machine in turn. On the current machine, we start at time $T' + \delta T$ and keep inserting until $T' + \delta T + \varepsilon T$ is reached. If the current object is a setup time, we discard it and continue with the next machine and object. If, on the other hand, it is a job, we split it such that the remaining space on the current machine can be perfectly filled. We can place all objects like this, however the first job part placed on a machine might be missing a setup. We can insert the missing setups because they have length at most δT and between time T' and $T' + \delta T$ there is free space. □

Next, we consider the jobs with small setup times: Let I_2 be the instance we get by removing the small jobs with small setup times $\mathcal{J}^{\text{sst}}_{\text{small}}$ and setting the setup time of the big jobs with small setup times to zero, i.e., $\bar{s}_j = 0$ for each $j \in \mathcal{J}^{\text{sst}}_{\text{big}}$. Note that in the resulting instance each small job has a big setup time. Furthermore, let $L := \sum_{j\in\mathcal{J}^{\text{sst}}_{\text{small}}} p_j + s_j$. Then L is an obvious lower bound for the space taken up by the jobs from $\mathcal{J}^{\text{sst}}_{\text{small}}$ in any schedule.

Lemma 10 *If there is a schedule with makespan at most T for I_1, then there is also a (T, L)-schedule for I_2; and if there is a γT-layered (T', L)-schedule for I_2 with T' a multiple of γT, then there is also a schedule with makespan at most $(1+\gamma^{-1}\mu)T' + (\mu+\varepsilon)T$ for I_1.*

Proof The first claim is obvious, and for the second consider a γT-layered (T', L)-schedule for I_2. We create a sequence that contains the jobs of $\mathcal{J}^{\text{sst}}_{\text{small}}$ and their setups such that each job is directly preceded by its setup. Remember that the remaining space in partly filled slots is not counted as free space. Hence, since the overall length of the objects in the sequence is L, there is is enough space in the free slots of the schedule to place them. We do so in a greedy fashion guaranteeing that each job is placed on exactly one machine: We insert the objects from the sequence into the free slots considering each machine in turn, starting on the current machine from the beginning of the schedule, and moving on towards its end. If an object cannot be fully placed into the current slot there are two cases: It could be a job or a setup. In the former case, we cut it and continue placing it in the next slot, or, if the current slot was the last one, we place the rest at the end of the schedule. In the latter case, we discard the setup and continue with the next slot and object. The resulting schedule is increased by at most εT, which is caused by the last job placed on a machine.

To get a proper schedule for I_1 we have to insert some setup times: For the large jobs with small setup times and for the jobs that were cut in the greedy procedure. We do so by inserting a time window of length μT at each multiple of γT and at the end of the original schedule on each machine. By this, the schedule is increased by at most $\gamma^{-1}\mu T' + \mu T$. Since all the job parts in need of a setup are small and did start at multiples of μT or at the end, we can insert the missing setups. Note that blocks that span over multiple layers are cut by the inserted time windows. This, however, can easily be repaired by moving the cut pieces properly down. □

We continue by rounding the medium and big setup and all the processing times. In particular, we round the processing times and the big setup times up to the next multiple of $\varepsilon\delta T$ and the medium setup times to the next multiple of $\varepsilon\mu T$, i.e., $\bar{p}_j = \lceil p_j/(\varepsilon\delta T)\rceil \varepsilon\delta T$ for each job j, $\bar{s}_j = \lceil s_j/(\varepsilon\delta T)\rceil \varepsilon\delta T$ for each big setup job $j \in \mathcal{J}^{\text{bst}}$, and $\bar{s}_j = \lceil s_j/(\varepsilon\mu T)\rceil \varepsilon\mu T$ for each medium setup job $j \in \mathcal{J}^{\text{mst}}_{\text{big}}$.

Lemma 11 *If there is a (T, L)-schedule for I_2, then there is also an $\varepsilon\delta T$-layered, layer-compliant $((1+3\varepsilon)T, L)$-schedule for I_3; and if there is a γT-layered (T', L)-schedule for I_3, then there is also such a schedule for I_2.*

While the second claim is easy to see, the proof of the first is rather elaborate and unfortunately a bit tedious. Hence, since we believe Lemma 11 to be fairly plausible by itself, we postpone its proof to the end of the section and proceed discussing its use.

For the big and small setup jobs, both processing and setup times are multiples of $\varepsilon\delta T$. Therefore, the length of each of their blocks in an $\varepsilon\delta T$-layered, layer-compliant schedule is a multiple of $\varepsilon\delta T$. For a medium setup job, on the other hand, we know that the overall length of its blocks has the form $x\varepsilon\delta T + y\varepsilon\mu T$, with non-negative integers x and y. In particular, it is a multiple of $\varepsilon\mu T$ because $\varepsilon\delta T = (1/\varepsilon^2)\varepsilon\mu T$. In a $\varepsilon\delta T$-layered, layer-compliant schedule, for each medium setup job the length of all but at most one block is a multiple of $\varepsilon\delta T$ and therefore a multiple of $\varepsilon\mu T$. If both the overall length and the lengths of all but one block are multiples of $\varepsilon\mu T$, this is also true for the one remaining block. Hence, we will use the MCIP not to find an $\varepsilon\delta T$-layered, layer-compliant schedule in particular, but an $\varepsilon\delta T$-layered one with block sizes as described above and maximum free space.

Based on the simplification steps, we define two makespan bounds $\bar{T}$ and $\check{T}$: Let $\bar{T}$ be the makespan bound we get by the application of the Lemmata 9–11, i.e., $\bar{T} = (1+3\varepsilon)T$. We will use the MCIP to find an $\varepsilon\delta T$-layered $(\bar{T}, L)$-schedule for I_3 and apply the Lemmata 9–11 backwards to get a schedule for I with makespan at most $\check{T} = (1+(\varepsilon\delta)^{-1}\mu)\bar{T} + (\mu+\varepsilon)T + (\varepsilon+\delta)T \leq (1+9\varepsilon)T$ (using $\varepsilon \leq 1/2$).

