Theory of Computing Systems (2025) 69:33
https://doi.org/10.1007/s00224-025-10232-z

Total Completion Time Scheduling Under Scenarios

Thomas Bosman[1] · Martijn van Ee[2] · Ekin Ergen[3] · Csanád Imreh[4] · Alberto Marchetti-Spaccamela[5] · Martin Skutella[3] · Leen Stougie[1,6]

Accepted: 16 July 2025

Abstract
Scheduling jobs with given processing times on identical parallel machines so as to minimize their total completion time is one of the most basic scheduling problems. We study this classical problem under uncertainty, in which the uncertainty is modeled by a set of scenarios. In our model, a scenario is defined as a subset of a predefined and fully specified set of jobs. The aim is to find an assignment of the whole set of jobs to identical parallel machines such that the schedule, obtained for the given scenarios by simply skipping the jobs not in the scenario, optimizes a function of the total completion times over all scenarios. While the underlying scheduling problem without scenarios can be solved efficiently by a simple greedy procedure (SPT rule), scenarios, in general, make the problem NP-hard. We paint an almost complete picture of the evolving complexity landscape, drawing the line between easy and hard. One of our main algorithmic contributions relies on a deep structural result on the maximum imbalance of an optimal schedule, based on a subtle connection to Hilbert bases of a related convex cone.

Keywords Machine scheduling · Total completion time · Scenarios · Complexity

1 Introduction

For a set J of n jobs with given processing times p_j, $j \in J$, one of the oldest results in scheduling theory states that scheduling the jobs in order of non-decreasing processing times on identical parallel machines in a round robin procedure minimizes the total completion time, that is, the sum of completion times of all jobs [1].

Our co-author Csanád Imreh tragically passed away on January 5th, 2017.

Extended author information available on the last page of the article

We study an interesting generalization of this classical scheduling problem where the input in addition specifies a set of K scenarios $\mathcal{S} = \{S_1, \ldots, S_K\}$, with a scenario being a subset of the jobs: $S_k \subseteq J$, $k = 1, \ldots, K$. The task is then to find, for the entire set of jobs J, a parallel machine schedule, which is an assignment of all jobs to machines and on each machine an order of the jobs assigned to it. This naturally induces a schedule for each scenario by simply skipping the jobs not in the scenario. To be precise, jobs not contained in a particular scenario do *not* contribute to the total completion time of that scenario and, in particular, do *not* delay later jobs assigned to the same machine.

We aim to find a schedule for the entire set of jobs that optimizes a function of the total completion times of the jobs over all scenarios. More specifically, we focus on two functions on the scheduling objectives: in the MinMax version, we minimize the maximum total completion time over all scenarios, and in the MinAvg version we minimize the average of the total completion times over all scenarios. In the remainder of the paper we refer to the MinMax version as MinMaxSTC (MinMax Scenario scheduling with Total Completion time objective) and to the MinAvg version as MinAvgSTC.

Optimization under scenarios Scenarios are commonly used in optimization to model uncertainty in the input or different situations that need to be taken into account. A variety of approaches has been proposed that appear in the literature under different names. For instance, scenarios have been introduced to model discrete distributions over parameter values in stochastic programming [2], or as samples in sampling average approximation algorithms for stochastic problems with continuous distributions over parameter values [3]. In robust optimization [4], scenarios describe different situations that should be taken into account and are often specified as ranges for parameter values that may occur. Moreover, in data-driven optimization, scenarios are often obtained as observations. The problems we consider also fit in the general framework of a priori optimization [5]: the schedule for the entire set of jobs can be seen as an a priori solution which is updated in a simple way to a solution for each scenario. In the scheduling literature, different approaches to modeling scenarios have been introduced, for which we refer to a very recent overview by Shabtay and Gilenson [6]. Another related and popular framework is that of min–max regret, aiming at obtaining a solution minimizing the maximum deviation, over all possible scenarios, between the value of the solution and the optimal value of the corresponding scenario [7].

Not surprisingly, for many problems, multiple scenario versions are fundamentally harder than their single scenario counterparts. Examples are the shortest path problem with a scenario specified by the destination [7, 8], and the metric minimum spanning tree problem with a scenario defining the subset of vertices to be in the tree [5]. Scenario versions of NP-hard combinatorial optimization problems were also considered in the literature such as, for example, set cover [9] and the traveling salesperson problem [10].

Related work As we have already discussed above, a variety of approaches to optimization under scenarios appear in the literature under different names. Here we mention work in the field of scheduling that is more closely related to the model considered in this paper. We repeat a reference to the survey [6] and references therein for an overview of scheduling under scenarios.

