# Empowering the Configuration-IP – New PTAS Results for Scheduling with Setups Times

**Klaus Jansen**[1]
Department of Computer Science, Kiel University, Kiel, Germany
kj@informatik.uni-kiel.de

**Kim-Manuel Klein**
Department of Computer Science, Kiel University, Kiel, Germany
kmk@informatik.uni-kiel.de

**Marten Maack**
Department of Computer Science, Kiel University, Kiel, Germany
mmaa@informatik.uni-kiel.de

**Malin Rau**
Department of Computer Science, Kiel University, Kiel, Germany
mra@informatik.uni-kiel.de

## 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 chose 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 $f(1/\varepsilon) \times \mathrm{poly}(|I|)$ with a single exponential term in $f$ for the first and a double exponential one for the second case. Previously, only constant factor approximations of $5/3$ and $4/3 + \varepsilon$ respectively were known. Furthermore, we present an EPTAS for 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.

**2012 ACM Subject Classification** Theory of computation → Scheduling algorithms, Theory of computation → Discrete optimization

**Keywords and phrases** Parallel Machines, Setup Time, EPTAS, $n$-fold integer programming

**Digital Object Identifier** 10.4230/LIPIcs.ITCS.2019.44

**Related Version** The long version is hosted on arXiv [14], `https://arxiv.org/abs/1801.06460`.

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

[1] German Research Foundation (DFG) project JA 612/20-1

## 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 [10], and the recent results for both problems typically use configuration IPs as a core technique, see, e.g., [11, 15]. 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 $\sigma : \mathcal{J} \to [m]$, 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, where 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., [3, 15]). Hence, only a constant number of configurations is needed, yielding 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 [21, 17], 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 $f(1/\varepsilon) \times \mathrm{poly}(|I|)$, 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 there is no optimal polynomial time algorithm, unless P=NP, and also a so-called *fully polynomial* PTAS (FPTAS) – which is an EPTAS with a polynomial function $f$ – cannot be hoped for.

**Machine Scheduling with Classes.** The configuration IP is used in a wide variety of approximation schemes for machine scheduling problems [3, 15]. However, the approach often ceases to work for scheduling problems in which the jobs have to fulfill some additional requirements, like, for instance, class dependencies. 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 bounded in number and guaranteed to be sufficiently big [16]. However, it is not hard to see that the configuration IP still cannot be trivially extended, while preserving its solvability. For instance, extending the configurations with multiplicities of setup times will not work, because then we have to make sure that a configuration is used for a fitting subset of classes, creating the need to encode class information into the configurations or introduce other class dependent variables.

**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 configurations of processing times, and the actual configurations 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 [21, 17] 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 – cover the jobs of the class. Utilizing these properties, we can formulate the MCIP in the framework of $n$-fold integer programms – a class of IPs whose variables and constraints fulfill certain uniformity requirements. In 2013 Hemmecke, Onn, and Romanchuk [12] showed that $n$-fold IPs can be efficiently solved, and very recently both Eisenbrand, Hunkenschröder and Klein [9] and independently Koutecký, Levin and Onn [20] developed algorithms with greatly improved running times for the problem. For a detailed description of the MCIP, the reader is referred to Section 3. In Figure 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 [16]. 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 beside the partition of the jobs, it suffices to find 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