Utilization of the MCIP

Similar to the splittable case, the basic objects are the (big) jobs, i.e., $\mathcal{B} = \mathcal{J}_{\text{big}}$, and their single value is their processing time ($D = 1$). The modules, on the other hand, are more complicated, because they additionally need to encode which layers are exactly used and, in case of the medium jobs, to which degree the last layer is filled. For the latter, we introduce buffers, representing the unused space in the last layer and define modules as tuples (ℓ, q, s, b) of starting layer, job piece size, setup time and buffer size. For a module $M = (\ell, q, s, b)$, we write $\ell_M = \ell$, $q_M = q$, $s_M = s$ and $b_M = b$, and we define the size $\Lambda(M)$ of M as $s + q + b$. The overall set of modules $\mathcal{M}$ is the union of the modules for big, medium and small setup jobs $\mathcal{M}^{\text{bst}}$, $\mathcal{M}^{\text{mst}}$ and $\mathcal{M}^{\text{sst}}$ that are defined as follows. Let $Q^{\text{bst}} = \{q \mid q = x\varepsilon\delta T, x \in \mathbb{Z}_{>0}, q \leq \bar{T}\}$ and $Q^{\text{mst}} = \{q \mid q = x\varepsilon\mu T, x \in \mathbb{Z}_{>0}, q \leq \bar{T}\}$ be the sets of possible job piece sizes of big and medium setup jobs; $S^{\text{bst}} = \{s \mid s = x\varepsilon\delta T, x \in \mathbb{Z}_{\geq 1/\varepsilon}, s \leq \bar{T}\}$ and $S^{\text{mst}} = \{s \mid s = x\varepsilon\mu T, x \in \mathbb{Z}_{\geq 1/\varepsilon}, s \leq \delta T\}$ be the sets of possible big and medium setup times; $B = \{b \mid b = x\varepsilon\mu T, x \in \mathbb{Z}_{\geq 0}, b < \varepsilon\delta T\}$ the set of possible buffer sizes; and $\Xi = \{1, \dots, 1/(\varepsilon\delta) + 3/\delta\}$ the set of layers. We set:

$$\mathcal{M}^{\text{bst}} = \{[\,|\,\}](\ell, q, s, 0)\ell \in \Xi, q \in Q^{\text{bst}}, s \in S^{\text{bst}}, (\ell-1)\varepsilon\delta T + s + q \leq \bar{T}$$

$$\mathcal{M}^{\text{mst}} = \big\{(\ell, q, s, b) \in \Xi \times Q^{\text{mst}} \times S^{\text{mst}} \times B \,\big|\, x = s + q + b \in \varepsilon\delta T\mathbb{Z}_{>0}, (\ell-1)\varepsilon\delta T + x \leq \bar{T}\big\}$$

$$\mathcal{M}^{\text{sst}} = \{[\,|\,\}](\ell, \varepsilon\delta T, 0, 0)\ell \in \Xi$$

Concerning the small setup modules, note that the small setup jobs have a setup time of 0 and therefore may be covered slot by slot. We establish $\varepsilon\mu T = 1$ via scaling, to ensure integral values. A big, medium or small job is eligible for a module if it is also big, medium or small, respectively, and the setup times fit.

We have to avoid that two modules M_1, M_2 whose corresponding time intervals overlap are used to cover the same job or in the same configuration. Such an overlap occurs if there is some layer ℓ used by both of them, that is, $(\ell_M - 1)\varepsilon\delta T \leq (\ell - 1)\varepsilon\delta T < (\ell_M - 1)\varepsilon\delta T + \Lambda(M)$ for both $M \in \{M_1, M_2\}$. Hence, for each layer $\ell \in \Xi$, we set $\mathcal{M}_\ell \subseteq \mathcal{M}$ to be the set of modules that use layer ℓ. Furthermore, we partition the modules into groups Γ by size and starting layer, i.e., $\Gamma = \{G \subseteq \mathcal{M} \mid M, M' \in G \iff \Lambda(M) = \Lambda(M') \wedge \ell_M = \ell_{M'}\}$. The size of a group $G \in \Gamma$ is the size of a module from G, i.e., $\Lambda(G) = \Lambda(M)$ for $M \in G$. Unlike before we consider *configurations of module groups* rather than module sizes. More precisely, the set of configurations $\mathcal{C}$ is given by the configurations of groups such that for each layer at most one group using this layer is chosen, i.e., $\mathcal{C} = \{C \in \mathbb{Z}_{\geq 0}^{\Gamma} \mid \forall \ell \in \Xi : \sum_{G \subseteq \mathcal{M}_\ell} C_G \leq 1\}$. With this definition we prevent overlap conflicts on the machines. Note that unlike in the cases considered so far, the size of a configuration does not correspond to a makespan in the schedule, but to used space, and the makespan bound is realized in the definition of the modules instead of in the definition of the configurations. To also avoid conflicts for the jobs, we extend the basic MCIP with additional locally uniform constraints. In particular, the constraints of the extended MCIP for the above definitions with adapted notation and without duplication of the configuration variables are given by:

$$\sum_{C \in \mathcal{C}} x_C = m \tag{12}$$

$$\sum_{C \in \mathcal{C}(T)} C_G x_C = \sum_{j \in \mathcal{J}} \sum_{M \in G} y_M^{(j)} \qquad \forall G \in \Gamma \tag{13}$$

$$\sum_{M \in \mathcal{M}} q_M y_M^{(j)} = p_j \qquad \forall j \in \mathcal{J} \tag{14}$$

$$\sum_{M \in \mathcal{M}_\ell} y_M^{(j)} \leq 1 \qquad \forall j \in \mathcal{J}, \ell \in \Xi \tag{15}$$

Like in the first two cases, we minimize the summed-up size of the configurations via the objective function $\sum_C \Lambda(C) x_C$. Note that in this case the size of a configuration does not have to equal its height. It is easy to see that the last constraint is indeed locally uniform. However, since we have an inequality instead of an equality, we have to introduce $|\Xi|$ slack variables in each brick, yielding:

Observation 2 The MCIP extended like above is an n-fold IP with brick-size $t = |\mathcal{M}| + |\mathcal{C}| + |\Xi|$, brick number $n = |\mathcal{J}|$, $r = |\Gamma| + 1$ globally uniform and $s = D + |\Xi|$ locally uniform constraints.