Closest to the problems considered in this paper is the work of Feuerstein et al. [11] who also consider scenarios given by subsets of jobs and develop approximation algorithms as well as non-approximability results for minimizing the *makespan*, i.e. the time until completion of all jobs, on identical parallel machines, both for the MinMax and the MinAvg version. In fact, the hardness results for our problem given in Proposition 1 below follow directly from their work.

In *multi-scenario models*, a discrete set of scenarios is given, and certain parameters (e.g., processing times) of jobs can have different values in different scenarios. Several papers follow this model, mainly focusing on single machine scheduling problems. Various functions of scheduling objectives over the scenarios are considered, that have the MinMax and the MinAvg versions as special cases. Yang and Yu [12], for example, study a multi-scenario model and show that the MinMax version of minimizing total completion time is NP-hard even on a single machine and with only 2 scenarios, whereas in our model 2-scenario versions are generally easy. (Notice that our model is different from simply assigning a processing time of 0 to a job in a scenario if the job is not present in that scenario.) Aloulou and Della Croce [13] present algorithmic and computational complexity results for several single machine scheduling problems under scenarios. Mastrolilli, Mutsanas, and Svensson [14] consider the MinMax version of minimizing the weighted total completion time on a single machine and prove interesting approximability and inapproximability results. Kasperski and Zieliński [15] consider a more general single machine scheduling problem in which precedence constraints between jobs are present and propose a general framework for solving such problems with various objectives.

Kasperski, Kurpisz, and Zieliński [16] study multi-scenario scheduling with parallel machines and the makespan objective function, where the processing time of each job depends on the scenario; they give approximability results for an unbounded number of machines and a constant number of scenarios. Albers and Janke [17] as well as Bougeret, Jansen, Poss, and Rohwedder [18] study a budgetary model with uncertain job processing times; in this model, each job has a regular processing time while in each scenario up to Γ jobs fail and require additional processing time. The considered objective function is to minimize the makespan: [18] proposes approximate algorithms for identical and unrelated parallel machines while [17] analyses online algorithms in this setting.

Our contribution We give a nearly complete overview of the complexity landscape of total completion time scheduling under scenarios. Tables 1 and 2 summarize our observations for MinMaxSTC and MinAvgSTC, respectively. The rows of both tables correspond to different assumptions on the number of scenarios K, whereas the columns specify assumptions on the given number of machines m.

First of all, it is not difficult to observe that both MinMaxSTC and MinAvgSTC are strongly NP-hard if K can be arbitrarily large; see last row of Tables 1 and 2. This even holds for the special case of unit length jobs and only two jobs per scenario. Moreover, for the case of MinMaxSTC on two machines, we get a tight non-approximability result and corresponding approximation algorithm, while for MinAvgSTC we can prove that the

Table 1 The complexity landscape of MinMaxSTC on *m* machines with *K* scenarios

	$m = 2$	$m \in O(1)$	*m* part of input
$K = 2$	poly	poly	poly
$3 \leq K \in O(1)$	weakly NP-hard, pseudo-poly, FPTAS	weakly NP-hard, pseudo-poly, FPTAS	weakly NP-hard, poly if $p_j \in O(1)$
K part of input	strongly NP-hard [11], no $(2 - \varepsilon)$-approx [11], 2-approx	strongly NP-hard [11]	strongly NP-hard [11]

Table 2 The complexity landscape of MinAvgSTC on *m* machines with *K* scenarios

	$m = 2$	$m \in O(1)$	*m* part of input
$K = 2$	poly	poly	poly
$3 \leq K \in O(1)$	poly	poly	poly if $p_j \equiv 1$[1]
K part of input	strongly NP-hard [11], no PTAS [19, 20], $5/4$-approx [21, 22]	strongly NP-hard [11], no PTAS [19, 20], $(3/2 - 1/2m)$-approx [21, 22]	strongly NP-hard [11], no PTAS [19, 20], $(3/2 - 1/2m)$-approx [21, 22]

[1]This result can be generalized to a constant number of distinct weights. We omit the details and provide a brief explanation

problem is APX-hard, i.e., there is no PTAS, unless P=NP; see Section 3. For only $K = 2$ scenarios, however, both problems can be solved to optimality in polynomial time; see first row of Tables 1 and 2. Even better, in Section 3 we present a simple algorithm that constructs an 'ideal' schedule for the entire set of jobs simultaneously minimizing the total completion time in both scenarios. These results develop a clear complexity gap between the case of two and arbitrarily many scenarios.