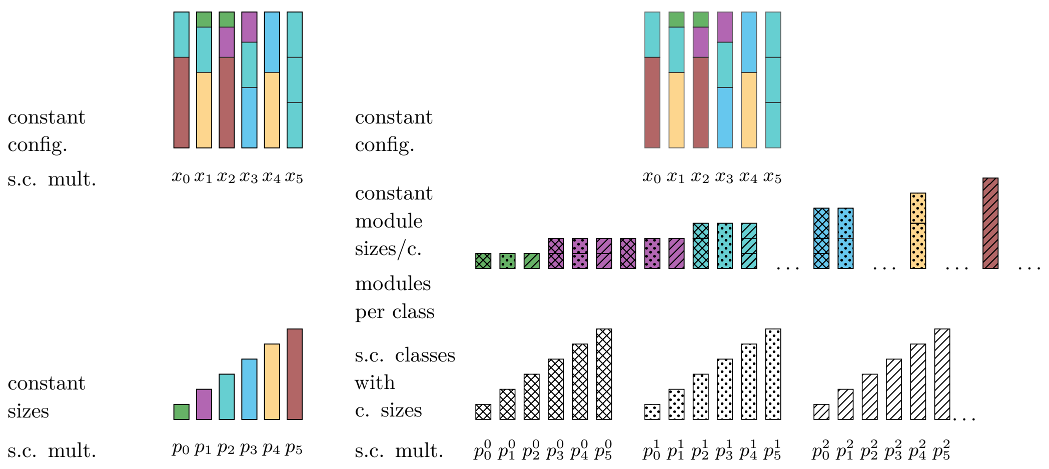
constant config.
s.c. mult.
$x_0\ x_1\ x_2\ x_3\ x_4\ x_5$
constant sizes
s.c. mult.
$p_0\ p_1\ p_2\ p_3\ p_4\ p_5$
constant config.
$x_0\ x_1\ x_2\ x_3\ x_4\ x_5$
constant module sizes/c.
modules per class
s.c. classes with c. sizes
s.c. mult.
$p_0^0\ p_1^0\ p_2^0\ p_3^0\ p_4^0\ p_5^0$
$p_0^1\ p_1^1\ p_2^1\ p_3^1\ p_4^1\ p_5^1$
$p_0^2\ p_1^2\ p_2^2\ p_3^2\ p_4^2\ p_5^2$

**Figure 1** On the left, there is a schematic representation of the configuration IP. There are constant 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.

parts of the same job may not be processed in parallel. In 1999 Schuurman and Woeginger [26] 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, Ye and Zhang [5]. 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 $2^{\mathcal{O}(1/\varepsilon^3 \log^4 1/\varepsilon)}K^2nm\log(Km)$ in the setup class model, $2^{\mathcal{O}(1/\varepsilon^2 \log^3 1/\varepsilon)}n^2\log^3(nm)$ in the splittable, and $2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}}n^2m\log m\log(nm)$ in the preemptive model (remember that $n$, $m$, and $K$ denote the number of jobs, machines, and setup classes). Note that all three problems are strongly NP-hard, due to trivial reductions from machine scheduling, and our results are therefore in some sense best possible.

Summing up, the main achievement of this work is the development of the module configuration IP and its application in the development 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 elaborate, and therefore interesting by themselfs. 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 depended constraints. On a more conceptual level, we gave 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 [13] to get a guess of the makespan $T$. This approach is introduced in Section 2 together with $n$-fold IPs and formal definitions of the problems. In the following section, we develop the module configuration IP, in its basic form and argue that it is indeed an $n$-fold IP. The EPTAS results described in the last section 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. Missing proofs and details can be found in the long version of the paper [14].

**Related work.** For an overview on $n$-fold IPs and their applications, we refer to the book by Onn [24]. There have been recent applications of $n$-fold integer programming to scheduling problems in the context of parameterized algorithms: Knop and Koutecký [18] 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. [6]. Furthermore, Knop, Koutecký and Mnich [19] 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.