Lemma 12 *With the above definitions, there is an $\varepsilon\delta T$-layered $(\bar{T}, L)$-schedule for I_3 in which the length of a block is a multiple of $\varepsilon\delta T$ if it belongs to a small or big setup job or a multiple of $\varepsilon\mu T$ otherwise, if and only if the extended* MCIP *has a solution with objective value at most $m\bar{T} - L$.*

Proof We first consider such a schedule for I_3. For each machine, we can derive a configuration that is given by the starting layers of the blocks together with the summed-up length of the slots the respective block is scheduled in. The size of the configuration C is equal to the used space on the respective machine. Hence, we can fix some arbitrary job j and set $x_C^{(j)}$ to the number of machines corresponding to j (and $x_C^{(j')} = 0$ for $j' \neq j$). Keeping in mind that in an $\varepsilon\delta T$-layered schedule the free space is given by the free slots, the above definition yields an objective value bounded by $m\bar{T} - L$ because there was free space of at least L. Next, we consider the module variables for each job j in turn: If j is a small setup job, we set $y_{(\ell,\varepsilon\delta T,0,0)}^{(j)}$ to 1 if j occurs in ℓ and to 0 otherwise. Now, let j be a big setup job. For each of its blocks, we set $y_{(\ell,z-s_j,s_j,0)}^{(j)} = 1$, where ℓ is the starting layer and z the length of the block. The remaining variables are set to 0. Lastly, let j be a medium setup job. For each of its blocks, we set $y_{(\ell,z-s_j,s_j,b)}^{(j)} = 1$, where ℓ is the starting layer of the block, z its length and $b = \lceil z/(\varepsilon\delta T)\rceil \varepsilon\delta T - z$. Again, the remaining variables are set to 0. It is easy to verify that all constraints are satisfied by this solution.

If, on the other hand, we have a solution (x, y) to the MCIP with objective value at most $m\bar{T} - L$, we reserve $\sum_j x_C^{(j)}$ machines for each configuration C. There are enough machines to do this, because of (12). On each of these machines we reserve space: For each $G \in \Gamma$, we create an allocated space of length $\Lambda(G)$ starting from the starting layer of G if $C_G = 1$. Let j be a job and ℓ be a layer. If j has a small setup time, the variable $y_{(\ell,\varepsilon\delta T,0,0)}^{(j)}$ may have the value 0 or 1. In the latter case, we create a piece of length $\varepsilon\delta T$ and place it into an allocated space of length $\varepsilon\delta T$ in layer ℓ. If, on the other hand, j is a big or medium setup job, we consider each possible job part length $q \in Q^{\text{bst}}$ or $q \in Q^{\text{mst}}$, respectively, create $y_{(\ell,q,s_j,0)}^{(j)}$ or $y_{(\ell,q,s_j,b)}^{(j)}$ (with $b = \lceil q/(\varepsilon\delta T)\rceil \varepsilon\delta T - \varepsilon\delta T$) pieces of length q, and place them together with their setup time into allocated spaces of length q in layer ℓ. Because of (14), the entire job is split up by this, and because of (13), there are enough allocated spaces for all the job pieces. The makespan bound is ensured by the definition of the modules, and overlaps are avoided due to the definition of the configurations and (15). Furthermore, the used slots have an overall length equal to the objective value of (x, y) and therefore there is at least L free space. □

Result

Summing up the above considerations, we get:

Algorithm 3

1. Determine a suitable class of medium setup jobs. If there is no such class, report that there is no schedule with makespan T and terminate the procedure.
2. Generate the modified instance I_3:
   - Remove the small jobs with medium setup times.
   - Remove the small jobs with small setup times, and decrease the setup time of big jobs with small setup time to 0.
   - Round the big processing times, as well as the medium, and the big setup times.

3. Build and solve the MCIP for I_3.
4. If the MCIP is infeasible, or the objective value greater than $m\bar{T} - L$, report that I has no solution with makespan T.
5. Otherwise build the $\varepsilon\delta T$-layered schedule with a makespan of at most $\bar{T}$ and a free space of at least L for I_3.
6. Transform the schedule into a schedule for I with makespan at most $\breve{T}$:
   - Use the prerounding processing and setup times.
   - Insert the small jobs with small setup times into the free slots and insert the setup times of the big jobs with small setup times.
   - Insert the small jobs with medium setup times.

We analyze the running time of the procedure and start by bounding the parameters of the extended MCIP. We have $|\mathcal{B}| = n$ and $D = 1$ by definition, and the number of layers $|\Xi|$ is obviously $\mathcal{O}(1/(\varepsilon\delta)) = \mathcal{O}(1/\varepsilon^{2/\varepsilon+1}) = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$. Furthermore, it is easy to see that $|Q^{\text{bst}}| = \mathcal{O}(1/(\varepsilon\delta))$, $|Q^{\text{mst}}| = \mathcal{O}(1/(\varepsilon^3\delta))$, $|S^{\text{bst}}| = \mathcal{O}(1/(\varepsilon\delta))$, $|S^{\text{mst}}| = \mathcal{O}(1/(\varepsilon^3))$, and $|B| = \mathcal{O}(1/\varepsilon^2)$. This gives us $\mathcal{M}^{\text{bst}} \leq |\Xi||Q^{\text{bst}}||S^{\text{bst}}|$, $\mathcal{M}^{\text{mst}} \leq |\Xi||Q^{\text{mst}}||S^{\text{mst}}||B|$ and $\mathcal{M}^{\text{sst}} = |\Xi|$, and therefore $|\mathcal{M}| = |\mathcal{M}^{\text{bst}}| + |\mathcal{M}^{\text{mst}}| + |\mathcal{M}^{\text{sst}}| = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$. Since their are $\mathcal{O}(1/(\delta\varepsilon))$ distinct module sizes, the number of groups $|\Gamma|$ can be bounded by $\mathcal{O}(|\Xi|/(\varepsilon\delta)) = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$. Hence, for the number of configurations we get $|\mathcal{C}| = \mathcal{O}((1/(\varepsilon\delta))^{|\Gamma|}) = 2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}}$. By Observation 2, the modified MCIP has $r = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ many globally and $s = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ many locally uniform constraints; its brick number is n, and its brick size is $t = 2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}}$. All occurring values in the matrix are bounded by $\bar{T}$ yielding $\Delta \leq \bar{T} = 1/(\varepsilon\mu) + 1/\mu = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ due to the scaling step. Hence, we can solve the MCIP in time:

$$2^{\mathcal{O}(rs^2)}(rs\Delta)^{\mathcal{O}(r^2s+s^2)}(nt)^{1+o(1)} = 2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}} n^{1+o(1)}$$

A straight-forward realization of the procedure for the creation of the $\varepsilon\delta T$-layered $(\bar{T}, L)$-schedule for I_3 (the fifth step), which is described in the proof of Lemma 12, is linear with respect to m, yielding:

Theorem 5 *The algorithm for the preemptive model finds a schedule with makespan at most* $(1 + 9\varepsilon)T$ *or correctly determines that there is no schedule with makespan* T *in time* $2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}} n^{1+o(1)} m$.

