Scheduling Splittable Jobs on Configurable Machines

Matthew Casey
Northeastern University, Boston MA 02115, USA

Rajmohan Rajaraman
Northeastern University, Boston MA 02115, USA

David Stalfa
Northeastern University, Boston MA 02115, USA

Cheng Tan
Northeastern University, Boston MA 02115, USA

Abstract

Motivated by modern architectures allowing for the partitioning of a GPU into hardware separated instances, we initiate the study of scheduling splittable jobs on configurable machines. We consider machines that can be configured into smaller instances, which we call blocks, in multiple ways, each of which is referred to as a configuration. We introduce the Configurable Machine Scheduling (CMS) problem, where we are given n jobs and a set C of configurations. A schedule consists of a set of machines, each assigned some configuration in C with each block in the configuration assigned to process one job. The amount of a job's demand that is satisfied by a block is given by an arbitrary function of the job and block. The objective is to construct a schedule using as few machines as possible. We provide a tight logarithmic factor approximation algorithm for this problem in the general setting, a factor $(3 + \varepsilon)$ approximation algorithm for arbitrary $\varepsilon > 0$ when there are $O(1)$ input configurations, and a polynomial time approximation scheme when both the number and size of configurations are $O(1)$. Finally, we utilize a technique for finding conic integer combinations in fixed dimension to develop an optimal polynomial time algorithm in the case with $O(1)$ jobs, $O(1)$ blocks, and every configuration up to a given size.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling algorithms, Approximation algorithms, Configurable machines, Splittable jobs, Linear programming

Digital Object Identifier 10.4230/LIPIcs...22

Category APPROX

Related Version *Previous Version*: `https://doi.org/10.48550/arXiv.2312.05416`

Funding *Matthew Casey*: Partially supported by NSF grant CCF-1909363.
Rajmohan Rajaraman: Partially supported by NSF grants CCF-1909363 and CCF-2335187.
David Stalfa: Partially supported by NSF grant CCF-1909363.
Cheng Tan: Partially supported by NSF CAREER Award #2237295

1 Introduction

As the size of Deep Neural Network (DNN) models (particularly Large Language Models) continue to increase, there is a growing need to more efficiently allocate computational resources to these models at inference time. One challenge in efficiently allocating resources is that certain large models may require powerful GPUs, while other smaller models would greatly underutilize the power of such GPUs. To combat this issue, modern GPUs (e.g., NVIDIA A30, A100, H100) include a new hardware feature called Multi-Instance GPU

(MIG). MIG enables a GPU to be partitioned into smaller hardware isolated GPU instances, each with their own processors, memory, L2 cache, and bus bandwith.

While MIG theoretically allows GPUs to avoid wasting resources, the feature raises the problem of efficiently scheduling on MIG-enabled GPUs. The problem presents two main challenges which must be considered simultaneously, and so compound the complexity of finding a solution. The first challenge is to partition the GPU into a configuration of smaller, variably sized GPU instances that can be used to execute DNN models. The second challenge is to assign models to these instances based on their resource demands. The problem is further complicated by the facts that (a) different configurations of GPU instances may have varying levels of computational and memory resources, (b) the resources are non-fungible for DNN models in the sense that increasing the size of a GPU instance may not linearly increase its performance [17], and (c) due to hardware constraints, some partitions of the GPU may not be available [13].

No prior work has provided algorithms with provable performance guarantees in the presence of (a-c), and currently deployed scheduling algorithms ignore either (b) [14], or (c) [16], or all three [19]. While this problem has gained much attention [17, 10, 11], these investigations primarily rely on heuristics with no formal guarantees. This paper is the first to establish theoretical bounds for scheduling on MIG-enabled GPUs.

We provide a natural formalization of the above problem, initiating a systematic theoretical study of scheduling splittable jobs in configurable machines. We call this problem *Configurable Machine Scheduling* or CMS. We consider machines that can be partitioned into multiple *configurations* of smaller instances, which we call *blocks*. We consider jobs that have certain demands that need to be satisfied by allocating blocks to it. Each job also has a corresponding table that specifies how much of the job's demand is satisfied by a given block type. A *schedule* specifies each machine's configuration and which job each of the machine's blocks is to execute. Our goal is to construct a schedule that satisfies all job demands using as few machines as possible.

Configurable Machine Scheduling (CMS) A CMS instance is defined by a set C of *machine configurations* and a set J of *jobs*, as well as an integer k indicating the number of available *block types*.

- Each machine configuration $\sigma \in C$ is a multiset of blocks represented as a length k vector. For $i \in [k] = \{1, \ldots, k\}$, we let $\sigma_i \in \mathbb{Z}_{\geq 0}$ indicate the number of blocks of type i in σ.
- For each job $j \in J$, there is an associated *demand* d_j as well as a length k *throughput table* f_j. For a job $j \in J$ and block type $i \in [k]$, the value of $f_j(i)$ denotes the amount of j's demand satisfied when it is executed on a single block of type i.

The goal is to satisfy all job demands on as few *machines* as possible. A machine μ specifies how each block is allocated for each job. Specifically, $\mu(i, j)$ represents the number of blocks of type i on which machine μ executes job j, with the constraint that, there exists a configuration σ such that $\sum_j \mu(i, j) \leq \sigma_i$, for each $i \in [k]$. That is, each machine has an implicit configuration, and the total number of blocks of type i used by machine μ cannot exceed the number of blocks of type i included in μ's configuration.

A *schedule* consists of a multiset M of machines, with M_μ indicating the number of instances of machine μ in M. In any schedule, all job demands must be completely satisfied, i.e. for any schedule M, we require $\sum_{\mu \in M} \sum_{i \in [k]} M_\mu \cdot \mu(i, j) \cdot f_j(i) \geq d_j$ for each job j. The formal objective is, then, to construct a schedule M that minimizes the total number of machine instances, given by $\sum_\mu M_\mu$. (See Figure 1 for an example.)

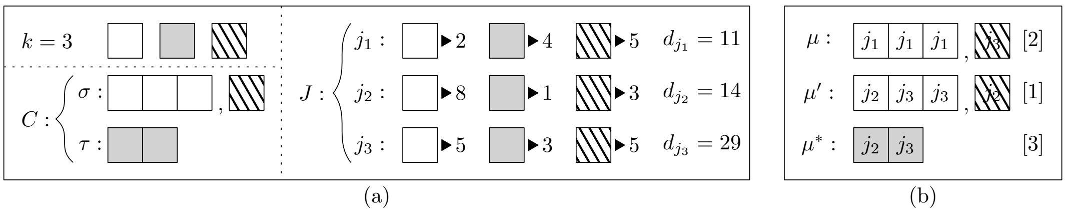
$k = 3$
C :
σ :
τ :
J :
j_1 : ▶2 ▶4 ▶5 $d_{j_1} = 11$
j_2 : ▶8 ▶1 ▶3 $d_{j_2} = 14$
j_3 : ▶5 ▶3 ▶5 $d_{j_3} = 29$
μ : j_1 j_1 j_1 j_3 [2]
μ' : j_2 j_3 j_3 j_2 [1]
μ^* : j_2 j_3 [3]
(a)
(b)

Figure 1 Example CMS instance (a) with schedule (b). The instance has $k = 3$ block types, $|C| = 2$ configurations, and $|J| = 3$ jobs. Configuration σ has three type 1 blocks, and one type 3 block. Configuration τ has two type 2 blocks. Job j_1 had demand 11, with throughput $f_{j_1}(1) = 2$, $f_{j_1}(2) = 4$, and $f_{j_1}(3) = 5$. Job j_2 had demand 14, with throughput $f_{j_2}(1) = 8$, $f_{j_2}(2) = 1$, and $f_{j_2}(3) = 3$. Job j_3 had demand 29, with throughput $f_{j_3}(1) = 5$, $f_{j_3}(2) = 3$, and $f_{j_3}(3) = 5$. The schedule uses six machines: two instances of μ, one instance of μ', and three instances of μ^*. μ and μ' have configuration σ. μ executes j_1 on three type 1 blocks and j_3 on one type 3 block. μ' executes j_2 on one type 1 block and one type 3 block, and j_3 on two type 1 blocks. Machine μ^* has configuration τ and executes j_2 on one type 2 block and j_3 on the other. Summing the throughput over all machines is sufficient to satisfy all job demands.

In any CMS instance, we assume that, for all j, f_j maps to $\{\ell \in \mathbb{N} \colon \ell \leq d_j\}$, which we can ensure with only a polynomial increase in the length of the input, and no loss in the value of the optimal solution. The *size* of a configuration σ, denoted $|\sigma|$, is the number of blocks in the configuration, i.e. $|\sigma| = \sum_{i \in [k]} \sigma_i$. Furthermore, for convenience, some of our algorithms output the schedule as a multiset of configurations, one for each machine in the output schedule, and a multiset of blocks for each job. In the appendix, we prove that this format of the output is without loss of generality, since it can be efficiently transformed to a schedule as formally defined above.