There is extensive literature concerning scheduling problems with setup times. We highlight a few closely related results and otherwise refer to the surveys [1, 2]. In the following, we use the term $\alpha$-approximation as an abbreviation for polynomial time algorithms with approximation guarantee $\alpha$. The setup class model was first considered by Mäcker et al. [22] 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 [16] 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. [5] 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. [7]. They achieve a $(1 + \phi)$-approximation, where $\phi$ 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 the (weighted) completion times as the objective function, e.g. [25, 8]. 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 [26]. The mentioned $(4/3 + \varepsilon)$-approximation for the general case [26] 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 [23] presented, among other things, a $(2 - 1/(\lfloor m/2 \rfloor + 1))$-approximation for this model, and later Chen [4] 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 [13], 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 models, 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 $\mathcal{J}' = \{(j,k) \,|\, j \in \mathcal{J}, k \in [\kappa(j)]\}$ 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 $\xi(j,k) + s_j + \lambda_j(k)p_j \le \xi(j')$ or $\xi(j',k') + s_{j'} + \lambda_{j'}(k)p_{j'} \le \xi(j)$. 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 [26] 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 $\sigma$, 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 given an assignment, 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 $C_{\max} = \max_{i\in[m]} \sum_{j\in\sigma^{-1}(i)} p_j + \sum_{k\in\{k_j \,|\, j\in\sigma-1(i)\}} s_k$.

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.

**Dual Approximation.** All of the presented algorithms follow the dual approximation framework introduced by Hochbaum and Shmoys [13]: 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 $B \le b\mathrm{OPT}$ 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 $T^* \le (1 + \mathcal{O}(\varepsilon))\mathrm{OPT}$ after $\mathcal{O}(\log b/\varepsilon)$ iterations. Note that for all of the considered problems there are known simple approximation algorithms. 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 (IP) following the notation of [12] and [18] 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 $((r + ns) \times nt)$-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:

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

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*. We set $\Delta$ to be the maximum absolute value occurring in $A$. Up to recently the best known algorithm for solving $n$-fold IPs was due to Hemmecke, Onn and Romanchuk [12]:

▶ **Theorem 2.** *Let $\varphi$ be the encoding length of $w$, $b$, $\ell$, $u$ and $\Delta$. The $n$-fold integer programming problem can be solved in time $\mathcal{O}(\Delta^{3t(rs+st+r+s)}n^3\varphi)$, when $r$, $s$ and $t$ are fixed.*

However, in 2018 both Eisenbrand, Hunkenschröder and Klein [20] and independently Koutecký, Levin and Onn [20] developed algorithms with improved and very similar running times. We state a variant due to Eisenbrand et al. that is adapted to our needs:

▶ **Theorem 3.** *Let $\varphi$ be the encoding length of the largest number occurring in the input, and $\Phi = \max_i(u_i - \ell_i)$. The $n$-fold integer programming problem can be solved in time $(rs\Delta)^{\mathcal{O}(r^2s+rs^2)}t^2n^2\varphi\log(\Phi)\log(nt\Phi)$.*

## 3 Module Configuration IP

In this section, we state the configuration IP for machine scheduling and 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. Before that, however, we formally introduce the concept of *configurations*. Given a set of objects $A$, a configuration $C$ of these objects is a vector of multiplicities indexed by the objects, i.e., $C \in \mathbb{Z}^A_{\geq 0}$. 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 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, $\mathcal{C}_A(B) = \{C \in \mathbb{Z}^A_{\geq 0} \mid \Lambda(C) \leq B\}$.

**Configuration IP.** We give a recollection of the configuration IP for scheduling on identical parallel machines. Let $P$ be the set of 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 $p$ ($\Lambda(p) = \text{p}$ in this context). 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 $B_1, \dots, B_D$ 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* $\mathcal{M}$ 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. Corresponding to the basic objects, each module $M \in \mathcal{M}$ also has $D$ integer values $M_1, \dots, M_D$, as well as 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. 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 chose configurations for each machine, we have variables $x_C$ for each $C \in \mathcal{C}$ and constraints corresponding to (1). Furthermore, we chose 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 $h \in H$. It turns out however, that to properly cover the basic objects with modules, we need 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 the following constraints.

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

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

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

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 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.