Proof of Lemma 11

We divide the proof into three steps, which can be summarized as follows:

1. We transform a (T, L)-schedule for I_2 into a $((1 + 3\varepsilon)T, L)$-schedule for I_3 in which the big setup jobs are already properly placed inside the layers.
2. We construct a flow network with integer capacities and a maximum flow based on the placement of the remaining jobs in the layers.
3. Using flow integrality and careful repacking, we transform the schedule into a $\varepsilon\delta T$-layered, layer-compliant schedule.

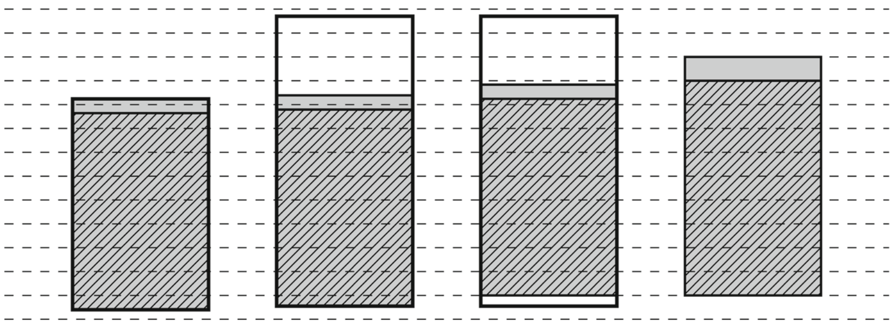

Fig. 2 The stretching and rounding steps, for a small job part with big setup time starting in the first layer of the schedule, depicted from left to right: The schedule and the containers are stretched; the block is moved up; and the processing and the setup time are increased. The hatched part represents the setup time, the thick rectangle the container, and the dashed lines the layers, with $\varepsilon = \delta = 1/8$

More precisely, the above transformation steps will produce a $\varepsilon\delta T$-layered, layer-compliant $((1 + 3\varepsilon)T, L)$-schedule with the additional properties that too much processing time may be inserted for some jobs or setup times are produced that are not followed by the corresponding job pieces. Note that this does not cause any problems: We can simply remove the extra setups and processing time pieces. For the medium jobs, this results in a placement with at most one used slot that is not fully filled, as required in a layer-compliant schedule.

Step 1 Remember that a block is a job piece together with its setup time placed in a given schedule. Consider a (T, L)-schedule for I_2 and suppose that for each block in the schedule there is a container perfectly encompassing it. Now, we stretch the entire schedule by a factor of $(1 + 3\varepsilon)$ and in this process we stretch and move the containers correspondingly. The blocks are not stretched but moved in order to stay in their container, and we assume that they are positioned at the bottom, that is, at the beginning of the container. Note that we could move each block inside its respective container without creating conflicts with other blocks belonging to the same job. In the following, we use the extra space to modify the schedule. Similar techniques are widely used in the context of geometric packing algorithms.

Let j be a big setup job. In each container containing a block belonging to j, there is a free space of at least $3\varepsilon\delta T$ because the setup time of j is at least δT and therefore the container had at least that length before the stretching. Hence, we have enough space to perform the following two steps. We move the block up by at most $\varepsilon\delta T$ such that it starts at a multiple of $\varepsilon\delta T$. Next, we enlarge the setup time and the processing time by at most $\varepsilon\delta T$ such that both are multiples of $\varepsilon\delta T$. Now the setup time is equal to the rounded setup time, while the processing time might be bigger because we performed this step for each piece of the job. We outline the procedure in Fig. 2.

We continue with the small setup jobs. These jobs are big and therefore for each of them there is a summed up free space of at least $3\varepsilon^2 T$ in the containers belonging to the respective job—more than enough to enlarge some of the pieces such that their overall length matches the rounded processing time.

Lastly, we consider the medium setup jobs. These jobs are big as well and we could apply the same argument as above, but we need to be a little bit more careful in order to additionally realize the rounding of the setup times and an additional technical step we need in the following. Fix a medium setup job j and a container filled with a block belonging to j. Since the setup time has a length of at least μT, the part of the container filled with it was increased by at least $3\varepsilon\mu T$. Hence, we can enlarge the setup time to the rounded setup time without using up space in the container that was created due to the processing time part. We do this for all blocks belonging to medium setup jobs. The extra space in the containers of a medium setup job due to the processing time parts is still at least $3\varepsilon^2 T \geq 3\varepsilon\delta T$. For each medium setup job j, we spend at most $\varepsilon\delta T$ of this space to enlarge its processing time to its rounded size and again at most $\varepsilon\delta T$ to create a little bit of extra processing time in the containers belonging to j. The size of this extra processing time is bounded by $\varepsilon\delta T$ and chosen in such a way that the overall length of all blocks belonging to j in the schedule is also a multiple of $\varepsilon\delta T$. Because of the rounding, the length of the added extra processing time for each j is a multiple of $\varepsilon\mu T$. The purpose of the extra processing time is to ensure integrality in the flow network, which is constructed in the next step.

Note that the free space that was available in the original schedule was not used in the above steps, in fact, it was even increased by the stretching. Hence, we have created a $((1+3\varepsilon)T, L)$-schedule for I_3—or a slightly modified version thereof—and the big setup jobs are already well-behaved with respect to the $\varepsilon\delta T$-layers, that is, they start at multiples of $\varepsilon\delta T$ and fully fill the slots they are scheduled in.