A finer distinction between easy and hard can thus be achieved by considering the case of constantly many scenarios $K \geq 3$; see middle row in Tables 1 and 2. These results constitute the main contribution of this paper.

Our results on MinMaxSTC for constantly many scenarios $K \geq 3$ are presented in Section 4. Here it turns out that MinMaxSTC is weakly NP-hard already for $K = 3$ scenarios and $m = 2$ machines, but can be solved in pseudo-polynomial time for any constant number of scenarios and machines via dynamic programming. Moreover, the dynamic program together with standard rounding techniques immediately implies the existence of an FPTAS for this case. If the number of machines *m* is part of the input, however, our previous dynamic programming approach fails. If all job processing times are bounded by a constant, this issue can be circumvented by another dynamic program.

MinAvgSTC with constantly many scenarios $K \geq 3$ is studied in Section 5. Somewhat surprisingly, and in contrast to MinMaxSTC, it turns out that MinAvgSTC remains easy as long as the number of machines *m* is bounded by a constant. This observation is again based on a dynamic programming algorithm. Moreover, we conjecture that the problem even remains easy if *m* is part of the input. More precisely, we conjecture that there always exists an optimal solution such that the imbalance between machine loads in each scenario remains bounded by $g(K)$ for some (exponential) function *g* that only depends on the number of scenarios K, but not on *m* or *n*. Using a subtle connection to the cardinality of Hilbert bases for convex cones, we prove that our conjecture is true for instances with unit job weights. In this case we obtain an efficient algorithm with running time $m^{h(K)} \cdot poly(n)$ for some function *h*.

It is also possible to generalize this result to instances with a constant number of distinct job weights; e.g., where the job weights are bounded by a constant.

2 Preliminaries

We start by defining the problem of minimizing the sum of completion times with scenarios when jobs have varying processing times and unit weights. We will then argue why this is equivalent to the jobs having unit processing times and varying weights. The latter facilitates the proofs of our results substantially.

In the sequel, we denote the set of integers $\{1, \ldots, \ell\}$ simply by $[\ell]$. An instance of the total completion time scheduling problem under scenarios is defined by a triple $(J, M, \mathcal{S})$, where $J = [n]$ is a set of jobs with non-negative job processing times $p_j, j \in [n]$, a set of parallel identical machines $M = [m]$, and a set of scenarios $\mathcal{S} = \{S_1, \ldots, S_K\}$, where $S_k \subseteq J, k \in [K]$. At the expense of an additional $O(n \log n)$ in running time, we assume that the jobs are ordered by *non-increasing* processing times $p_1 \geq p_2 \geq \cdots \geq p_n$.[1]

[1] In view of the SPT rule, this ordering might seem counterintuitive. But it turns out to be convenient as we argue below in Remark 1.

The task is to find a machine assignment, that is, a map $\varphi : [n] \to [m]$, or equivalently, a partitioning of the jobs $J_1, \ldots, J_m$ with the understanding that jobs in J_i shall be optimally scheduled on machine $i \in [m]$, that is, according to the *Shortest Processing Time first* (SPT) rule (i.e., in reverse order of their indices). Thus, the completion time of a particular job $j' \in J_i$ in scenario *k* is the sum over all processing times of jobs $j \in J_i \cap S_k$ with $j \geq j'$; it follows that the contribution of jobs in J_i to the total completion in scenario $k \in [K]$ is:

$$\sum_{j' \in J_i \cap S_k} \sum_{j \in J_i \cap S_k : j' \leq j} p_j = \sum_{j \in J_i \cap S_k} p_j \cdot |\{j' \in J_i \cap S_k : j' \leq j\}|$$

The problem of minimizing the sum of completion times is a special case of the problem of minimizing the weighted sum of completion times; in the weighted version each job has a processing time p_j and a weight w_j and the objective is to minimize $\sum_{j \in J} w_j C_j$ where C_j denotes the completion time of job *j*. On a single machine the shortest weighted processing time (Smith rule), in which jobs are sequenced in non-decreasing order of p_j/w_j, provides an optimal solution. On the other hand, the problem is NP-hard even in the case of two identical machines [23].

In the sequel of our paper we will use the following equivalence between two special case of the weighted sum of completion times.

Remark 1 ([24]) The problem of minimizing the (unweighted) sum of completion times of a set of jobs with varying processing times and unit weights is equivalent to the problem of minimizing the total weighted completion time of jobs with unit processing times and weights $w_j := p_j$.