1.1 Our results

Our CMS problem formulation yields a rich landscape of optimization problems, which vary depending on the properties of block types, configurations, and the job demand tables. In this paper, we explore the general CMS problem and three restricted versions of the problem. We obtain near-tight approximations or optimal results for the associated problems (see Table 1).

Table 1 Results for Configurable Machine Scheduling. n is the number of jobs, k is the number of block types, and $c = \max_{\sigma \in C}\{|\sigma|\}$ is the maximum size of any configuration. All results are proved in this paper. The hardness of CMS with $O(1)$ jobs and $O(1)$ configuration size is unknown.

Problem	Algorithm	Approximation	Hardness
General	LP+ Greedy	$O(\log cnk)$	$\Omega(\log nk)$
$O(1)$ configurations	Extreme-Point LP Rounding	$(2+\varepsilon)$OPT + $\lvert C \rvert$ $3+\varepsilon$	2
$O(1)$ configurations of $O(1)$ size	Small/Large Job LP	$1+\varepsilon$	?
$O(1)$ number of jobs and blocks, with all configurations up to a given size	Conic Integer Combinations in Fixed Dimension	1	-

General CMS (Section 2). Using a reduction from minimum multiset multicover [15], we first observe that CMS is hard to approximate to within a factor of $\Omega(\log nk)$, where n is the number of jobs and k the number of blocks. We then present a factor $O(\log(cnk))$ approximation algorithm, where n is the number of jobs, k the number of blocks, and c is the size of the largest configuration, which is essentially tight given the above hardness result. Our algorithm constructs a schedule by greedily selecting the highest throughput configuration on the basis of a linear programming relaxation.

The logarithmic-hardness result for the general problem motivates us to consider restricted versions with a constant number of configurations, which are also of practical interest.

CMS with a constant number of configurations (Section 3). Using a reduction from Partition, we observe that CMS, *even with one configuration and two jobs*, is hard to approximate to within a factor of 2. Our main result is an algorithm that, for any instance of CMS with a constant number of configurations C and arbitrary $\varepsilon > 0$, uses at most $(2+\varepsilon)\text{OPT} + |C|$ machines where OPT is the number of machines needed in the optimal solution, asymptotically almost matching the hardness result for a constant number of configurations. We also show that our algorithm always returns a $3+\varepsilon$ approximation. Our algorithm builds on the seminal LP rounding technique of [9] and exploits the structure of extreme-point solutions to iteratively and carefully round the LP variables.

To find more tractable cases, we study a further restriction of the problem that bounds the size of configurations.

CMS with a constant number of configurations of constant size (Section 4). We next consider CMS with a constant number of configurations, each of constant size (i.e., having a constant number of blocks). We show that the problem is solvable in pseudo-polynomial time; our main result here is a PTAS based on rounding a novel LP relaxation for the problem.

Our LP based approximations require the number of configurations given in the input to be constant. Our final result explores nontrivial tractable models where the number of jobs is constant while the number and size of configurations are not constant.

CMS with a constant number of jobs and blocks, with all configurations up to a given size (Section 5). We consider CMS with a constant number of jobs, a constant number of block types, and where every configuration up to a given size is available. We give an algorithm that solves this problem optimally in polynomial time. Our algorithm uses a technique for finding conic integer combinations in fixed dimension, which was developed by Goemans and Rothvoss in their study of bin packing with constant number of job types [5]. We also show that this technique extends to a more general setting where configurations are defined by a constant number of rational polytopes.

1.2 Our Techniques

General CMS. The logarithmic factor approximation algorithm for the most general case uses two algorithmic approaches, each of which faces challenges on its own. The first approach uses the natural heuristic of greedily allocating machines to execute as much total throughput as possible. When each job can be fully executed on at most one block of each type, we show that this algorithm achieves a logarithmic approximation ratio. However, in general the algorithm's approximation ratio is $\Omega(n)$. The second approach is based on an LP relaxation that uses job-block variables to indicate how many blocks of a given type are used to execute a job. When the job-block variables are large enough (greater than 1) they can be rounded and scheduled using a multi-set multi-cover algorithm to achieve a logarithmic approximation ratio. On the other hand, variables with low values cannot be rounded up without a large loss in the approximation ratio, and cannot be rounded down because even a small fractional block allocation may represent a significant amount of *throughput*. Our solution is to marry these approaches. We first solve the LP relaxation and use a multi-set multi-cover algorithm to schedule those job-block pairs with large variable values. For any jobs with remaining demand, we show that they can be fully executed on at most one block of each type, and ensure that the maximum ratio of execution function values is at most the number of block types k. Executing the greedy algorithm on these remaining jobs yields an algorithm with a logarithmic approximation ratio.

Constant number of configurations. In this case, we use essentially the same LP as for the general problem, simplifying it to use only variables that represent how many of each block type are allocated to a job. We then build a graph with nodes representing the block types and jobs, and insert an edge between a job and block type if the corresponding variable in the LP is nonzero. We leverage extreme point properties to prove that the graph is a pseudo-forest; i.e., each component is either a tree or a tree with a cycle. Our key technical contribution is to carefully round the LP solution by exploiting this tree structure and the constraints on the possible LP values mandated by the fact that, by our construction, no block can satisfy more demand than the job requires. This algorithm returns a $(2 + \varepsilon)\textsc{opt} + |C|$ approximation. We extend this result to a $(3 + \varepsilon)\textsc{opt}$ approximation (which is better for $\textsc{opt} < |C|$) by running the above algorithm on each subset of the input configurations, and then returning the best solution.

Constant number of configurations of constant size. The LP used in the preceding two variants of the problem has an integrality gap of 2. This holds even for simple instances with one configuration of constant size and two jobs. To obtain a PTAS for a constant number of configurations of constant size, we first divide the jobs into small and large jobs based on whether the number of machines needed to serve their demand exceeds a constant threshold (based on a parameter ε). We then formulate a new LP that imposes different constraints for the small jobs taking into account that there is a bounded number of ways their demands can be allocated. Through a careful rounding of the LP, we derive a $(1 + \varepsilon)$-approximation algorithm for the problem.

Constant number of jobs and blocks, and all configurations up to a given size. In this setting neither the number or configurations nor the size of any configuration is constant, and so any approach based on our LP relaxations faces the obstacle that either the program has an integrality gap of at least 2, or is intractable. Therefore, in order to solve this problem optimally, more sophisticated techniques are required. The approach we develop is based on a technique for finding conic integer combinations in fixed dimension which was developed by Goemans and Rothvoss in their study of bin packing with constant job types [5]. The challenges of this approach lie in formulating the problem to appropriately leverage the power of the apparatus. In our case, this involves providing a rational representation of the problem on an individual machine that yields a complete solution when combined with the representations of other machines. In the general case, the nonlinear relationship of blocks to job throughput *and* to configuration size renders the problem extremely difficult, even when jobs and blocks are constant. However, when all configurations up to a given size are available, a configuration can be represented as a linear combination of block allocations, making the problem tractable. In fact, as long as the number of jobs and blocks types is constant, the general technique for finding conic integer combinations allows us to devise an optimal polynomial time algorithm whenever the set of configurations can be represented using a constant number of rational polytopes.

1.3 Related work

Configurable machine scheduling has connections to many well-studied problems in combinatorial optimization, including bin-packing, knapsack, multiset multicover, and max-min fair allocation. General CMS generalizes the multiset multicover problem [8, 6, 15], for which the best approximation factor achievable in polynomial time is $O(\log m)$ where m is the sum of the sizes of the multisets [15, 18]. The hardness of approximating the problem to within an $O(\log n)$ factor follows from the result for set cover [4].

As we note above, CMS is NP-complete even for the case of one configuration and two jobs. The single configuration version can be viewed as a fair allocation problem with each block representing an item and each job representing a player that has a value for each item (given by the demand table) and a desired total demand. The objective is to minimize the maximum number of copies we need of each block so that they can be distributed among the players satisfying their demands. In contrast, the Santa Claus problem in fair allocation [1] (also studied under a different name in algorithmic game theory [12]) aims to maximize the minimum demand that can be satisfied with the available set of blocks. The best result for the Santa Claus problem is a quasi-polynomial $O(n^{\varepsilon})$-approximation algorithm, where $\varepsilon = O(\log\log n/\log n)$ [2], though factor $O(1)$ approximation algorithms are known for special cases (e.g., see [3]).

1.4 Discussion and Open Problems

Our study has focused on a *combinatorial* version of CMS in which each machine can be configured as a collection of abstract blocks. It is also natural to consider a *numerical* version of CMS in which each block type is an item of a certain size, and each configuration has a certain capacity and can only fit blocks whose sizes add up exactly to its capacity. Note that instances of numerical CMS can be presented more compactly than general instances of CMSsince the allowable configurations can be captured by configuration capacities and block sizes. The approximation ratios established for CMS apply to numerical CMS as well; however, it is not certain that there is also a logarithmic hardness for numerical CMS. Thus, an intriguing open problem is whether numerical CMS admits an approximation factor significantly better than the logarithmic factor established in Section 2. Also of interest is a numerical CMS variant where all capacity-bounded configurations are allowed, for which we believe techniques from unbounded knapsack [7] and polytope structure results of the kind we show in Section 5 would be useful.