Step 2 Note that for each job j and layer $\ell \in \Xi$, the overall length $q_{j,\ell}$ of job and setup pieces belonging to j and placed in ℓ is bounded by $\varepsilon\delta T$. We say that j is *fully*, or *partially*, or *not* scheduled in layer ℓ if $q_{j,\ell} = 1$, or $q_{j,\ell} \in (0, 1)$, or $q_{j,\ell} = 0$, respectively. Let X_j be the set of layers in which j is scheduled partially and Y_ℓ the set of (medium or small setup) jobs partially scheduled in ℓ. Then $a_j = \sum_{\ell \in X_j} q_{j,\ell}$ is a multiple of $\varepsilon\delta T$, and we set $n_j = a_j/(\varepsilon\delta T)$. Furthermore, let $b_\ell = \sum_{j \in Y_\ell} q_{j,\ell}$ and $k_\ell = \lceil b_\ell/(\varepsilon\delta T)\rceil$.

Our flow network has the following structure: There is a node v_j for each medium or small setup job, a node u_ℓ for each layer ℓ, as well as a source α and a sink ω. The source node is connected to the job nodes via edges (α, v_j) with capacity n_j, and the layer nodes are connected to the sink via edges (u_ℓ, ω) with capacity k_ℓ. Lastly, there are edges (v_j, u_ℓ) between job and layer nodes with capacity 1 if j is partially scheduled in layer ℓ or 0 otherwise. In Fig. 3, a sketch of the network is given.

The schedule can be used to define a flow f with value $\sum_j n_j$ in the network by setting $f(\alpha, v_j) = n_j$, $f(u_\ell, \omega) = b_\ell/(\varepsilon\delta T)$, and $f(v_j, u_\ell) = q_{j,\ell}/(\varepsilon\delta T)$. It is easy to verify that f is a maximum flow, and because all capacities in the flow network are integral, we can find another maximum flow f' with integral values.

Step 3 We start by introducing some notation and a basic operation for the transformation of the schedule: Given two machines i and i' and a time t, a *machine swap* between i and i' at moment t produces a schedule in which everything that was scheduled on i from t on is now scheduled on i' and vice versa. If on both machines there is

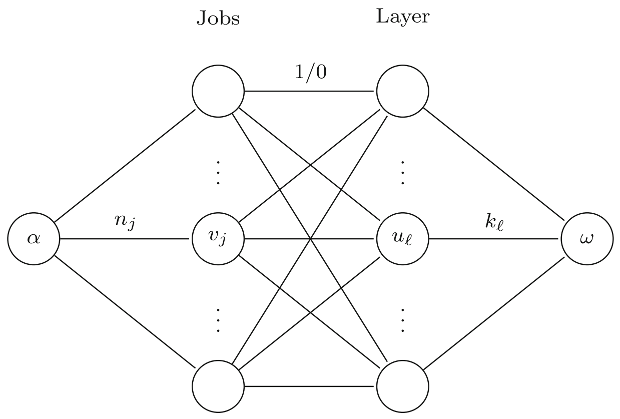

Fig. 3 Flow network for layers and partially scheduled jobs

either nothing scheduled at t, or blocks are starting or ending at t, the resulting schedule is still feasible. Moreover, if there is a block starting at t on one of the machines and another one belonging to the same job ending on the other, we can merge the two blocks and transform the setup time of the first into processing time. We assume in the following that we always merge if this is possible when performing a machine swap. Remember that by definition blocks belonging to the same job cannot overlap. However, if there was overlap, it could be eliminated using machine swaps [33].

If a given slot only contains pieces of jobs that are partially scheduled in the layer, we call the slot *usable*. Furthermore, we say that a job j is *flow assigned* to layer ℓ if $f'(v_j, u_\ell) = 1$. In the following, we will iterate through the layers, create as many usable slots as possible, reserve them for flow assigned jobs, and fill them with processing and setup time of the corresponding jobs later on. To do so, we have to distinguish different types of blocks belonging to jobs that are partially placed in a given layer: Inner blocks which lie completely inside the layer and touch at most one of its borders, upper cross-over blocks which start inside the layer and end above it, and lower cross-over blocks which start below the layer and end inside it. When manipulating the schedule layer by layer, the cross-over jobs obviously can cause problems. To deal with this, we will need additional concepts: A *repair piece* for a given block is a piece of setup time of length less than $\varepsilon\delta T$, with the property that the block and the repair piece together make up exactly one setup of the respective job. Hence, if a repair-piece is given for a block, the block is comprised completely of setup time. Moreover, we say that a slot reserved for a job j has a *dedicated setup* if there is a block of j including a full setup *starting or ending* inside the slot.

In the following, we give a detailed description of the transformation procedure followed by a high-level summarization. The procedure runs through two phases. In the first phase the layers are transformed one after another from bottom to top. After a layer is transformed the following invariants will always hold:

1. A scheduled block either includes a full setup or has a repair piece. In the latter case it was an upper cross-over block in a previous iteration.
2. Reserved slots that are not full have a dedicated setup.

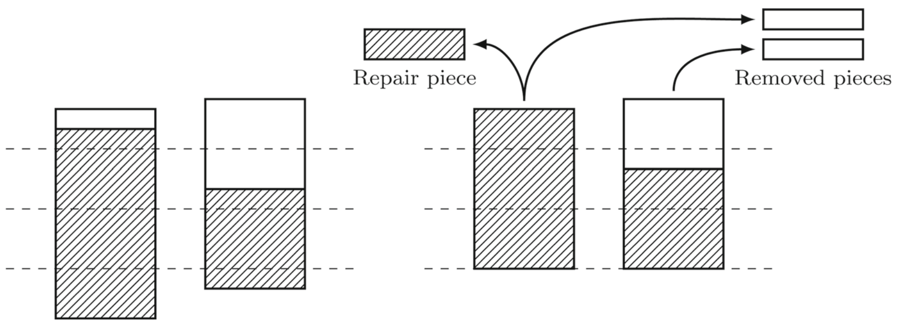

Fig. 4 The rectangles represent blocks, the hatched parts the setup times, and the dashed lines layer borders. The push and cut step is performed on two blocks. For one of the two a repair piece is created

Note that the invariants are trivially fulfilled in the beginning. During the first phase, we remove some job and setup parts from the schedule that are reinserted into the reserved slots in the second phase. Let $\ell \in \Xi$ denote the current layer.

In the first step, our goal is to ensure that jobs that are fully scheduled in ℓ occupy exactly one slot thereby creating as many usable slots as possible. Let j be a job that is fully scheduled in layer ℓ. If there is a block belonging to j and ending inside the layer at time t, there is another block belonging to j and starting at t because j is fully scheduled in ℓ and there are no overlaps. Hence, we can perform a machine swap at time t between the two machines the blocks are scheduled on. We do so for each job fully scheduled in the layer and each corresponding pair of blocks. After this step, there are at least k_ℓ usable slots and at most k_ℓ flow assigned jobs in layer ℓ.