Indeed, this equivalence, first observed by Eastman, Even, and Issacs [24], is based on the following observations: *i)* on identical parallel machines, unweighted jobs with varying processing times are optimally scheduled using the SPT rule, and *ii)* on identical parallel machines, weighted jobs with unitary processing times are optimally scheduled in order of non-increasing weights.

The idea behind the equivalence is best seen from a so-called 2-dimensional Gantt-chart; see Fig. 1 and the work of Goemans and Williamson [25], Megow and Verschae [26], or Cho, Shmoys, and Henderson [27].

In the weighted version the objective of MinMaxSTC is then to minimize

$$\max_{k \in [K]} \sum_{i=1}^{m} \sum_{j \in J_i \cap S_k} w_j \cdot |\{j' \in J_i \cap S_k : j' \leq j\}|, \tag{1}$$

whereas MinAvgSTC aims to minimize

$$\frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{m} \sum_{j \in J_i \cap S_k} w_j \cdot |\{j' \in J_i \cap S_k : j' \leq j\}|. \tag{2}$$

By the equivalence we notice that the results in the Tables 1 and 2 hold as well for the weighted version with unit processing times (just replace the occurrence of p_j in the tables by w_j).

In order to proving our results, we will switch from here on completely to the weighted version with unit processing times. In the sequel, we further neglect the constant $1/K$-term in the MinAvgSTC and minimize the sum over all scenarios.

Finally, instead of viewing our instances in light of scenario sets S_k, $k \in [K]$, an alternative approach which will prove useful for many arguments is to consider the jobs using their *type*s, where, we define the type T of a job $j \in [n]$ as the index set of the scenarios in which the job appears, that is, $T = \{k \in [K] : j \in S_k\}$.

3 Arbitrary K is Hard, but $K = 2$ is Easy

We start describing the complexity landscape of our problem by varying the number of scenarios. Since our problem generalizes its well-known case for $K = 1$, which is solvable in polynomial time by the SPT rule, it is of interest whether the problem becomes hard for varying the number K of the scenarios. In the sequel, we answer this question affirmatively for both objective functions. In contrast, we also demonstrate that the case $K = 2$ is solvable in polynomial time for both objective functions. These two findings set the scene for the main question we find ourselves asking in the subsequent sections: While increasing the number K of scenarios, when does the problem become hard?

3.1 NP-hardness for an Unbounded Number of Scenarios

For an unbounded number of scenarios, there is a straightforward proof that both MinMaxSTC and MinAvgSTC are NP-hard on $m \geq 3$ machines which relies on a simple reduction from the graph coloring problem: Given a graph, we interpret its nodes as unweighted unit length jobs and every edge as one scenario consisting of the two jobs that correspond to the end nodes of the edge. Obviously, the graph has an m-coloring without monochromatic edges if and only if there is a schedule such that the total completion time of each scenario is 2 (i.e., both jobs complete at time 1 on a machine of their own).

Since it is easy to decide whether the nodes of a given graph can be colored with two colors, the above reduction does not imply NP-hardness for $m = 2$ machines. For this case, however, the inapproximability results for the multi scenario makespan problem in [11] can be adapted to similar inapproximability results for our problems. These results also already hold for unweighted unit length jobs.

Proposition 1 *For two machines and all jobs having unit lengths and weights, it is NP-hard to approximate MinMaxSTC within a factor $2 - \varepsilon$ and MinAvgSTC within ratio 1.011. The latter even holds if all scenarios contain only two jobs.*

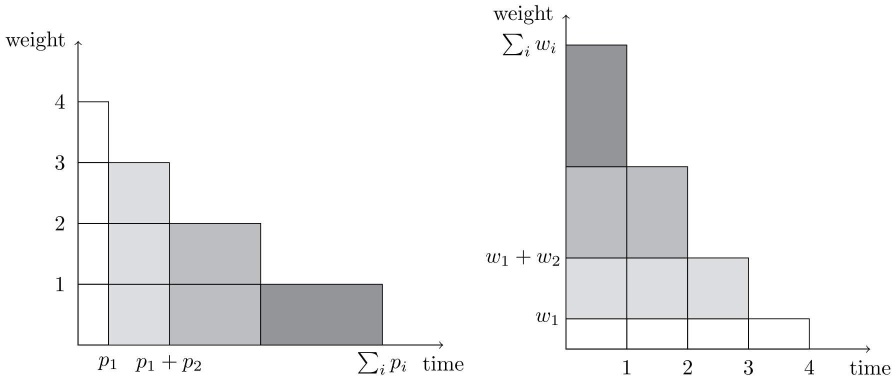


Fig. 1 Left: original schedule. Right: equivalent “weight-schedule”. In both cases the objective value is equal to the total area of the rectangles

Proof For MinMaxSTC we use essentially the same reduction as in the proof of Theorem 1 in [11] which we present below.