Our results indicate several directions for future research. One open problem is to devise approximation algorithms that leverage structure in the set of available configurations. In practice, the configuration sets associated with multi-instancing GPUs might not be arbitrary sets, e.g. the blocks of Nvidia's A100 GPU are structured as a tree and every valid configuration is a set of blocks with no ancestor-descendant relations [17]. Showing improved bounds for such cases seems to be a challenging, but potentially fruitful area of research.

Another open problem lies in shrinking the gap between our upper and lower bounds. The hard instance for CMS with a constant number of configurations has a constant-size solution, showing for instance that it is NP-hard to distinguish a problem with solution size 1 from one with solution size 2. These lower bounds are sufficient to show hardness of approximation, but do not rule out the possibility of asymptotic PTAS (even additive constant approximations). Furthermore, we have not been able to show any hardness for CMS with a constant number of configurations of constant size, and this is an important and interesting open problem.

Finally, our focus has been on the objective of minimizing the number of machines, which aims to meet all demands using minimum resources. Our results can be extended to minimizing makespan, given a constant number of machines. However, approximations for other objectives such as completion time or flow time, in both offline and online settings, are important directions for further research.

2 General CMS logarithmic approximation

In this section, we consider the most general model of CMS with an arbitrary configuration set C over k block types, and n jobs in J. Our first lemma presents an approximation-preserving reduction from multiset multicover to CMS. We thus obtain that no polynomial time algorithm can achieve an approximation ratio better than $\Omega(\log nk)$ (assuming P $\neq$ NP). The lemma also implies that an improvement to our approximation ratio would yield an improvement to the best known approximation for multiset multicover.

▶ **Lemma 1.** *There is an approximation-preserving reduction from the multiset multicover problem to* CMS*.*

Proof. Consider an arbitrary instance I of multiset multicover. Let $\mathcal{U}$ denote the set of elements and $\mathcal{C}$ the collection of multisets in the set cover instance. Let r_e denote the coverage requirement for element e. We can assume without loss of generality that there do not exist two multisets S and S' with $S \subseteq S'$, since we can eliminate S from the set collection otherwise. We construct an instance of CMS where each multiset S is a configuration and each element e is both a block type and a job. The job e has demand r_e, which can only be satisfied by r_e blocks of type e.

Any multiset multicover solution, given by a collection M of multisets, corresponds to a solution for CMS: each multiset S in M is a machine configured according to S. Therefore, the number of multisets in M is the same as the number of machines in the CMS solution. Furthermore, since each element e is covered r_e times in M, it follows that each job e has r_e occurrences of block type e included in CMS solution, thus satisfying the demand for e. Similarly, every CMS solution with m machines is a collection of m multisets, with each multiset corresponding to the configuration of a machine. Since the objective function value achieved by each of the two solutions is identical, the reduction is approximation-preserving. ◀

Algorithm 1 Logarithmic Approximation for CMS

```
1 Formulate and Solve a Linear Relaxation (Constraints 1-4)
    Round variables down if their fractional component is less than $(1/2k)$
2 Solve Problem over the Integer Components of Variables (Algorithm 2)
    Construct partial schedule $S$ via multiset-multicover defined over variable integer components
3 Greedily Schedule any Jobs with Remaining Demand (Algorithm 3)
    Construct a partial schedule $S'$ to satisfy any remaining demand by greedily configuring each machine to maximize throughput
4 Output the schedule formed by the additive union $(S \oplus S) \oplus (S' \oplus S')$
```

The main result of this section is Algorithm 1, an $O(\log(\max_{\sigma \in C}\{|\sigma|\} \cdot n \cdot k)$-approximation algorithm for CMS. The first step of Algorithm 1 solves a linear relaxation of CMS, which minimizes $\sum_\sigma y_\sigma$ subject to:

$$\sum_j x_{i,j} \leq \sum_{\sigma \in C} y_\sigma \cdot \sigma_i i \in [k] \tag{1}$$

$$\sum_i f_j(i) \cdot x_{i,j} \geq d_j \qquad j \in J \tag{2}$$

$$x_{i,j} \geq 0 \qquad i \in [k] \text{ and } j \in J \tag{3}$$

$$y_\sigma \geq 0 \qquad \sigma \in C \tag{4}$$

Terms and Constraints. Each variable $x_{i,j}$ indicates the number of blocks of type i that are assigned to execute job j. Each variable y_σ indicates the number of machines that use configuration σ. Constraint 1 ensures a schedule cannot use more blocks of a given type than appear across all allocated machines. Constraint 2 states that the total number of blocks executing a job must be sufficient to satisfy its demand. It is easy to verify that this program relaxes CMS and is solvable in polynomial time.

Let (x^*, y^*) be variable assignments that yield an optimal solution to (1-4). For the second step of Algorithm 1, we separate the integer from the fractional components of the x-variables. We define $\bar{x}_{i,j} = \lfloor x^*_{i,j} \rfloor$.

We define $\hat{x}_{i,j} = 0$ if either (i) $(x^*_{i,j} - \lfloor x^*_{i,j} \rfloor) < \frac{1}{2k}$ or (ii) $f_j(i) \cdot (x^*_{i,j} - \lfloor x^*_{i,j} \rfloor) < \max_{i'}\{f_j(i') \cdot (x^*_{i,j} - \lfloor x^*_{i,j} \rfloor)\}/k$, otherwise $\hat{x}_{i,j} = x^*_{i,j} - \lfloor x^*_{i,j} \rfloor$. The second step of Algorithm 1 then calls Algorithm 2 to provide a schedule for the problem $(C, \bar{J}, k)$ with modified demands $\bar{d}_j = \min\{d_j, \sum_i f_j(i) \cdot \bar{x}_{i,j}\}$.

Algorithm 2. We define the set $A = \{(\sum_j \lfloor x_{i,j} \rfloor, i) : i \in [k]\}$. We construct schedule S by using the greedy multiset multicover algorithm given in [15] on the instance (A, C).

Step three of Algorithm 1 then constructs a schedule S' to satisfy any remaining demand given by the fractional components $\hat{x}$ via Algorithm 3, which greedily allocates the highest throughput machines until all demands are met. Finally, Algorithm 1 outputs the schedule S^* such that, $S^*_\mu = 2(S_\mu + S'_\mu)$ for each μ.

Algorithm 3. For each $j \in J$ and $i \in [k]$, we initialize $\hat{f}_j(i) = f_j(i)$. While there is some job that hasn't been fully executed, do the following. Compute a machine $\mu_{\max}$ by iterating over all configurations σ and greedily allocating each block of σ to the job with the highest

throughput for that block (accounting for demand already satisfied). Of these machines, greedily set $\mu_{\max}$ to be the one with highest total throughput. Allocate enough instances of μ such that some job's remaining demand becomes less than $\hat{f}_j(i)$ for some i. Then, for each j, i, update $\hat{f}_j(i) \leftarrow \min\{\hat{f}_j(i), D_j\}$ where D_j is the job's remaining demand. Then repeat.

In the remainder of the section, we provide analysis of Algorithm 1. The following lemmas establish bounds on the lengths of the schedules produced by Algorithm 3. We note that there exist instances for which Algorithm 3 produces schedules with length $\Omega(n)$, and so it cannot be used to solve CMS without some additional processing (which, for us, involves reducing the instance via Algorithm 2).

▶ **Lemma 2.** *If $d_j \le \sum_i f_j(i)$ for all i (i.e. each job can be executed on at most one block of each type) then Algorithm 3 returns an $O(\log(\max_{\sigma\in C}\{|\sigma|\} \cdot nk\rho))$ approximation, where $\rho = \max_{i,i',j}\{f_j(i)/f_j(i')\}$.*

Proof. Consider the following integer program formulation of CMS, which minimizes $\sum_{t,\sigma} w_{t,\sigma}$ subject to: $\sum_{t,\sigma,i} z_{t,i,j} \cdot f_j(i) \ge d_j,\ j \in J$ and $\sum_j z_{t,i,j} \le w_{t,\sigma} \cdot \sigma_i,\ t \in \mathbb{N}, \sigma \in C, i \in [k]$ and $\sum_\sigma w_{t,\sigma} \le 1,\ t \in \mathbb{N}$ and $w_{t,\sigma}, z_{t,\sigma,i,j} \in \mathbb{N},\ t \in \mathbb{N}, \sigma \in C, i \in [k], j \in J$. In this program, $w_{t,\sigma} = 1$ if the tth machine instance has configuration σ (0 otherwise), $z_{t,i,j}$ = the number of blocks of type i that the tth machine uses to execute job j. It is easy to see that the program relaxes CMS.

Let (w^*, z^*) be a variable assignment that minimizes the objective with value η. Then

$$\sum_j d_j \le \sum_{\mu\in S} \mu(i,j) \cdot f_{\mu(i)}(i) \le \eta \cdot k \cdot \max_{\sigma\in C}\{|\sigma|\} \cdot \max_{i,j}\{f_j(i)\}$$

The fact that $d_j \le \sum_i f_j(i)$ for all j implies that $\eta \le nk$ since, in the worst case, each block is executed on its own machine. So we can infer that $\log(\sum_j d_j) \le 2 \cdot \log(nk \cdot \max_{\sigma\in C}\{|\sigma|\} \cdot \max_{i,j}\{f_j(i)\})$. In the remainder of the proof, we show that the greedy algorithm has an approximation ratio of $\log(\sum_j d_j)$.