Next, we consider upper cross-over blocks of jobs that are partially scheduled in the layer ℓ but are not flow assigned to it. These are the blocks that cause the most problems, and we perform a so-called *push and cut step* (see Fig. 4) for each of them: If q is the length of the part of the block lying in ℓ, we cut away the upper part of the block of length q and move the remainder up by q. If the piece we cut away does contain some setup time, we create a repair piece for the block out of this setup time. The processing time part of the piece, on the other hand, is removed. Note that this step preserves the first invariant. The repair piece is needed in the case that the job corresponding to the respective block is flow assigned to the layer in which the block ends.

We now remove all inner blocks from the layer as well as the parts of the upper and lower cross-over blocks that lie in the layer. After this all usable slots are completely free. Furthermore, note that the first invariant might be breached by this.

Next, we arbitrarily reserve usable slots for jobs flow assigned to the layer. For this, note that due to the definition of the flow network, there are at most k_ℓ jobs flow assigned to the layer and there are at least as many usable slots, as noted above. This step might breach the second invariant as well. Using machine swaps at the upper and lower border of the layer, we then ensure that the upper and lower cross-over blocks of the jobs flow assigned to the layer lie on the same machine as the reserved slot. Note that for each job there can be at most one upper or lower cross-over block, respectively, in the layer.

To restore the invariants, we perform the following repair steps for each job j flow assigned to the layer:

Case 1 If there is an upper cross-over block for j or a lower cross-over block without a repair peace, we reinsert the removed part (or parts) at the end or beginning of the slot, respectively. This provides a dedicated setup for the job and furthermore the first invariant once again holds for the respective cross-over blocks.

Case 2 If there is neither an upper nor a lower block for j, there is an inner block belonging to j. This has to be the case because otherwise the capacity in the flow network between j and ℓ is 0, and j could not have been flow assigned to ℓ. Moreover, this inner block contains a full setup, and we can place it in the beginning of the slot thus providing the dedicated setup. The invariants are both restored.

Case 3 The last possibility is that there is no upper cross-over block but a lower cross-over block with a repair piece. In this case, the removed part of the block is fully comprised of setup and we reinsert it in the beginning of the reserved slot. Furthermore, we insert as much setup of the repair piece as possible. If the repair piece is not used up, we now consider the remainder as the new repair piece of the block. Hence, the first invariant holds, and since the slot is full in this case, the second one holds as well. If, on the other hand, the full repair piece is inserted, we thereby provide a dedicated setup for the slot and the block once again contains a full setup. In this case, the jobs does not have a repair piece anymore.

After the first phase is finished, we have to deal with the removed pieces in the second one. The overall length of the reserved slots for a job j equals the overall length a_j of its setup and job pieces from layers in which j was partially scheduled. Since we did not create or destroy any job piece, we can place the removed pieces corresponding to job j into the remaining free space of the slots reserved for j, and we do so after transforming them completely into processing time. Because of the second invariant, there is a dedicated setup in each slot, however, it may be positioned directly above the newly inserted processing time. This can be fixed by switching the processing time with the top part of the respective setup time. Furthermore, there may be some blocks that still have a repair piece. We may remove these blocks together with their repair pieces.

Lastly, all remaining usable slots are completely free at the end of this procedure, and since the others are full, they have an overall size of at least L. We conclude the proof of Lemma 11 with an overview of the transformation procedure.

Algorithm 4 *Phase 1:* For each layer $\ell \in \Xi$, considered bottom to top, perform the following steps:

1. Use machine swaps to ensure that jobs fully scheduled in ℓ occupy exactly one slot.
2. For each upper cross-over block of a job partially scheduled but not flow assigned to ℓ perform a push and cut step.
3. Remove inner blocks and parts of cross-over blocks that lie in ℓ.

4. Reserve usable slots for jobs flow assigned to the layer.
5. Use machine swaps to ensure that cross-over blocks of flow assigned jobs lie on the same machine as the reserved slot.
6. For each job j flow assigned to the layer, perform one of the repair steps.

Phase 2:

1. Transform all removed pieces into processing time and insert the removed pieces into the reserved slots.
2. If processing time has been inserted ahead of the dedicated setup of the slot, reschedule properly.
3. Remove blocks that still have a repair piece.

5 Improvements of the running time

In this section, we revisit the splittable and the setup time model. For the former, we address the problem of the running time dependence in the number of machines m, and for both, we present an improved rounding procedure yielding a better running time.

5.1 Splittable model: machine dependence

In the splittable model, the number of machines m may be super-polynomial in the input size because it is not bounded by the number of jobs n. Hence, we need to be careful already when defining the schedule in order to get a polynomially bounded output. We say a machine is *composite* if it contains more than one job, and we say it is *plain* if it contains at most one job. For a schedule with makespan T, we call each machine *trivial* if it is plain and has load T or if it is empty and *nontrivial* otherwise. We say a schedule with makespan T is *simple* if the number of nontrivial machines is bounded by $\binom{n}{2}$.

Lemma 13 *If there is a schedule with makespan T for I there is also a simple schedule with makespan at most T.*

Proof Let there be a schedule with makespan T for I. For the first step, let us assume there are more than $\binom{n}{2}$ composite machines. In this case, there exist two distinct machines i_1 and i_2 and two distinct jobs j_1 and j_2 such that both machines contain parts of both jobs since there are at most $\binom{n}{2}$ different pairs of jobs. For $x, y \in \{1, 2\}$, let $t(x, y)$ be the processing time combined with the setup time of job $x \in \{j_1, j_2\}$ on machine $y \in \{i_1, i_2\}$. W.l.o.g., let $t(j_1, i_1)$ be the smallest value of the four. We swap this job part and its setup time with some of the processing time of the job j_2 on machine i_2. If the processing time of j_2 on i_2 is smaller than $t(j_1, i_1)$, there is no processing time of j_2 on i_2 left and we can discard the corresponding setup time. Afterwards, the makespan has not increased and at least one machine processes one job less. We can repeat this step iteratively until there are at most $\binom{n}{2}$ machines containing more than one job.