## 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 Section 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, 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

In the setup class model, simplification steps similar to the ones developed by Jansen and Land [16] can be used. We give a brief overview:

The set of big setup jobs $\mathcal{J}^{\text{bst}}$ is given by the jobs belonging to classes with setup times at least $\varepsilon^3 T$ and the small setup jobs $\mathcal{J}^{\text{sst}}$ are all the others. Furthermore, we call a job *tiny* or *small*, if its processing time is smaller than $\varepsilon^4 T$ or $\varepsilon T$ respectively, and *big* or *large* otherwise. For any given set of jobs $J$, we denote the subset of tiny jobs from $J$ with $J_{\text{tiny}}$ and the small, big and large jobs analogously. 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: We remove the small jobs with small setup times $\mathcal{J}^{\text{sst}}_{\text{small}}$; increase the setup times of the remaining classes with small setup times to $\varepsilon^3 T$; replace the tiny jobs from $\mathcal{J}^{\text{bst}}_{\text{tiny}}$ with placeholders of size $\varepsilon^4 T$; and round up the resulting processing and setup times in a geometric and a subsequent arithmetic rounding step. The above steps yield certain makespan bounds $\bar{T} \leq \breve{T} = (1 + \mathcal{O}(\varepsilon))T$, and for the resulting instance $I'$, the sets $P$ and $S$ of distinct processing and setup times have bounded cardinality, that is, $|P|, |S| \in \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$, and each processing and setup time is a multiple of $\varepsilon^5 T$. 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 $L = \sum_{j \in \mathcal{J}^{\text{sst}}_{\text{small}}} p_j + \sum_{k \in Q} s_k$.

▶ **Theorem 4.** *If there is a schedule for $I$ with makespan $T$, there is a $(\bar{T}, L)$-schedule for $I'$; and if there is a $(\bar{T}, L)$-schedule for $I'$, there is a schedule for $I$ with makespan $\breve{T}$.* ◀

**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., $\mathcal{M} = \{(C, s) \mid C \in \mathcal{C}_P(\bar{T}), s \in S, s + \Lambda(C) \leq \bar{T}\}$, 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 $I'$, iff the* MCIP *has a solution with objective value at most $m\bar{T} - L$.*

**Proof.** Let there be a $(\bar{T}, L)$-schedule for $I'$. Then the schedule on a given machine corresponds to a distinct configuration $C$ that can be determined by counting for each possible group size $a$ the batches of jobs from the same class whose length together with the setup time adds up to an overall length of $a$. 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 $L$ free space in the schedule. For each class $k$ and module $M$, we count the number of machines on which the 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 $C$, we create $C_G$ slots of length $\Lambda(G)$ on $i$ for each group $G$. 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.** To analyze the running time of the resulting procedure, we mainly have to 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 $|H| \in \mathcal{O}(1/\varepsilon^5)$, 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 $K$, $\mathcal{O}(1/\varepsilon^5)$ globally, and $\mathcal{O}(1/\varepsilon \log 1/\varepsilon)$ 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. Furthermore, the values of the objective function, the right hand side, and the upper and lower bounds on the variables are bounded by $\mathcal{O}(n/\varepsilon^5)$, yielding a bound of $\mathcal{O}(\log n/\varepsilon^5)$ for the encoding length of the biggest number in the input $\varphi$. Lastly, all variables can be bounded by 0 from below and $\mathcal{O}(m/\varepsilon^3)$ from above, yielding $\Phi = \mathcal{O}(m/\varepsilon^3)$.

By Theorem 3 and some arithmetic, the MCIP can be solved in time:

$$(rs\Delta)^{\mathcal{O}(r^2 s + rs^2)} t^2 n^2 \varphi \log(\Phi) \log(nt\Phi) = 2^{\mathcal{O}(1/\varepsilon^{11} \log^2 1/\varepsilon)} K^2 \log n \log m \log Km$$

When building the actual schedule, we iterate through the jobs and machines, yielding an additional term that is linear in $n$ and $m$. To get the running time stated below Theorem 1, additional rounding steps for the module sizes and slightly altered modules are needed. For details we refer to the long version of the paper.

## 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 the long version, we show how this problem can be overcome and further improve the running time.