Let $(\bar{w}, \bar{z})$ be the variable assignment given by schedule produced by Algorithm 3. Let $\bar{u}_t = \sum_{\sigma,i,j} \bar{z}_{t,\sigma,i,j}$ for every t. We define u^*_t and v^*_t to be any values that satisfy the following equalities: $u^*_t + v^*_t = \sum_{\sigma,i,j} f_j(i) z^*_{t\sigma,i,j},\ t \in \mathbb{N}$ and $\sum_t v^*_t = \sum_j d_j - \sum_{t\le\eta} \sum_{\sigma,i,j} f_j(i)\bar{z}_{t,\sigma,i,j}$ and $\sum_t u^*_t = \sum_j d_j - \sum_t v^*_t$. Informally, $\bar{u}_t$ is amount of throughput achieved by machine instance t of $(\bar{w}, \bar{z})$. u^*_t is the amount of throughput achieved by machine instance t of (w^*, t^*) that is also satisfied by one of the first η machines in $(\bar{w}, \bar{z})$. v^*_t is the remaining demand satisfied by machine instance t of (w^*, z^*). It is easy to see that, $\sum_{t\le\eta} \bar{u}_t = \sum_t u^*_t$. We show that $2\sum_{t\le\eta} \bar{u}_t \ge \sum_{t\le\eta} v^*_t$. Suppose, for the sake of contradiction, that the claim is false. Then for some t, $2\bar{z}_{\tau+1,\sigma,i,j} < v^*_t$. However, since v^* represents demand not satisfied by $(\bar{w}, \bar{z})$ in the first η machines, and since the machine $\mu_{\max}$ in Algorithm 3 has throughput at least half of the maximum (proved below), in this case the throughput of μmax would be at least v^*_t. This is a contradiction.

We now show that, when Algorithm 3 constructs $\mu_{\max}$, its throughput is at least half the throughput of the highest throughput machine possible. Let σ be the configuration used by $\mu_{\max}$. We show that the maximum throughput machine μ^* over σ has throughput no more than twice that of $\mu_{\max}$. We consider an integer program formulation of the problem of finding the maximum throughput machine: $w_j \le d_j,\ j \in J$ and $w_j \le \sum_i z_{i,j} f_j(i),\ j \in J$ and $\sum_j z_{i,j} \le \sigma_i,\ i \in B$.

The set B includes all block *instances* in σ. Let (w^*, z^*) represent the solution to this program given by the optimal machine μ^*. Let $(\bar{w}, \bar{z})$ represent the solution to this program given by $\mu_{\max}$. For each block i, we set u_i and v_i to be any values that satisfy the following equations. $u_i + v_i = \sum_j z^*_{i,j} f_j(i),\ i \in [k]$ and $\sum_i v_i = \sum_j w^*_j - \sum_{i,j} \bar{z}_{i,j} f_j(i)$ and $\sum_i u_i = \sum_j w^*_j - \sum_i v_i$. Informally, u_i is the amount of demand satisfied by block i of μ^* that is also satisfied by $\mu_{\max}$, and v_i u_i is the amount of demand satisfied by block i of μ^* that is not satisfied by $\mu_{\max}$. For any block i, it is easy to see that $\sum_j \bar{z}_{i,j} \geq u_i$. We can also infer that $\sum_j \bar{z}_{i,j} \geq v_i$ because, otherwise, the greedy algorithm would have chosen to execute the same job as μ on block i. This proves the result.

Since $\sum_{t \leq \eta} \bar{u}_t = \sum_t u^*_t$ and $4 \sum_{t \leq \eta} \bar{u}_t \geq \sum_t v^*_t$, the total demand satisfied by $(\bar{w}, \bar{z})$ is at least $1/4$ the total demand. It is straightforward to generalize this reasoning to show that every η machine instances of $(\bar{w}, \bar{z})$, reduce the total demand by a factor of $1/4$, which proves our claim. ◀

▶ **Theorem 3.** *Algorithm 1 returns an $O(\log(\max_{\sigma \in C}\{|\sigma|\} \cdot n \cdot k))$ approximation to the* CMS *problem.*

Proof. Let S and S' represent the schedules produced by Algorithm 2 and Algorithm 3, respectively. We first argue that S has length $O(\log(\max_\sigma\{|\sigma|\} \cdot n)) \cdot \text{OPT}$. Algorithm 2 reduces scheduling the integer components of the variables to an instance of multi-set multi-cover in which there are n elements and the largest covering multi-set has size $\max_\sigma\{|\sigma|\}$. The claim follows directly from Theorem 5.1 in [15].

We now show that S' has length $O(\log(\max_\sigma\{|\sigma|\} \cdot nk)) \cdot \text{OPT}$. By Lemma 2, we need only to show that $d_j \leq \sum_i f_j(i)$ and that $\max_{j,i,i'}\{f_j(i)/f_j(i')\} = O(k)$. We can infer $d_j \leq \sum_i f_j(i)$ because (C, J, k) with modified demand tables and demands $\hat{f}, \hat{d}$ is defined over $\hat{x}$, so each job can be completely executed by one block of each type. Also, the definition of $\hat{x}$ entails that for each j, every nonzero value of $\hat{x}_{i,j}$ (resp. $\hat{f}_j(i) \cdot \hat{x}_{i,j}$) is within a factor of $2k$ (resp. k) of every other.

Finally, in defining $\hat{x}$, we rounded down $x^*_{i,j}$ if (i) $z^*_{i,j} < 1/2k$ or if (ii) $z^*_{i,j} \cdot f_j(i) < \max_{i'}\{z^*_{i',j} \cdot f_j(i')\}/k$. Job j's total reduction in demand from (i) is no more than $d_j \sum_i x^*_{i,j} - \bar{x}_{i,j} \leq d_j/2$, which is accounted for by doubling S_1 and S_2 in the output. Job j's total reduction in demand due to (ii) is at most $\max_{i'}\{z^*_{i',j} \cdot f_j(i')\}$ which is accounted for in setting $\hat{x}_{i,j} = 2z^*_{i,j}$ for all remaining i's. Each doubles our approximation ratio. ◀

3 CMS with a constant number of configurations

We consider CMS with n jobs and a set C of $O(1)$ configurations, each of arbitrary size. We first observe that the problem is NP-hard to approximate to within a factor of two.

▶ **Lemma 4.** CMS *with a constant number of configurations is hard to approximate to within a factor of 2.*

Proof. We present a reduction from Partition to combinatorial CMS. Given an instance of Partition with a set S of n elements $0 < a_1 < a_2 < \cdots < a_n$, we construct the following instance. We consider one configuration that contains n blocks all of a different type, labeled $1, ..., n$. We have two jobs j_1, j_2, both with the demand function given by $f(i) = a_i$. The demand for each job is $\frac{1}{2} \sum_i a_i$.

We claim that the number of machines needed for scheduling the job is one if and only if the Partition instance has a yes answer. If the Partition instance has a yes answer, then there exists a way to split the n blocks into two parts so that each part's value adds up to $\sum_i a_i/2$. We use one machine, and assign the blocks to each job according to the Partition solution. The demand function ensures that the demand of the job is satisfied. Conversely, if

Algorithm 4 Schedule for CMS with $O(1)$ configurations

```
Input: A CMS instance $(C, J, k)$
1  $L \leftarrow \{ \lfloor (1+\varepsilon)^i \rfloor \mid 0 \le i \le \log_{1+\varepsilon}(\sum_j d_j) \}$, $Sol \leftarrow \{\}$
2  foreach $C^* \in P(C)$, the powerset of $C$ do
3      foreach $m \in L^{|C^*|}$, where $m_\sigma$ is the number of copies of $\sigma$ do
4          $B^* \leftarrow \{i \in [k] \mid \exists \sigma \in C^*$ s.t. $i \in \sigma\}$ is the set of blocks present in $C^*$
5          Construct the feasibility LP, $LP_f$ with constraints from equations (1′), (2),
             and (3).
               $\sum_j x_{i,j} \le \sum_{\sigma \in C^*} m_\sigma \cdot \sigma_i$        block types $i \in B^*$        (1′)
6          if $LP_f$ is feasible with extreme-point solution $x$ then
7              Graph $G \leftarrow (J \cup B^*, E)$ with $E = \{ (i,j) \mid x_{i,j} > 0 \}$
8              foreach Component $S \in G$ that has a cycle $K$ do
9                  Pick some job $j$ in the cycle $K$, and let $b_1, b_2$ be its neighbors in the
                     cycle
10                 if $x_{b_1,j} \cdot f_j(b_1) \ge x_{b_2,j} \cdot f_j(b_2)$ then $E \leftarrow E \setminus \{(b_2, j)\}$ else
                   $E \leftarrow E \setminus \{(b_1, j)\}$
11                 Make $j$ the root of the remaining tree $S$
12             foreach Job $j \in J$ do
13                 for the parent block $p$ of $j$, do $x^*_{p,j} \leftarrow \lfloor 2x_{p,j} \rfloor$
14                 foreach child block $c$ of $j$ do $x^*_{c,j} \leftarrow \lceil 2x_{c,j} \rceil$
15             foreach Configuration $\sigma \in C^*$ do $y^*_\sigma \leftarrow 2m_\sigma + 1$
16             if fewer configurations are used in $y^*$ than in $Sol$ then $Sol \leftarrow (x^*, y^*)$
17             break out of iteration
18 return $Sol$ transformed from $(x^*, y^*)$ format to a machine schedule according to
     Algorithm 6
```

the demand of the two jobs is satisfied by one machine, then each job has at least $\sum_i a_i/2$ demand satisfied. According to the demand function f, $\sum_i a_i$ is the maximum amount of demand that can be satisfied by this configuration. Thus each job has exactly $\sum_i a_i/2$ demand satisfied, and so there is some partition of S into two parts such that each part sums to $\sum_i a_i/2$. ◀