In the second step, we shift processing time from the composite machines to the plain ones. We do this for each job until it is either not contained on a composite machine or each plain machine containing this job has load T. If the job is no longer contained on a composite machine, we shift the processing time of the job such that all except one machine containing this job has load T. Since this job does not appear on any composite machine, the number of such machines can in this case be bounded by $\binom{n-1}{2}$ by repeating the first step. Therefore, the number of nontrivial machines is bounded by $\binom{n-i}{2} + i \leq \binom{n}{2}$ for some $i \in \{0, \ldots, n\}$. □

For a simple schedule, a polynomial representation of the solution is possible: For each job, we state the number of trivial machines containing this job or fix a first and last trivial machine belonging to this job. This enables a polynomial encoding length of the output, given that the remaining parts of the jobs are not fragmented into too many parts which can be guaranteed using the results of Sect. 4.

To guarantee that the MCIP finds a simple solution, we need to modify it a little. We have to ensure that nontrivial configurations are not used too often. Let $\mathcal{C}' \subseteq \mathcal{C}$ be the set of nontrivial configurations, i.e., the set of configurations containing more than one module or one module with size smaller than T. We add the following globally uniform constraint to the MCIP:

$$\sum_{C \in \mathcal{C}'} x_C \leq \binom{|\mathcal{J}^{\text{bst}}|}{2} \tag{16}$$

Since this is an inequality, we have to introduce a slack variable increasing the brick size by one. However, this does not change the running time.

The number of modules with maximum size denotes for each job in $\mathcal{J}^{\text{bst}}$ how many trivial machines it uses. The other modules can be mapped to the nontrivial configurations and the jobs can be mapped to the modules.

We still have to schedule the jobs in $\mathcal{J}^{\text{sst}}$. We do this as described in the proof of Lemma 6. We fill the nontrivial machines greedily step by step starting with the jobs having the smallest processing time. When these machines are filled, there are some completely empty machines left. Now, we estimate how many machines can be completely filled with the current job j. This can be done by dividing the remaining processing time by $T - s_i$ in $\mathcal{O}(1)$. The remaining part is scheduled on the next free machine. This machine is filled up with the next job and again the number of machines which can be filled completely with the rest of this new job is determined. These steps are iterated until all jobs in $\mathcal{J}^{\text{sst}}$ are scheduled. This greedy procedure needs at most $\mathcal{O}(|\mathcal{J}^{\text{bst}}|(|\mathcal{J}^{\text{bst}}| - 1) + |\mathcal{J}^{\text{sst}}|) = \mathcal{O}(n^2)$ operations. Therefore, we can avoid the dependence in the number of machines at the cost of a quadratic dependency in n in the running time.

5.2 Improved rounding procedures

To improve the running time in the splittable and setup class model, we reduce the number of module sizes via a geometric and an arithmetic rounding step. In both cases, the additional steps are performed following all the other simplification steps. The basic idea is to include setup times together with their corresponding job pieces or batches of jobs respectively into containers with suitably rounded sizes and to model these containers using the modules. The containers have to be at least as big as the objects they contain and the load on a machine is given by the summed up sizes of the containers on the machine. Let H^* be a set of container sizes. Then an H^*-structured schedule is a schedule in which each setup time together with its corresponding job piece or batch of jobs is packed in a container with the smallest size $h \in H^*$ such that the summed up size of the setup and the job piece or batch of jobs is upper bounded by h.

Splittable Model Consider the instance I_2 for the splittable model described in Sect. 4.2. In this instance, each setup and processing time is a multiple of $\varepsilon^2 T$ and we are interested in a schedule of length $(1+2\varepsilon)T$. For each multiple h of $\varepsilon^2 T$, let $\tilde{h} = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} h/(\varepsilon^2 T)\rceil} \varepsilon^2 T$ and $\bar{h} = \lceil \tilde{h}/\varepsilon^2 T \rceil \varepsilon^2 T$, and $\bar{H} = \{\bar{h} \mid h \in \varepsilon^2 T \mathbb{Z}_{\geq 1}, h \leq (1+2\varepsilon)^2 T\}$. Note that $|\bar{H}| \in \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$

Lemma 14 *If there is a* $((1+2\varepsilon)T, L')$*-schedule for* I_2 *in which the length of each job part is a multiple of* $\varepsilon^2 T$*, there is also an* $\bar{H}$*-structured* $((1+2\varepsilon)^2 T, L')$*-schedule for* I_2 *with the same property.*

Proof Consider such a schedule for I_2 and a pair of setup time s and job piece q scheduled on some machine. Let $h = s + q$. Stretching the schedule by $(1+2\varepsilon)$ creates enough space to place the pair into a container of size $\bar{h}$, because $(1+\varepsilon)h \leq \tilde{h}$, and $\varepsilon h \leq \varepsilon^2 T$, since $s \geq \varepsilon T$. □

To implement this lemma into the procedure, the processing time bounds $\bar{T}$ and $\check{T}$ both have to be properly increased. Modeling an $\bar{H}$-structured schedule can be done quite naturally: We simply redefine the size $\Lambda(M)$ of a module $M = (s, q) \in \mathcal{M}$ to be $\overline{(s+q)}$. With this definition, we have $|H| = |\bar{H}| = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$ yielding an improved running time for solving the MCIP of:

$$2^{\mathcal{O}(1/\varepsilon^2 (\log 1/\varepsilon)^3)} n^{1+o(1)}$$

Combining this with the results above and the considerations in Sect. 4.2 yields the running time claimed below Theorem 1.

Setup Class Model In the setup class model, an analogous approach also yields a reduced set of module sizes, that is, $|H| = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$. Therefore, the MCIP can be solved in time:

$$2^{\mathcal{O}(1/\varepsilon^3 (\log 1/\varepsilon)^4)} K^{1+o(1)}$$

Hence, we get the running time claimed beneath Theorem 1.

6 Conclusion

We presented a more advanced version of the classical configuration IP, showed that it can be solved efficiently using algorithms for n-fold IPs, and developed techniques to employ the new IP for the formulation of efficient polynomial time approximation schemes for three scheduling problems with setup times for which no such algorithms were known before.

For further research the immediate questions are whether improved running times for the considered problems, in particular for the preemptive model, can be achieved; whether the MCIP can be solved more efficiently; and to which other problems it can be reasonably employed. From a broader perspective, it would be interesting to further study the potential of new algorithmic approaches in integer programming for approximation, and, on the other hand, further study the respective techniques themselves.