We reduce the hypergraph balancing problem to MinMaxSTC. The variant of the hypergraph balancing problem that we use, proven to be NP-hard in [28], is given as follows:

Instance: A $(2\ell + 1)$-regular hypergraph that can be 2-colored in a near-balanced way, i.e., which admits a coloring $c : V(H) \to \{1, 2\}$ with the property that for each hyperedge each colour is used to color either ℓ or $\ell + 1$ vertices; formally

$$|e \cap c^{-1}(i)| \in \{\ell, \ell + 1\}$$

for all $e \in E(H)$ and $i \in \{1, 2\}$.

Task: Find a 2-coloring without a monochromatic hyperedge, i.e., $c_0 : V(H) \to \{1, 2\}$ with the property that $e \cap c_0^{-1}(i) \neq \emptyset$ for all $e \in E(H)$ and $i \in \{1, 2\}$.

Given a hypergraph H that is an instance of the hypergraph balancing problem, we observe a one-to-one correspondence with unit-weight instances of MinMax-STC, where the jobs are given by $V(H)$, the scenarios correspond precisely to hyper-edges seen as vertex subsets and $m = 2$. Then, the 2-vertex coloring c of H that is assumed to exist accounts to the fact that there is a distribution of the jobs onto the two machines such that in each scenario, one machine is assigned $\ell + 1$ jobs and the other machine is assigned ℓ jobs. This means that the sum of completion times is $\sum_{i=1}^{\ell} i + \sum_{i=1}^{\ell+1} i = (\ell + 1)^2$ in every scenario, since jobs have unit processing times and unit weights. In other words, if we can solve MinMaxSTC in polynomial time, a schedule will be output with at most the objective value of $(\ell + 1)^2$.

We now argue such a schedule corresponds to a hypergraph without a monochromatic hyperedge, meaning that we can solve the hypergraph balancing problem.

For the sake of a contradiction, assume not, i.e., there is a monochromatic hyperedge in the corresponding hypergraph H of the output. This monochromatic hyperedge corresponds to a scenario S_k of size $2\ell + 1$ for the supposedly optimal MinMaxSTC solution, in which all jobs have been assigned to the same machine. For the scenario *k*, the sum of completion times is $\sum_{i=1}^{2\ell+1} i = (2\ell+1)(\ell+1) > (\ell+1)^2$, contradicting the optimality of the schedule.

In order to show the inapproximability, let $\varepsilon > 0$ be a fixed but arbitrary constant. A $(2-\varepsilon)$-approximation of MinMaxSTC has an objective value $(2-\varepsilon)(\ell+1)^2 < (2\ell+1)(\ell+1)$ for $\ell > \frac{1}{\varepsilon} - 1$ and therefore corresponds to a hypergraph without a monochromatic hyperedge, similar to above. Since $\varepsilon > 0$ is a constant, instances with $\ell \leq \frac{1}{\varepsilon} - 1$ can be solved in constant time.

For proving the NP-hardness of MinAvgSTC, we adapt the reduction in the proof of Theorem 6 in [11] to the total completion time objective.

Consider a Max Cut instance and assign to each vertex a job with weight 1 and for each edge a scenario. Then a cut gives a partition of the vertices, and accordingly, the jobs. If the scenario is in the cut then the sum of completion times of both jobs is $1 + 1 = 2$; if it is not in the cut, then the two jobs of the scenario are assigned to the same machine and the sum of completion times is $1 + 2 = 3$. In total, the objective value of the MinAvgSTC instance is 3 times the number of edges minus the size of the corresponding cut. In particular, since the number of edges is a constant, the minimum-cost schedule corresponds to the maximum cut.