Our main result in this section is a polynomial time algorithm that returns a solution with cost the minimum of $(2+\epsilon)$OPT $+ |C|$ and $(3+\varepsilon)$OPT, for arbitrary $\varepsilon > 0$, where OPT is optimal cost. Our algorithm, detailed in Algorithm 4, guesses the number of each configuration used in an optimal solution, to within a factor of $1+\varepsilon$ (see line 3), and then builds on the paradigm of [9] by carefully rounding an extreme-point optimal solution for a suitable instantiation of LP(1-4) (given in line 5).

Using extreme-point properties, we establish Lemma 5, the proof of which closely follows [9].

▶ **Lemma 5.** *Every component in graph G of line 7 has at most one cycle.*

Proof. This proof follows a similar structure as the proof of Lemma 17.6 in [18]. We will use a proof by contradiction. First, consider a component in G, called G_c. Then consider the restriction of the LP, LP_c, to only the jobs and block types present in the component. Also let x_c be the restriction of x to those jobs and blocks present in the component. Let $x_{\bar{c}}$ be the rest of x. Note that x_c is a feasible solution to LP_c since all the blocks that satisfy demand for jobs in G_c are connected to those jobs in the original graph G and thus are also included in G_c, so we continue to satisfy all the demand for these jobs. Now assume for contradiction that x_c is not an extreme point in LP_c. Then $\exists x_1, x_2, \lambda$ where x_1 and x_2 are feasible solutions to LP_c and $\lambda \in (0,1)$ such that we have $x_c = \lambda \cdot x_1 + (1-\lambda)\cdot x_2$.

Now we show that $x_1 + x_{\bar{c}}$ and $x_2 + x_{\bar{c}}$ are feasible solutions to the LP. First, consider that x_1, x_2 have disjoint jobs and block types from $x_{\bar{c}}$. Thus, we can consider their corresponding constraints separately. Furthermore, together, x_1, x_2, and x_c cover all the constraints (since they cover all jobs and block types). Thus we need only verify that x_1, x_2 satisfy their constraints, and $x_{\bar{c}}$ satisfies its constraints. Since x_1, x_2 are feasible solutions to LP_c we know they satisfy the constraints in LP relevant to them. And since $x_{\bar{c}}$ is part of the feasible solution x, it must also satisfy the constraints relevant to it. Between the two, all the constraints of the LP are satisfied, since together they cover all jobs and blocks.

But then since $x = \lambda \cdot (x_1 + x_{\bar{c}}) + (1-\lambda)\cdot(x_2 + x_{\bar{c}})$ we can say that x is a convex combination of two other solutions. Thus, x is not an extreme point solution. But, since x is an extreme point solution, we reach a contradiction.

Therefore, x_c must be an extreme point solution in LP_c. We know that the number of tight constraints in an extreme point solution is at least as many as the number of variables. Thus at most $|B_c^*| + |J_c|$ constraints can be not tight (coming from constraints $1'$ and 2). Since we create an edge only if an x variable is nonzero, or equivalently, its nonzero constraint is not tight, we know that the number of edges in G_c must be at most the number of blocks + jobs in G_c. In other words, the number of edges is at most the number of nodes. Since C was chosen arbitrarily, we conclude that every component in G has at most one cycle. ◀

▶ **Lemma 6.** *Algorithm 4 returns a feasible integer solution to* LP(1-4).

Proof. Since the algorithm returns the least cost rounded solution over all iterations, we need to show that (x^*, y^*) is a feasible integer solution to LP(1-4). By our rounding, $x^*_{i,j}$ and y^*_σ are integers for each i, j, σ. It remains to show that (x^*, y^*) is feasible in LP(1-4). Constraints 3 and 4 are true by definition of x^*, y^*.

We now consider constraint 1. If a block type i is not in B^*, then this constraint is satisfied because $x_{i,j} = 0$ for all j, and thus $x^*_{i,j} = 0$ for all j. Now we consider blocks that are in B^*. By Lemma 5, each component of G has at most one cycle. In the algorithm we remove an edge from each of these cycles, so the resulting graph is a forest. Thus each block type i has one parent and so is a child of one job. This means that all $x_{i,j}$ variables associated with block i are rounded as $\lfloor 2x_{i,j} \rfloor$, except for the parent of i, p_i. So

$$\begin{aligned}\sum_j x^*_{i,j} = \sum_{j \neq p_i} \lfloor 2x_{i,j} \rfloor + \lceil 2x_{i,p_i} \rceil \le \sum_j 2x_{i,j} + 1 &\le \sum_{\sigma \in C^*} 2m_\sigma \cdot \sigma_i + 1 \\ &\le \sum_{\sigma \in C^*} (2m_\sigma + 1)\cdot \sigma_i \le \sum_{\sigma \in C} y^*_\sigma \cdot \sigma_i\end{aligned}$$

The third inequality follows from constraint $1'$ since x satisfies LP_f, and the fourth inequality holds since $i \in B^*$ implying that there is at least one $\sigma \in C^*$ with $\sigma_i \ge 1$. Thus, constraint 1 is satisfied.

Now, we consider constraint 2. First, we consider some job j whose edge was not removed. Then, since G becomes a forest after pruning edges we obtain that either the children or the parent of j satisfy at least half of its demand. If its children satisfy at least half of its demand then we have $\sum_{\text{children of } j} f_j(i) \cdot x_{i,j} \geq \frac{1}{2} d_j$ and thus we obtain

$$\sum_i f_j(i) \cdot x^*_{i,j} \geq \sum_{\text{children of } j} f_j(i) \cdot \lceil 2x_{i,j} \rceil \geq 2 \sum_{\text{children of } j} f_j(i) \cdot x_{i,j} \geq d_j,$$

Therefore, the constraint is satisfied. Otherwise, its parent p satisfies at least half of its demand implying that $x_{p,j} \geq \frac{1}{2}$ since we have $f_j(p) \leq d_j$ by our assumption on the input. Then, $x^*_{p,j} = \lfloor 2x_{p,j} \rfloor > x_{p,j}$, yielding $\sum_i x^*_{i,j} \cdot f_j(i) \geq \sum_i x_{i,j} \cdot f_j(i) \geq d_j$ since x is a feasible solution to LP_f. So the constraint is satisfied.

Finally, we consider any job j that had an edge removed in the cycle. Assume without loss of generality that (b_2, j) was removed from the graph. Since j is the root of the tree it is in (by line 11), all of its neighboring blocks are its children. Then, we have

$$\sum_i x^*_{i,j} \cdot f_j(i) = \sum_{i \neq b_2} \lceil 2x_{i,j} \rceil \cdot f_j(i) \geq x_{b_1,j} \cdot f_j(b_1) + x_{b_2,j} \cdot f_j(b_2) + \sum_{i \neq b_1, b_2} 2x_{i,j} \cdot f_j(i)$$
$$\geq \sum_i x_{i,j} \cdot f_j(i) \geq d_j.$$

The second inequality comes as a consequence of line 10 and the fact (b_2, j) was removed from the graph, which implies that $2x_{b_1,j} \cdot f_j(b_1) \geq x_{b_1,j} \cdot f_j(b_1) + x_{b_2,j} \cdot f_j(b_2)$. So the constraint is satisfied in all cases. Thus (x^*, y^*) is a feasible integer solution to LP(1-4). ◀

▶ **Theorem 7.** *Algorithm 4 returns a* $\min\{(3+\varepsilon)\text{OPT}, (2+\epsilon)\text{OPT} + |C|\}$ *approximation to the* CMS *problem in polynomial time if the number of configurations is constant.*

Proof. We show that the algorithm returns a solution with the stated approximation factor. Consider the iteration where $C^* = C^{\text{OPT}}$ where C^{OPT} is the set of configurations used by an optimal integer solution. The algorithm will iterate through potential counts m_σ for each σ in C^*, round and return a schedule the first time LP_f has a feasible solution; let m be the vector of how many configurations are used in this iteration. By Lemma 6, the solution returned is feasible.