Acknowledgements We thank Syamantak Das for helpful discussions on the problem.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Allahverdi, A.: The third comprehensive survey on scheduling problems with setup times/costs. Eur. J. Oper. Res. **246**(2), 345–378 (2015). https://doi.org/10.1016/j.ejor.2015.04.004
2. Allahverdi, A., Gupta, J.N., Aldowaisan, T.: A review of scheduling research involving setup considerations. Omega **27**(2), 219–239 (1999)
3. Allahverdi, A., Ng, C., Cheng, T.E., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. Eur. J. Oper. Res. **187**(3), 985–1032 (2008)
4. Alon, N., Azar, Y., Woeginger, G.J., Yadid, T.: Approximation schemes for scheduling on parallel machines. J. Sched. **1**(1), 55–66 (1998)
5. Chen, B.: A better heuristic for preemptive parallel machine scheduling with batch setup times. SIAM J. Comput. **22**(6), 1303–1318 (1993)
6. Chen, B., Ye, Y., Zhang, J.: Lot-sizing scheduling with batch setup times. J. Sched. **9**(3), 299–310 (2006)
7. Chen, L., Marx, D., Ye, D., Zhang, G.: Parameterized and approximation results for scheduling with a low rank processing time matrix. In: LIPIcs-Leibniz International Proceedings in Informatics, vol. 66. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
8. Correa, J., Marchetti-Spaccamela, A., Matuschke, J., Stougie, L., Svensson, O., Verdugo, V., Verschae, J.: Strong LP formulations for scheduling splittable jobs on unrelated machines. Math. Program. **154**(1–2), 305–328 (2015)
9. Correa, J., Verdugo, V., Verschae, J.: Splitting versus setup trade-offs for scheduling to minimize weighted completion time. Oper. Res. Lett. **44**(4), 469–473 (2016)
10. Cslovjecsek, J., Eisenbrand, F., Hunkenschröder, C., Rohwedder, L., Weismantel, R.: Block-structured integer and linear programming in strongly polynomial and near linear time. In: D. Marx (ed.) Proceed-

ings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021, pp. 1666–1681. SIAM (2021)
11. Eisenbrand, F., Hunkenschröder, C., Klein, K.: Faster algorithms for integer programs with block structure. In: 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9–13, 2018, Prague, Czech Republic, pp. 49:1–49:13 (2018)
12. Eisenbrand, F., Hunkenschröder, C., Klein, K., Koutecký, M., Levin, A., Onn, S.: An algorithmic theory of integer programming. CoRR (2019). http://arxiv.org/abs/abs/1904.01361
13. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting-stock problem. Oper. Res. **9**(6), 849–859 (1961)
14. Goemans, M.X., Rothvoss, T.: Polynomiality for bin packing with a constant number of item types. J. ACM **67**(6), 38:1–38:21 (2020)
15. Hemmecke, R., Onn, S., Romanchuk, L.: N-fold integer programming in cubic time. Math. Program. **137**, 1–17 (2013)
16. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. J. ACM (JACM) **34**(1), 144–162 (1987)
17. Jansen, K., Klein, K., Maack, M., Rau, M.: Empowering the configuration-ip—new PTAS results for scheduling with setups times. CoRR (2018). http://arxiv.org/abs/abs/1801.06460
18. Jansen, K., Klein, K., Maack, M., Rau, M.: Empowering the configuration-ip—new PTAS results for scheduling with setups times. In: 10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10–12, 2019, San Diego, California, USA, pp. 44:1–44:19 (2019)
19. Jansen, K., Klein, K., Verschae, J.: Closing the gap for makespan scheduling via sparsification techniques. Math. Oper. Res. **45**(4), 1371–1392 (2020)
20. Jansen, K., Land, F.: Non-preemptive scheduling with setup times: A ptas. In: European Conference on Parallel Processing, pp. 159–170. Springer, Berlin (2016)
21. Jansen, K., Lassota, A., Rohwedder, L.: Near-linear time algorithm for n-fold ilps via color coding. SIAM J. Discret. Math. **34**(4), 2282–2299 (2020)
22. Kannan, R.: Minkowski's convex body theorem and integer programming. Math. Oper. Res. **12**(3), 415–440 (1987)
23. Knop, D., Koutecký, M.: Scheduling meets n-fold integer programming. J. Sched. **21**, 1–11 (2017)
24. Knop, D., Koutecký, M., Levin, A., Mnich, M., Onn, S.: Multitype integer monoid optimization and applications. CoRR **abs/1909.07326** (2019)
25. Knop, D., Koutecký, M., Mnich, M.: Combinatorial n-fold integer programming and applications. Math. Program. **184**(1), 1–34 (2020)
26. Koutecký, M., Levin, A., Onn, S.: A parameterized strongly polynomial algorithm for block structured integer programs. In: 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9–13, 2018, Prague, Czech Republic, pp. 85:1–85:14 (2018)
27. Lenstra Jr., H.W.: Integer programming with a fixed number of variables. Math. Oper. Res. **8**(4), 538–548 (1983)
28. Loera, J.A.D., Hemmecke, R., Köppe, M.: Algebraic and Geometric Ideas in the Theory of Discrete Optimization. MOS-SIAM Series on Optimization, vol. 14. SIAM, Philadelphia (2013)
29. Mäcker, A., Malatyali, M., auf der Heide, F.M., Riechers, S.: Non-preemptive scheduling on machines with setup times. In: Workshop on Algorithms and Data Structures, pp. 542–553. Springer, Berlin (2015)
30. Monma, C.L., Potts, C.N.: Analysis of heuristics for preemptive parallel machine scheduling with batch setup times. Oper. Res. **41**(5), 981–993 (1993)
31. Onn, S.: Nonlinear Discrete Optimization. Zurich Lectures in Advanced Mathematics. European Mathematical Society, Zurich (2010)
32. Schalekamp, F., Sitters, R., Van Der Ster, S., Stougie, L., Verdugo, V., Van Zuylen, A.: Split scheduling with uniform setup times. J. Sched. **18**(2), 119–129 (2015)
33. Schuurman, P., Woeginger, G.J.: Preemptive scheduling with job-dependent setup times. In: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms, pp. 759–767. Society for Industrial and Applied Mathematics, Philadelphia (1999)