Let us assume that a $(1+\alpha)$-approximation algorithm for MinAvgSTC exists for a suitable $\alpha > 0$. which outputs a schedule of total completion time d_{alg}. Then, the corresponding cut has a value of $d_{alg} - 3K$, where K is the number of edges (or equivalently, scenarios of the schedule). Let s_{alg} and s_{opt} denote the sizes of the cuts that correspond to the output of the approximation algorithm and the optimal solution of MinAvgSTC, respectively. Then, it holds that

$$(1+\alpha)(3K - s_{opt}) = (1+\alpha)d_{opt} \geq d_{alg} = 3K - s_{alg},$$

and rearranging the terms, we obtain

$$\Rightarrow s_{alg} \geq -3\alpha K + (1+\alpha)s_{opt} \geq -6\alpha s_{opt} + (1+\alpha)s_{opt} = (1-5\alpha)s_{opt}.$$

Here, the last inequality holds because $s_{opt} \geq \frac{K}{2}$, as every graph has a cut that includes at least half of its edges (to see this, one may consider that, in a uniformly random cut, every edge is included with a probability of $\frac{1}{2}$). By this observation it follows that a $(1+\alpha)$-approximation for MinAvgSTC yields a $(1-5\alpha)$-approximation for Max Cut and our results follow from the known inapproximability of Max Cut [19, 20]. □

We notice that, for $m = 2$ machines, any algorithm for MinMaxSTC that assigns all jobs to the same machine (in SPT order) gives a 2-approximation. The approximability of MinMaxSTC for more than two machines is left as an interesting open question.

The following approximation result for MinAvgSTC follows from a corresponding result for classical machine scheduling without scenarios.

Proposition 2 *For MinAvgSTC there is a* $(3/2 - 1/2m)$*-approximation algorithm for arbitrarily number of scenarios and aribitrarily many machines, even if m is part of the input.*

Proof The result is a consequence of the following approximation result for the total weighted machine scheduling problem without scenarios [21, 22]. Let J be a set of jobs to be scheduled on m machines; if all jobs in J are assigned to machines independently and uniformly at random, then the expected total weighted completion time of the resulting schedule is at most a factor $3/2 - 1/2m$ away from the optimum.

Note that, in our scenario scheduling model, this upper bound holds for each single scenario; therefore, by linearity of expectation, it yields a randomized $(3/2 - 1/2m)$ -approximation algorithm for MinAvgSTC. We also observe that this randomized algorithm can be de-randomized using standard techniques. □

3.2 Computing an Ideal Schedule for Two Scenarios

For $K = 2$ scenarios, both MinMaxSTC and MinAvgSTC are polynomial time solvable on any number of machines. Actually, we prove an even stronger result: one can find, in polynomial time, a schedule that has optimal objective function value in each of the two scenarios simultaneously.

Theorem 3 Let an instance of the total completion time scheduling under scenarios problem be given by two scenarios, n jobs and an arbitrary number of machines. After sorting the jobs in order of non-increasing weight, one can find in time linear in n a schedule that is simultaneously optimal for both scenarios, in particular for the objective functions MinMaxSTC and MinAvgSTC.

Proof We show how to assign the jobs in *non-increasing* order of their weights in order to be optimal for both scenarios. We would like to assign the next job to a machine that, in each scenario, belongs to a least loaded machine in terms of the number of jobs already assigned to it. However such a machine need not exist, as the sets of least-loaded jobs in both scenarios do not have to coincide. To circumvent this issue, we pack jobs which appear in one scenario to certain machines in case of a tie between machines, and jobs which appear in both scenarios to others, so as to keep each machine as balanced as possible. Then, we show that our method of doing so indeed ensures that the least loaded machines agree in both scenarios.

For this purpose, we define two m-dimensional vectors s_1 and s_2 for scenarios 1 and 2 respectively, containing the *relative* loads on the machines. The relative load of a machine in a scenario is 0 if this machine belongs to the least loaded machines in this scenario; it is 1 if it has one job more than a least loaded machine, etc. In our assignment process, jobs will always be assigned to machines with relative load 0 in each of the scenarios they belong to. This ensures that we will end up with a schedule that is optimal for both scenarios simultaneously, because restricted to each scenario, loading jobs onto a least loaded machine every time leads to an optimal solution according to the SPT rule.

Initially, both vectors s_1 and s_2 are zero vectors, since no jobs have been assigned yet. When assigning job j, let μ_k be the lowest entry (lowest numbered machine) equal to zero in the vector s_k ($k \in \{1,2\}$). Moreover, let ν_k ($k \in \{1,2\}$) be the highest entry equal to zero in s_k. So initially, $\mu_1 = \mu_2 = 1$ and $\nu_1 = \nu_2 = m$. We apply the following assignment procedure, where we use $\mathbb{1}$ to denote the all-1 vector. For $j = 1$ to n, we apply the following case distinction:

- If $j \in S_1 \setminus S_2 \rightarrow$ assign j to machine μ_1 and increase μ_1 by 1.
- If $j \in S_2 \setminus S_1 \rightarrow$ assign j to machine μ_2 and increase μ_2 by 1.
- If $j \in S_1 \cap S_2 \rightarrow$ assign j to machine $\nu_1 = \nu_2$ and decrease both ν_1 and ν_2 by 1.
- If $s_1 = \mathbb{1} \rightarrow$ reset s_1 to become the all-0 vector. Reindex the machines such that s_2 becomes of the form $(1, \ldots, 1, 0, \ldots, 0)$. Reset $\mu_1 = 1$, $\nu_1 = m$, $\mu_2 = \mu_2 + m - \nu_2$, and $\nu_2 = m$. Do analogously if $s_2 = \mathbb{1}$.

This accounts to assigning every job $j \in S_1 \triangle S_2 = (S_1 \setminus S_2) \cup (S_2 \setminus S_1)$ to a lowest-index least loaded machine and every job $j \in S_1 \cap S_2$ to a highest-index least loaded machine.

We prove that for each job there is always a machine with relative load 0 in each scenario in which the job appears, thus implying the theorem. This is obviously true if job j appears in only one scenario, since after assigning $j-1$ jobs, $s_k \neq \mathbb{1}$, $k = 1, 2$. Hence there is always a machine with relative load 0. For job j appearing in both scenarios, we have to show that we maintain $\nu_1 = \nu_2$, in which case the same machine has relative load 0 to accommodate job j. The only way it can happen that $\nu_1 \neq \nu_2$ is if ever machine ν_1 was used for a job j' that only appeared in scenario 1. But that can only have happened if $\mu_1 = \nu_1$, in which case s_1 becomes an all-1 vector, and by resetting it to 0 and the renumbering of the machines, the relation $\nu_1 = \nu_2$ had been restored. □

The reasoning used in the above proof can no longer be employed if $K \geq 3$. Indeed, it is not necessarily possible to assign a job j to a machine simultaneously least loaded in every scenario k with $j \in S_k$. Consider the instance with $n = 3$ jobs and $m = 2$ machines, given by unit weights $w_j \equiv 1$ and scenarios $S_1 = \{2,3\}$, $S_2 = \{1,3\}$, $S_3 = \{1,2\}$. As the scenarios are symmetric, we may suppose that the jobs are to be scheduled in increasing order of their indices. The first job is assigned to the first machine without loss of generality. Then, job 2 must be assigned to machine 2, as it is the unique least loaded machine with respect to the first scenario S_1. The third job, on the other hand, cannot be assigned anymore: This job lies in $S_1 \cap S_2$. However, machine i is the unique least loaded machine with respect to scenario S_i ($i \in \{1,2\}$).

The above example can be generalized to any number $m > 2$ of machines by extending the instance with $m - 2$ jobs $j \in S_1 \cap S_2 \cap S_3$ of weight 1. In Section 4, the example as well as this extension will be utilized as a building block while proving the NP-hardness of the variant $K \geq 3$.

4 The MinMax Version

If job weights are bounded by a constant, MinMaxSTC (and also MinAvgSTC) can be solved efficiently on any number of machines by dynamic programming. For simplicity, we only discuss the case of unit job weights here, but our approach can easily be generalized to the case of weights bounded by some constant. The dynamic program leading to the following theorem is based on enumeration of machine configurations.

Theorem 4 If the number of scenarios K is constant, and all jobs have unit weights (next to unit processing times), then MinMaxSTC and MinAvgSTC can be solved to optimality in polynomial time on any number of machines.

Proof We show how to efficiently solve MinAvgSTC and MinMaxSTC by dynamic programming. In the following we encode schedules by tuples (m', π, d) with m' a guessed number of machines, $\pi = (\pi_T)_{T \subseteq [K]}$, where π_T is a guessed number of jobs of type *T* and $d = (d_k)_{k \in [K]}$, where d_k is a guessed total completion time in scenario *k*. Denoting by n_T the total number of jobs of type *T* in the instance,

$$\begin{aligned} m' &\in \{0, 1, \ldots, m\} && \\ \pi_T &\in \{0, 1, \ldots, n_T\} && \text{for every} \quad T \subseteq [K], \\ d_k &\in \{0, 1, \ldots, \tfrac{1}{2}n(n+1)\} && \text{for every} \quad k \in [K]. \end{aligned}$$

Notice that the number of such tuples is

$$(m+1)\left(\prod_{T \subseteq [K]} (n_T + 1)\right)\left(\tfrac{1}{2}n(n+1)\right)^K,$$

which is polynomial in the input size.