For the simplification of the instance, similar ideas to the setup class case can be employed. We give a brief overview:

In this context, the set of big setup jobs $\mathcal{J}^{\text{bst}}$ is given by the jobs with setup times at least $\varepsilon T$ 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. The instance is simplified in two steps: First, all the small setup jobs are removed, and next the remaining processing and setup times are rounded up to multiples of $\varepsilon^2 T$. These steps yield certain makespan bounds $\bar{T} \leq \breve{T} = (1 + \mathcal{O}(\varepsilon))T$, and for the resulting instance $I'$, the sets $P$ and $S$ of distinct processing and setup times have bounded cardinality, that is, $|P|, |S| \in \mathcal{O}(1/\varepsilon^2)$.

▶ **Theorem 6.** *If there is a schedule with makespan $T$ for $I$, there is also $(\bar{T}, L)$-schedule for $I'$ in which the length of each job part is a multiple of $\varepsilon^2 T$; and if there is a $(\bar{T}, L)$-schedule for $I'$, there is also a schedule for $I$ with makespan at most $\breve{T}$.*

We will use the MCIP to find a $(\bar{T}, L)$-schedule for $I'$ in which the length of each job part is a multiple of $\varepsilon^2 T$.

**Utilization of the MCIP.** The basic objects in this context are the (big setup) jobs, i.e., $\mathcal{B} = \mathcal{J}^{\text{bst}}$, 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., $\mathcal{M} = \{(q, s) \mid s, q \in S \times P\}$, 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., $\mathcal{B}_M = \{j \in \mathcal{J}' \mid s_j = s_M\}$. In order to ensure integral values, we establish $\varepsilon^2 T = 1$ via scaling. 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., $\mathcal{C} = \mathcal{C}_{\mathcal{M}}(\bar{T})$. We state the constraints of the MCIP for the above definitions with adapted notation and without duplication of the configuration variables:

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

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

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

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

▶ **Lemma 7.** *With the above definitions, there is a $(\bar{T}, L)$-schedule for $I'$ in which the length of each job piece is a multiple of $\varepsilon^2 T$, iff* MCIP *has a solution with objective value at most $m\bar{T} - L$.*

**Proof.** Given such a schedule for $I'$, the schedule on each machine corresponds to exactly one configuration $G$ that can be derived by counting the job pieces and setup times with the same summed up length $a$ and setting $C_G$ accordingly, where $G$ is the group of modules with length $a$. The size of the configuration $C$ is equal to the used space on the respective machine. Therefore, we can fix some arbitrary job $j$ and set the variables $x_C^{(j)}$ to the number of machines whose schedule corresponds to $C$ (and $x_C^{(j')} = 0$ for $j' \neq j$ and $C \in \mathcal{C}$). Since there is at least a free space of $L$ for the schedule, 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 $y^{(j)}_{(q,s_j)}$ 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 job $j$ and configuration $C$, we reserve $x^{(j)}_C$ 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 $y^{(j)}_{(q,s_j)}$ 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.** 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 $|\mathcal{B}| \leq n$ 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 $|\mathcal{C}| \leq |H|^{\mathcal{O}(1/\varepsilon+2)} = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$, because the size of each module is at least $\varepsilon T$ 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 $|\mathcal{B}| = n$, $r = |\Gamma| + 1 = \mathcal{O}(1/\varepsilon^2)$ 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 $1/\varepsilon^2$ and therefore $\Delta \leq 1/\varepsilon^2$. Furthermore, each occurring number can be bounded by $\mathcal{O}(m/\varepsilon^2)$ and this is an upper bound for each variable as well, yielding $\varphi = \mathcal{O}(\log m/\varepsilon^2)$ and $\Phi = \mathcal{O}(m/\varepsilon^2)$. Hence the MCIP, can be solved in time:

$$(rs\Delta)^{\mathcal{O}(r^2 s + rs^2)} t^2 n^2 \varphi \log(\Phi) \log(nt\Phi) = 2^{\mathcal{O}(1/\varepsilon^4 \log 1/\varepsilon)} n^2 \log^2 m \log nm$$

When building the actual schedule, we iterate through the jobs and machines, yielding an additional term that is linear in $n$ and $m$. To get the running time stated below Theorem 1, some additional considerations are needed. For details, we refer to the long version of the paper.