We now bound the cost by first arguing that $\sum_\sigma m_\sigma \leq (1+\varepsilon)\text{OPT}$. Observe that the y values in the optimal integer solution to LP(1-4) would yield a feasible solution to LP_f if they equalled the corresponding m values in LP_f (namely by setting the x variables in LP_f to the x values in the optimal integer solution to LP(1-4)). For each such y_i value, consider p_i, the first power of $1+\varepsilon$ that is at least y_i. Then, we have $y_i \leq \lfloor p_i \rfloor \leq (1+\varepsilon)y_i$. By definition of L, we will set values for m_σ such that they are greater than and within a factor of $(1+\varepsilon)$ of the y values from the optimal integer solution. Thus they will be feasible, since they use at least as many of each configuration and $\sum_\sigma m_\sigma \leq (1+\varepsilon)\text{OPT}$. Since we iterate through the m values in increasing order of $\sum_\sigma m_\sigma$, the first feasible solution will use at most this many configurations.

Now consider that the rounded solution y^* has $\sum_\sigma y^*_\sigma \leq \sum_\sigma (2m_\sigma + 1) = 2\sum_\sigma m_\sigma + |C^{OPT}| \leq 2(1+\varepsilon)\text{OPT} + |C^{OPT}|$. Since the optimal integer solution uses at least 1 of each configuration in C^{OPT}, we have that $\sum_\sigma y^*_\sigma \leq (3+\varepsilon)\text{OPT}$ and also that $\sum_\sigma y^*_\sigma \leq (2+\varepsilon)\text{OPT} + |C|$.

Finally, we prove that the runtime of algorithm 4 is polynomial if $|C| = O(1)$. The first for loop in the algorithm ranges over $2^{|C|}$ values. The inner for loop ranges over $(\log L)^{|C^*|}$ values. Remember that $L = \sum_j d_j$. But then $L \leq n \cdot \max_j d_j$. Thus the inner loop ranges over $\leq (\log(n \cdot \max_j d_j))^{|C^*|} \leq (\log n + \log \max_j d_j)^{|C|}$ values. Since d_j is specified as a number, it is specified using $\log d_j$ bits. Thus the inner loop runs a number of times polynomial in the input, except for the number of configurations. Lastly we analyze the body of the inner for loop. The size of the LP is polynomial in the size of the input, and thus constructing and solving it takes time polynomial in the size of the input. Constructing the graph takes time polynomial in the size of the LP, as does rounding using the graph. Thus overall the runtime of the algorithm is polynomial in the size of the input, except for it being exponential in the number of configurations. So if $|C| = O(1)$, the algorithm is polynomial. ◀

4 CMS with a constant number of configurations of constant size

In this section, we consider CMS with n jobs, a set C of a constant number of configurations with the additional constraint that each configuration has at most a constant number b of blocks. Let k be the total number of block types. Since $|C|$ and b are both constant, $k \le b|C|$ is a constant. In Section 4.2, we present our main result of this section, a PTAS for the problem. As a warmup, in Section 4.1, we present an optimal dynamic programming algorithm for the problem, which takes time $(nbd_{\max})^{O(k+|C|)}$; this is pseudo-polynomial time for constant k and $|C|$.

4.1 A pseudo-polynomial time algorithm

We present an optimal dynamic programming algorithm that takes time polynomial in n and the maximum demand. Recall that C denotes the set of configurations, and $|C|$ is constant. Let N denote the total number of machines. Then, there are $\binom{N+|C|-1}{|C|-1}$ ways of distributing the N machines among these configurations. Each way yields a specific number of blocks of each type. For given n_i, $1 \le i \le k$, let $S(j, n_1, n_2, \ldots, n_k)$ be True if the demand of jobs 1 through j can be satisfied using n_i blocks of type i, for each i. Then, we have

$$S(j, n_1, n_2, \ldots, n_k) = \bigvee_{m_i \le n_i, \forall i} (S(j-1, n_1 - m_1, n_2 - m_2, \ldots, n_b - m_k) \wedge T(j, m_1, m_2, \ldots, m_k))$$

where $T(j, m_1, m_2, \ldots, m_k)$ is true if and only d_j can be satisfied using m_i blocks of type i, for each i. Note that $T(j, m_1, m_2, \ldots, m_k)$ can be computed easily by inspecting the demand table of job j and d_j.

The algorithm computes $S(j, n_1, n_2, \ldots, n_k)$ for $1 \le j \le n$, $n_i \le Nb$; the number of different tuples equals $n(Nb)^k$. The time taken to compute a given $S(j, n_1, n_2, \ldots, n_k)$, given $S(j-1, n_1 - m_1, n_2 - m_2, \ldots, n_k - m_k)$ for all choices of m_i's, is proportional to the number of different choices of m_i's, which is bounded by $\binom{N+|C|-1}{|C|-1}$. We thus obtain that S can be computed in $n(Nb)^{O(k+|C|)}$. This computation, coupled with a binary search over possible values of N, yields the desired algorithm. Since N is bounded by n times the maximum demand, we obtain a pseudopolynomial time optimal algorithm if $|C|$ and k are bounded.

4.2 A polynomial-time approximation scheme

Blocks and patterns. Abusing notation slightly, we use $f_j(\sigma)$ to denote the total demand of j satisfied if every block in configuration σ is assigned to j. We partition the set J of jobs into two groups: the *large* jobs L and *small* jobs S. A job j is small if there exists a configuration σ such that $f_j(\sigma) \ge \varepsilon d_j$; otherwise, j is large.

Let $\varepsilon > 0$ be a given constant parameter, and let $\lambda = \varepsilon/(2b)$. We define a *pattern* π to be a size k list of integers π_1 through π_k that sum to no more than b/λ^2; π_i denotes the number of blocks i in pattern π. Let W be the set of all possible patterns. So, $|W| \leq (b/\lambda^2)^k$. We assign each small job a *type*. Job j is of type $t \in 2^W$ if each pattern $\pi \in t$ is such that the demand of j is satisfied if j is allocated π_i blocks i for $1 \leq i \leq k$. So, the number of job types is at most $2^{(b/\lambda^2)^k}$. Define constant $\gamma = 2^{(b/\lambda^2)^k}$.

The linear program. We define a linear program PTAS-LP using the following notation. In PTAS-LP, σ ranges over all configurations in C, $p \in \{1, \dots, k\}$ ranges over types of blocks, $x_{j,p}$ is the number of p-blocks dedicated to processing a large job j, y_σ is the number of machines we use with configuration σ, σ_p is the number of p-blocks in σ, $z_{t,\pi}$ is the number of small jobs of type t that are distributed according to pattern π, and n_t is the number of small jobs of type t. Recall that π_p is the pth entry of π. PTAS-LP minimizes $\sum_{\sigma \in C} y_\sigma$ subject to the following constraints

$$\sum_{j \in L} x_{i,j} + \sum_{t \in 2^W} \sum_{\pi \in W} (z_{t,\pi} \cdot \pi_i) \leq \sum_\sigma y_\sigma \cdot \sigma_i \qquad i \in [k] \tag{5}$$

$$\sum_{i \in [k]} f_j(p) \cdot x_{i,j} \geq d_j \qquad j \in L \tag{6}$$

$$\sum_\pi z_{t,\pi} \geq n_t \qquad t \in 2^W \tag{7}$$

$$x_{i,j}, y_\sigma, z_{t,\pi} \geq 0 \qquad j \in L, i \in [k], \sigma, t \in 2^W, \pi \in W \tag{8}$$

Constraints Constraint 5 guarantees that the number of blocks i that are used to execute jobs is at most the number of available blocks i. Constraint 6 ensures that each large job is fully executed, and constraint 7 guarantees that each small job is fully executed. Constraint 8 ensures non-negativity.

Lemma 8 proves that it is sufficient to consider schedules in which small jobs are executed by a bounded number of blocks. Lemma 9 uses Lemma 8 and shows PTAS-LP is an approximate relaxation for the problem.

▶ **Lemma 8.** *For any schedule with m machines, there exists a schedule with $m(1 + b\lambda)$ machines in which each small job is executed by at most b/λ^2 blocks.*

Proof. Consider any placement P that uses m machines. Suppose a small job j is in more than b/ε^2 blocks in P. Since each configuration is of size at most b, it follows that j is placed in at least $1/\lambda^2$ machines. Since j is small, there exists a configuration σ such that $f_j(\sigma) \geq \lambda d_j$. We remove j from each machine to which it is assigned in P and place it in $1/\lambda$ additional machines, each with configuration σ, guaranteeing that the demand of j is satisfied. Since each machine can hold at most k small jobs, this modification of P results in the increase in the number of machines by a factor of at most $(1 + b\lambda)$, yielding the desired claim. ◀

▶ **Lemma 9.** *The value of PTAS-LP is at most $(1 + b\lambda)$*OPT.

Proof. Let A be an optimal placement of the jobs on m machines. Using Lemma 8, we first compute a new placement B using at most $m(1 + b\lambda)$ machines in which each small job is placed in at most $1/\lambda^2$ machines.