## 4.3 Preemptive Model

In the preemptive model, we have to actually consider the time-line 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 $\Xi$ is given by $\{\ell \in \mathbb{Z}_{>0} \,|\, (\ell - 1)X \leq T\}$. 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, for $X$-layered schedules, we slightly alter the definition of free space: 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.*

**Simplification of the Instance.** In the preemptive model, we distinguish *big*, *medium* and *small* setup jobs, using two parameters $\delta$ and $\mu$: The big setup jobs $\mathcal{J}^{\mathrm{bst}}$ are those with setup time at least $\delta T$, the small $\mathcal{J}^{\mathrm{sst}}$ have a setup time smaller than $\mu T$, and the medium $\mathcal{J}^{\mathrm{mst}}$ are the ones in between. We set $\mu = \varepsilon^2\delta$ and we choose $\delta \in \{\varepsilon^1, \dots, \varepsilon^{2/\varepsilon^2}\}$ 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., $\sum_{j\in\mathcal{J}^{\mathrm{mst}}}(s_j + p_j) \leq m\varepsilon T$. 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 $\varepsilon T$ *big* and the others *small.* For a given set of jobs $J$, we call the subsets of big or small jobs $J_{\mathrm{big}}$ or $J_{\mathrm{small}}$ respectively. 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.

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

► **Lemma 8.** *If there is a schedule with makespan at most $T$ for $I$, there is also such a schedule for $I_1$, and if there is a schedule with makespan at most $T'$ for $I_1$ 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 $\mathcal{J}^{\mathrm{mst}}_{\mathrm{small}}$ 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 $\varepsilon T$. 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 $\delta T$ 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 $\mathcal{J}^{\mathrm{sst}}_{\mathrm{small}}$ and setting the setup time of the big jobs with small setup times to zero, i.e., $\bar{s}_j = 0$ for each $j \in \mathcal{J}^{\mathrm{sst}}_{\mathrm{big}}$. Note that in the resulting instance each small job has a big setup time. Furthermore, let $L := \sum_{j\in\mathcal{J}^{\mathrm{sst}}_{\mathrm{small}}} p_j + s_j$. Then $L$ is an obvious lower bound for the space taken up by the jobs from $\mathcal{J}^{\mathrm{sst}}_{\mathrm{small}}$ in any schedule.

► **Lemma 9.** *If there is a schedule with makespan at most $T$ for $I_1$, there is also a $(T, L)$-schedule for $I_2$; and if there is a $\gamma T$-layered $(T', L)$-schedule for $I_2$, with $T'$ a multiple of $\gamma T$, 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 $\gamma T$-layered $(T', L)$-schedule for $I_2$. We create a sequence that contains the jobs of $\mathcal{J}^{\mathrm{sst}}_{\mathrm{small}}$ 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 and 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: If its a setup we discard it and if its a job, 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. The resulting schedule is increased by at most $\varepsilon T$, 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 $\mu T$ at each multiple of $\gamma T$ 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 $\mu T$ 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 $j \in \mathcal{J}^{\mathrm{bst}}$, and $\bar{s}_j = \lceil s_j/(\varepsilon\mu T)\rceil \varepsilon\mu T$ for each medium setup job $j \in \mathcal{J}^{\mathrm{mst}}_{\mathrm{big}}$.

▶ **Lemma 10.** *If there is a $(T, L)$-schedule for $I_2$, there is also an $\varepsilon\delta T$-layered, layer-compliant $((1 + 3\varepsilon)T, L)$-schedule for $I_3$; and if there is a $\gamma T$-layered $(T', L)$-schedule for $I_3$, 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 can be found in the long version of the paper.