We now define variable assignments so that the value of PTAS-LP is no more than $(1 + b\lambda)m$. For each large job j and each block i, set $x_{i,j}$ to be the number of blocks i on which B executes j. For each small job type t and each pattern π, set $z_{t,\pi}$ to be the number of small jobs that are executed in pattern π according to B. Note that since each small job is placed in at most $1/\lambda^2$ machines, and hence at most b/λ^2 blocks, the placement of each small job follows one of the patterns in W. Set y_σ equal to the number of machines with configuration σ according to A.

Algorithm 5 Schedule for $O(1)$ configurations of $O(1)$ size

```
  Input: $(C, J, k)$
1 Solve PTAS-LP; let $(x, y, z)$ be the solution computed.
2 if $n \le b(|C| + \gamma)/\lambda$ then
3     Compute and return an optimal solution using enumeration
4 foreach large job $j$ and block $i$ do
5     $\widehat{x}_{i,j} = \lceil x_{i,j} \rceil$; Assign $\lceil x_{i,j} \rceil$ blocks $i$ to job $j$
6 foreach job type $t$ and pattern $\pi$ do
7     Assign blocks per pattern $\pi$ to each job in $\lceil z_{t,\pi} \rceil$ small jobs of type $t$
8 foreach configuration $\sigma$ do
9     Use $\lceil y_\sigma \rceil$ machines with configuration $\sigma$
```

It is easy to see that constraints (6 - 8) are satisfied. To see that constraint 5 is satisfied, observe that each machine used by B either has some block executing a large job (in which case it contributes toward the first term of 5) or it has some block executing a small job (in which case it contributes toward the second term). Therefore, the left hand side of 5 counts the total number of blocks needed to complete all the jobs, while the right hand side computes the total number of blocks supplied by the machines. ◀

▶ **Theorem 10.** *Algorithm 5 returns a $(1+\varepsilon)$ approximation to the* CMS *problem in polynomial time if the number of configurations is constant and they are of constant size.*

Proof. First, if $n \le b(|C| + \gamma)/\lambda$, then the algorithm returns an optimal solution. Otherwise, since each machine has at most b blocks, we obtain that $\text{OPT} \ge (|C| + \gamma)/\lambda$. We will show that the number of machines used is at most $(1 + b\lambda)\text{OPT} + \lambda^2 b\text{OPT} + |C| + \gamma$, which is at most $(1 + 2b\lambda)\text{OPT} = (1 + \varepsilon)\text{OPT}$.

Rounding up the x variables increases the number of blocks by at most the number of large jobs times the number of block types. Since each large job requires at least $1/\lambda^2$ machines, this increase in the number of blocks is at most $\lambda^2 b\text{OPT}$. Rounding up the z variables adds at most $1/\lambda^2$ blocks per small job type assigned to a given pattern, increasing the number of blocks by at most γ. Rounding up the y variables increases the number of machines by $|C|$. Taken together with the above increase in the number of blocks, each of which requires at most one machine, the total increase is bounded by $\lambda^2 b\text{OPT} + \gamma + |C|$. By Lemma 9, the LP optimal is at most $(1 + b\lambda)\text{OPT}$, yielding the desired claim.

The linear program PTAS-LP has at most $nk + |C| + \gamma \log \gamma$ variables and $k + n + \gamma$ linear constraints (other than the non-negativity ones), and can be solved in polynomial time. The enumeration for $n \le b(|C| + \gamma)/\lambda$ is constant time, while the rest of the algorithm is linear in the number of variables. The hidden constant, however, is doubly exponential in $|C|$ and the configuration size bound b, and exponential in $1/\varepsilon$. ◀

5 CMS with $O(1)$ number jobs and block types, and all configurations up to a given size

In this section, we consider models for which we are able to obtain optimal solutions in polynomial time. For models with a constant number of jobs, blocks, and configurations, the linear program (1-4) has constant size and so can be solved as an integer program. Many models, however, do not fit this framework and require more sophisticated methods. Consider a setting where the number n of jobs is constant, as well as the number k of block types. We also assume that there exists an integer K such that the set of available configurations C includes all configurations up to size K. For example, with $k = 2$ block types $\{1, 2\}$ and $K = 3$, the set of available configurations $C = \{\{1, 1, 1\}, \{1, 1, 2\}, \{1, 2, 2\}, \{2, 2, 2\}\}$. In this section, we show that a class of problems, including this model, is polynomial time solvable using methods developed by Goemans and Rothvoss [5] to optimally solve bin packing with a constant number of job types.

Consistent with our initial description of the CMS model, we assume all configurations are given as input. However, the input can be reduced exponentially if configurations are given as a single parameter K indicating the size of all valid configurations. The algorithm described in this section is polynomial time in either case – that is, the algorithm's runtime is polynomial in $\log K$ and $\log \max_{i,j}\{d_j/f_j(i)\}$ (assuming n and k are constant). We note that there is an easy reduction from bin packing with constant job types to this problem (when configurations are not listed individually).

Let P be the set of vectors:

$$\begin{bmatrix} u_{1,1} \\ \vdots \\ u_{k,n} \\ v_1 \\ \vdots \\ v_n \\ 1 \end{bmatrix} \quad \text{subject to} \quad \begin{aligned} &\sum_{j=1}^{n}\sum_{i=1}^{b} u_{i,j} \le K && && (9)\\ &v_j = \sum_{i=1}^{k} u_{i,j}\cdot f_j(i) && j \in J && (10)\\ &u_{i,j} \ge 0 && i\in[k],\ j \in J && (11)\\ &v_j \ge 0 && j \in J && (12)\end{aligned}$$

Let Q_H be the set of vectors

$$\begin{bmatrix} z_{1,1} \\ \vdots \\ z_{b,n} \\ w_1 \\ \vdots \\ w_n \\ h \end{bmatrix} \quad \text{subject to,} \quad \begin{aligned} &0 \le h \le H && && (13)\\ &w_j \ge d_j && j \in J && (14)\\ &z_{i,j} \ge 0 && i\in[k],\ j \in J && (15)\end{aligned}$$

▶ **Lemma 11** (Theorem 2.2 in [5]). *Let $N = nk + n + 1$. Given rational polyhedra $P, Q \in \mathbb{R}^N$ where P is bounded, one can find a vector $y \in \text{int.cone}(P \cap \mathbb{Z}^N) \cap Q$ and a vector $\lambda \in \mathbb{Z}_{\ge 0}^{|P\cap\mathbb{Z}^N|}$ such that $y = \sum_{x\in P\cap\mathbb{Z}^N} \lambda_x x$ in time $\text{enc}(P)^{2^{O(N)}} \cdot \text{enc}(Q)$, or decide that no such y exists. Moreover, the support of λ is bounded by 2^{2N+1}.*

▶ **Lemma 12.** *Given P and Q_H, the algorithm of Lemma 11 returns a vector λ iff there exists a solution using at most H machines. Moreover, a returned vector λ provides a solution that uses at most H machines.*

Proof. As in the lemma, let $N = nk + n + 1$. We first show that, if there exists a valid solution using at most H machines then $\text{int.cone}(P \cap \mathbb{Z}^N) \cap Q_H \ne \varnothing$. Suppose there exists a valid solution to the problem using at most H machines. We argue that each machine μ corresponds to an integer valued vector in P. Consider an arbitrary machine μ in the given solution. Recall that $\mu(i, j) =$ the number of blocks of type i on which μ executes job j. This implies that $\sum_j \sum_i \mu(i, j) \le K$. Setting $v_j = \sum_i \mu(i, j) \cdot f_j(i)$ ensures that the vector $(v_1 \dots v_n\ 1\ u_{1,1} \dots u_{b,n}) \in P$. Let p_μ be the vector in P corresponding to machine μ. Let $\lambda = (\lambda_\mu)$, where λ_μ is the multiplicity of machine μ in the given solution. Then $\sum_\mu \lambda_\mu \cdot x_\mu$ yields integer vector $q = (z_{1,1} \dots z_{b,n}\ w_1 \dots w_n\ h)$. Since every job is fully executed, we can infer that $w_j \ge d_j$. Since the solution uses at most H machines, we can infer that $0 \le h \le H$. Therefore, $q \in Q$, which implies that $\text{int.cone}(P \cap \mathbb{Z}^N) \cap Q \ne \varnothing$.

We now show that, if $\text{int.cone}(P \cap \mathbb{Z}^N) \cap Q_H \ne \varnothing$, then there exists a solution to the problem using at most H machines. Suppose there exists a vector $q \in \text{int.cone}(P \cap \mathbb{Z}^N) \cap Q_H$.