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 $\breve{T}$: Let $\bar{T}$ be the makespan bound we get by the application of the Lemmata 8-10, i.e., $\bar{T} = (1 + \mathcal{O}(\varepsilon))T$. We will use the MCIP to find an $\varepsilon\delta T$-layered $(\bar{T}, L)$-schedule for $I_3$, and apply the Lemmata 8-10 backwards to get schedule for $I$ with makespan at most $\breve{T} = (1 + \mathcal{O}(\varepsilon))\bar{T} = (1 + \mathcal{O}(\varepsilon))T$.

**Utilization of the MCIP.** Similar to the splittable case, the basic objects are the (big) jobs, i.e., $\mathcal{B} = \mathcal{J}_{\mathrm{big}}$, 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 $(\ell, q, s, b)$ 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 $\mathcal{M}^{\mathrm{bst}}$, $\mathcal{M}^{\mathrm{mst}}$ and $\mathcal{M}^{\mathrm{sst}}$ that are defined in the following. For this let $Q^{\mathrm{bst}} = \{q \mid q = x\varepsilon\delta T, x \in \mathbb{Z}_{>0}, q \leq \bar{T}\}$ and $Q^{\mathrm{mst}} = \{q \mid q = x\varepsilon\mu T, x \in \mathbb{Z}_{>0}, q \leq \bar{T}\}$ be the sets of possible job piece sizes of big and medium setup jobs; $S^{\mathrm{bst}} = \{s \mid s = x\varepsilon\delta T, x \in \mathbb{Z}_{\geq 1/\varepsilon}, s \leq \bar{T}\}$ and $S^{\mathrm{mst}} = \{s \mid s = x\varepsilon\mu T, x \in \mathbb{Z}_{\geq 1/\varepsilon}, s \leq \delta T\}$ 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 $\Xi = \{1, \ldots, 1/(\varepsilon\delta) + 3/\delta\}$ the set of layers. We set:

$$\begin{aligned}
\mathcal{M}^{\mathrm{bst}} &= \{(\ell, q, s, 0) \mid \ell \in \Xi, q \in Q^{\mathrm{bst}}, s \in S^{\mathrm{bst}}, (\ell-1)\varepsilon\delta T + s + q \leq \bar{T}\} \\
\mathcal{M}^{\mathrm{mst}} &= \{(\ell, q, s, b) \in \Xi \times Q^{\mathrm{mst}} \times S^{\mathrm{mst}} \times B \mid x = s + q + b \in \varepsilon\delta T\mathbb{Z}_{>0}, (\ell-1)\varepsilon\delta T + x \leq \bar{T}\} \\
\mathcal{M}^{\mathrm{sst}} &= \{(\ell, \varepsilon\delta T, 0, 0) \mid \ell \in \Xi\}
\end{aligned}$$

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 is given, if there is some layer $\ell$ 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 $\ell$. Furthermore, we partition the modules into groups $\Gamma$ by size and starting layer, i.e., $\Gamma = \{G \subseteq \mathcal{M} \mid M, M' \in G \Rightarrow \Lambda(M) = \Lambda(M') \wedge \ell_M = \ell_{M'}\}$. 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., $\mathcal{C} = \{C \in \mathbb{Z}^{\Gamma}_{\geq 0} \mid \forall \ell \in \Xi : \sum_{G \subseteq \mathcal{M}_\ell} C_G \leq 1\}$. 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 11.** *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, iff 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 the $j$ occurs in $\ell$, 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 $\ell$ 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 $\ell$ 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 $\ell$ be a layer. If $j$ has a small setup time, we create $y_{(\ell,\varepsilon\delta T,0,0)}^{(j)}$ pieces of length $\varepsilon\delta T$ and place these pieces into allocated spaces of length $\varepsilon\delta T$ in layer $\ell$. 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 $q \in Q^{\text{mst}}$, 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 $\ell$. 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.** 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. Furthermore, the numbers in the input can be bounded by $m2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ and all variables can be upper bounded by $\mathcal{O}(m)$. Hence, we have $\varphi = \mathcal{O}(\log m + 1/\varepsilon \log 1/\varepsilon)$ and $\Phi = \mathcal{O}(m)$, and due to Theorem 3 we can solve the MCIP in time:

$$(rs\Delta)^{\mathcal{O}(r^2 s + rs^2)} t^2 n^2 \varphi \log(\Phi) \log(nt\Phi) = 2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}} n^2 \log^2 m \log nm$$

When building the actual schedule we iterate through the jobs and machines yielding the slightly increased running time stated below Theorem 1.

## 5 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 themselfs.

## References

1 Ali Allahverdi, Jatinder ND Gupta, and Tariq Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239, 1999.

2 Ali Allahverdi, CT Ng, TC Edwin Cheng, and Mikhail Y Kovalyov. A survey of scheduling problems with setup times or costs. *European journal of operational research*, 187(3):985–1032, 2008.

3 Noga Alon, Yossi Azar, Gerhard J Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.

4 Bo Chen. A better heuristic for preemptive parallel machine scheduling with batch setup times. *SIAM Journal on Computing*, 22(6):1303–1318, 1993.

5 Bo Chen, Yinyu Ye, and Jiawei Zhang. Lot-sizing scheduling with batch setup times. *Journal of Scheduling*, 9(3):299–310, 2006.

6 Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 66. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

7 José Correa, Alberto Marchetti-Spaccamela, Jannik Matuschke, Leen Stougie, Ola Svensson, Víctor Verdugo, and José Verschae. Strong LP formulations for scheduling splittable jobs on unrelated machines. *Mathematical Programming*, 154(1-2):305–328, 2015.

8 José Correa, Victor Verdugo, and José Verschae. Splitting versus setup trade-offs for scheduling to minimize weighted completion time. *Operations Research Letters*, 44(4):469–473, 2016.

9 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. 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*, pages 49:1–49:13, 2018.

**10** Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

**11** Michel X Goemans and Thomas Rothvoß. Polynomiality for bin packing with a constant number of item types. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 830–839. SIAM, 2014.

**12** Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. N-fold integer programming in cubic time. *Mathematical Programming*, pages 1–17, 2013.

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

**14** Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the Configuration-IP – New PTAS Results for Scheduling with Setups Times. *CoRR*, abs/1801.06460v2, 2018. `arXiv:1801.06460`.

**15** Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the Gap for Makespan Scheduling via Sparsification Techniques. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 72:1–72:13, 2016.

**16** Klaus Jansen and Felix Land. Non-preemptive Scheduling with Setup Times: A PTAS. In *European Conference on Parallel Processing*, pages 159–170. Springer, 2016.

**17** Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, 1987.

**18** Dusan Knop and Martin Koutecký. Scheduling meets n-fold Integer Programming. *Journal of Scheduling*, pages 1–11, 2017.

**19** Dusan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n-fold Integer Programming and Applications. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 54:1–54:14, 2017.

**20** Martin Koutecký, Asaf Levin, and Shmuel Onn. 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*, pages 85:1–85:14, 2018.

**21** Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of operations research*, 8(4):538–548, 1983.

**22** Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. Non-preemptive scheduling on machines with setup times. In *Workshop on Algorithms and Data Structures*, pages 542–553. Springer, 2015.

**23** Clyde L Monma and Chris N Potts. Analysis of heuristics for preemptive parallel machine scheduling with batch setup times. *Operations Research*, 41(5):981–993, 1993.

**24** Shmuel Onn. Nonlinear discrete optimization. *Zurich Lectures in Advanced Mathematics, European Mathematical Society*, 2010.

**25** Frans Schalekamp, René Sitters, Suzanne Van Der Ster, Leen Stougie, Víctor Verdugo, and Anke Van Zuylen. Split scheduling with uniform setup times. *Journal of scheduling*, 18(2):119–129, 2015.

**26** Petra Schuurman and Gerhard J Woeginger. Preemptive scheduling with job-dependent setup times. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 759–767. Society for Industrial and Applied Mathematics, 1999.