Define $\lambda = (\lambda_x)_{x \in P \cap \mathbb{Z}^N}$ such that $\sum_{x \in P \cap \mathbb{Z}^N} x \cdot \lambda_x = q$. As above, we treat each element $x \in P \cap \mathbb{Z}^N$ as corresponding to a machine, and λ_x as the multiplicity of that machine in our final schedule. Similar reasoning shows that λ yields a valid solution, which completes the proof of the lemma. ◀

▶ **Theorem 13.** *Given an instance of* CMS *with $O(1)$ jobs and $O(1)$ block types, and where all configurations of size at most K are available, there exists an algorithm that computes an optimal solution in* $\mathrm{poly}(\log K, \log \max_{i,j}\{d_j/f_j(i)\})$ *time.*

Proof. The maximum number of machines needed in any solution is $m = n \cdot \max_{i,j}\{d_j/f_j(i)\}$. We binary search in the range $[0, m]$ to find the minimum integer value of H for which the algorithm of Lemma 11, given P and Q_H, returns a vector λ. By Lemma 12 this provides an optimal solution. ◀

We have shown the existence of an algorithm that computes an optimal solution to CMS with $O(1)$ number of jobs and block types, and with all configurations up to a given size. However, the Goemans-Rothvoss framework generalizes to a constant number of polytopes (Theorem 6.2 in [5]) which entails the following stronger claim.

▶ **Lemma 14.** *Suppose an instance of* CMS *has a constant number of jobs and blocks, and the set of all configurations can be specified using a set T of rational polytopes with $|T| = O(1)$. Then there exists an algorithm that computes an optimal solution in time polynomial in* $\log K$ *and* $\log \max_{i,j}\{d_j/f_j(i)\}$*).*

To prove Lemma 14, we use the following result from Goemans and Rothvoss.

▶ **Lemma 15** (Theorem 6.2 in [5]). *Given rational polytopes $\{P_t \subseteq \mathbb{R}^{nk+n+1} : t \in T\}$ and rational polyhedron $Q \subseteq \mathbb{R}^{n+1}$, define $X_t := \{x \in \mathbb{Z}^{n+1} : \exists y \in \mathbb{Z}^{nb},\ (x, y) \in P_t\}$. Then there is an algorithm that decides correctly whether* $\mathrm{int.cone}(\bigcup_{t \in T} X_t) \cap Q \neq \varnothing$ *in time* $(\mathrm{enc}(P))^{2^{O(nk+n+1+|T|)}} \cdot \mathrm{enc}(Q)^{O(1)}$. *In the affirmative case, the algorithm provides a vector $\lambda = (\lambda_{t,x})_{t \in T, x \in X}$ with $\lambda_{t,x} \in \mathbb{Z}_{\geq 0}$ and $(\sum_{t \in T} \sum_{x \in X} \lambda_x \cdot x) \in Q$. Moreover, the size of the support of λ is bounded by $2^{2(nk+n+1+|T|)+1}$.*

Proof of Lemma 14. For each $t \in T$, we specify P_t as an $nk + n + 1$ length vector as above, except the polytope is defined by arbitrary rational constraints specific to P_t. Q_H is defined as above. As in Lemma 12, we argue the following claim: given $\{P_t\}_{t \in T}$ and Q_H, the algorithm of Lemma 15 returns a vector λ iff there exists a solution using at most H machines. Moreover, a returned vector λ provides a solution that uses at most H machines. The reasoning is similar to that used in Lemma 12.

Let X_t be defined as in the lemma, for all $t \in T$. If there is a valid solution then for each machine μ there is some t such that μ can be represented as an element p of P_t. Also, we define $\lambda = (\lambda_{t,p})_{t \in T, p \in X}$ such that $\lambda_{t,p}$ provides the multiplicity of the machine corresponding to p in the solution. The fact that we begin with a valid solution on at most H machines ensures that $\sum_{t \in T} \sum_{x \in X} x \cdot \lambda_{t,x} \in Q_H$. In the other direction, we show that if the algorithm returns a vector λ, then λ provides a solution to the problem. As above, we can interpret $\lambda_{t,x}$ as providing the multiplicity of the machine corresponding to $x \in X$ in our solution – since there exists y such that $(x, y) \in P_t$, we can infer that x is a valid machine. By the constraints on Q_H, we can infer that the derived solution completes all jobs and uses at most H machines. This proves the lemma. ◀

References

1 Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 31–40, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1132516.1132522.

2 Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 107–116, 2009. doi:10.1109/FOCS.2009.51.

3 S. W. Cheng and Y. Mao. Restricted max-min allocation: Integrality gap and approximation algorithm. *Algorithmica*, 84:1835–1874, 2022.

4 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 624–633, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591884.

5 Michel X. Goemans and Thomas Rothvoss. Polynomiality for bin packing with a constant number of item types. *J. ACM*, 67(6), nov 2020. doi:10.1145/3421750.

6 Qiang-Sheng Hua, Amy Wang, Dongxiao Yu, and Francis Lau. Dynamic programming based algorithms for set multicover and multiset multicover problem. *Theor. Comput. Sci.*, 411:2467–2474, 06 2010. doi:10.1016/j.tcs.2010.02.016.

7 Zhihao Jiang and Haoyu Zhao. An fptas for stochastic unbounded min-knapsack problem. In Yijia Chen, Xiaotie Deng, and Mei Lu, editors, *Frontiers in Algorithmics*, pages 121–132, Cham, 2019. Springer International Publishing.

8 B. Korte and J. Vygen. *Bin-Packing*, pages 426–441. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. doi:10.1007/3-540-29297-7_18.

9 Jan Karel Lenstra, David B. Shmoys, and Eva Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 217–224, 1987. doi:10.1109/SFCS.1987.8.

10 Baolin Li, Tirthak Patel, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. Miso: exploiting multi-instance gpu capability on multi-tenant gpu clusters. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 173–189, 2022.

11 Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. Clover: Toward sustainable ai with carbon-aware machine learning inference service. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2023.

12 R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, EC '04, page 125–131, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/988772.988792.

13 NVIDIA Multi-Instance GPU User Guide. https://docs.nvidia.com/datacenter/tesla/pdf/NVIDIA_MIG_User_Guide.pdf, 2024.

14 Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. Solving large-scale granular resource allocation problems efficiently with pop. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 521–537, 2021.

15 Sridhar Rajagopalan and Vijay V. Vazirani. Primal-dual rnc approximation algorithms for set cover and covering integer programs. *SIAM J. Comput.*, 28:525–540, 1999. URL: https://api.semanticscholar.org/CorpusID:36747871.

16 Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. Nexus: A gpu cluster engine for accelerating dnn-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 322–337, 2019.

17 Cheng Tan, Zhichao Li, Jian Zhang, Yu Cao, Sikai Qi, Zherui Liu, Yibo Zhu, and Chuanxiong Guo. Serving DNN models with multi-instance GPUs: A case of the reconfigurable machine scheduling problem, 2021. arxiv:2109.11067. arXiv:2109.11067.

18 Vijay V. Vazirani. *Approximation Algorithms*. Springer Publishing Company, Incorporated, 2010.

19 Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. {AntMan}: Dynamic scaling on {GPU} clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 533–548, 2020.

A Constructing Schedules of Polynomial Size

For any schedule S with m machine instances, there exists a multiset of configurations T used in S and, for each job j, a multiset of blocks T_j used in S. Note both $\mathrm{enc}(T)$ and $\mathrm{enc}(T_j)$, for every j, are polynomial in $|C|, |J|$, and k, where $\mathrm{enc}(A)$ denotes the binary encoding length of a multiset A.

▶ **Lemma 16.** *Given a multiset of configurations T and, for each job j, a multiset of blocks T_j, we can output a schedule M such that (i) the number of block instances across all machines instances in M is at least the number of block instances in T, (ii) the multiset of blocks assigned to each job j across all machine instances in M is identical to T_j, and (iii) the description of M has length polynomial in $unique(T) + \sum_j unique(T_j)$, where $unique(A)$ denotes the number of distinct elements in multiset A.*

Proof. We construct M via Algorithm 6. We show that the number of different machines in the schedule produced by Algorithm 6 is polynomial in $|T| + \sum_j |T_j|$. A new machine is constructed in each iteration of the while-loop. In a single iteration of the while-loop, machines are allocated until condition (i) or condition (ii) holds. By the line 10 and 11 updates, the number of times these conditions can be met is at most $\mathrm{unique}(T) + \sum_j \mathrm{unique}(T_j)$. This proves the claim. ◀

Algorithm 6 Multiset-to-Machines.

```
Input: $(T, T_1, \dots, T_j)$
1  for all $\sigma$ : $s_\sigma \leftarrow$ the number of occurrences of $\sigma$ in $T$
2  for all $i, j$ : $t_{i,j} \leftarrow$ the number of occurrences of $i$ in $T_j$
3  while $s_\sigma + \sum_{i,j} t_{i,j} > 0$ do
4      choose any $\sigma \in T$ such that $s_\sigma > 0$
5      construct a new machine $\mu$ with configuration $\sigma$
6      foreach $i \in [k]$ do
7          assign $\sigma_i$ jobs to block $i$ in $\sigma$, ensuring that for every job $j$, $\mu(i,j) \le t_{i,j}$
8      allocate $a$ instances of $\mu$, where $a$ is the minimum value such that
         either (i) $a = s_\sigma$, or (ii) for some $i, j$, $a \cdot \mu(i,j) = t_{i,j}$
9      for all $i, j$ : $t_{i,j} \leftarrow t_{i,j} - a \cdot \mu(i,j)$
10     $s_\sigma \leftarrow s_\sigma - a$
```