For each tuple (m', π, d) define $A(m', \pi, d)$ to be *true* if there is an assignment of π_T jobs of type *T*, for every $T \subseteq [K]$, to m' machines such that the total completion time in every scenario $k \in [K]$ is d_k; otherwise, $A(m', \pi, d)$ is *false*. It thus holds that

$$A(0, \pi, d) = \begin{cases} \text{true} & \text{if } \pi_T = 0 \text{ for every } T \text{ and } d_k = 0 \text{ for every } \ k, \\ \text{false} & \text{otherwise.} \end{cases} \tag{3}$$

Let $Q := \mathbb{Z}_{\geq 0}^{\mathcal{P}([K])}$, i.e., the set of vectors with positive integer entries indexed by the subsets of [*K*]. Its elements $q \in Q$ are to be interpreted as the *configuration* of a single machine, that is, the number of jobs assigned to the machine indexed by their types. To be more precise, for $q \in Q$, $q_T = s$ shall account to the fact that *s* jobs of type *T* exist. Let $c_q \in \mathbb{Z}_{\geq 0}^K$ be given by the total completion times of the schedule consisting of packing jobs corresponding to the vector *q* onto a single machine, indexed by the scenarios $k \in K$. In other words, the entries of the vector c_q are given by

$$(c_q)_k = \frac{1}{2}\left(\sum_{T \subseteq K : k \in T} q_T\right)\left(1 + \sum_{T \subseteq K : k \in T} q_T\right).$$

Then, for $1 \leq m' \leq m$,

$$A(m', \pi, d) = \bigvee_{q \in Q : q \leq \pi} A(m' - 1, \pi - q, d - c_q); \tag{4}$$

here we set $A(m' - 1, \pi - q, d - c_q)$ to *false* if $d - c_q < 0$. and (4), all values $A(m', \pi, d)$ can be computed in polynomial time.

The value of an optimum solution to MinMaxSTC can now be determined by finding a tuple (*m*, *n*, *d*) with $A(m, n, d) = \text{true}$ and $\max_{k \in [K]} d_k$ minimum. Similarly, the value of an optimum solution to MinAvgSTC can be determined by finding a tuple (*m*, *n*, *d*) with $A(m, n, d) = \text{true}$ and $\sum_{k \in [K]} d_k$ minimum. Corresponding optimal machine assignments can be found by reverse engineering. □

In contrast to the tractability of the unit weight case, MinMaxSTC becomes NP-hard with general weights already on two machines and with three scenarios. Indeed, the proof of the following theorem relies on a reduction that uses weights in different orders of magnitude. In view of Theorem 3, the hardness establishes a complexity jump for MinMaxSTC when going from two to three scenarios.

Theorem 5 On any fixed number $m \geq 2$ of machines and with $K = 3$ scenarios, MinMaxSTC is (weakly) NP-hard.

Proof We reduce Partition-3 (Partition into 3 subsets instead of 2) to MinMaxSTC. Let $\{a_1, \ldots, a_n\}$ be an instance of Partition-3, i.e., we are looking for a partition $A_1 \dot\cup A_2 \dot\cup A_3 = [n]$ with $\sum_{j \in A_1} a_j = \sum_{j \in A_2} a_j = \sum_{j \in A_3} a_j = \frac{1}{3} \sum_{j=1}^{n} a_j$.

We create $(2m + 2)n$ jobs, grouping them in (disjoint) subsets $F_1, \ldots, F_n$ of size $2m + 2$ each. For $j = 1, \ldots, n$, we define the weights of the jobs in F_j as follows:

- Three of the jobs have weight Q_j, where $(Q_j)_{j \in \{1,\ldots,n\}}$ is a decreasing sequence of sufficiently large numbers to be determined later. These jobs appear in scenarios $\{1, 2\}$, $\{2, 3\}$ and $\{1, 3\}$ respectively. We shall refer to these jobs as *white*, and depict them in figures accordingly.
- Three of the jobs have weight $Q_j + a_j$. These jobs also appear in scenarios

$\{1,2\}$, $\{2,3\}$ and $\{1,3\}$ respectively. We refer to these jobs as *black* and also depict them in figures accordingly.
- The remaining $2(m-2)$ jobs appear in all three scenarios and have weight Q_j. We refer to them as the *gray* jobs.

In particular, there are no gray jobs for $m = 2$. Before determining Q_j, we formulate the assumptions that are needed for the reduction.

1. For $j = 1, \dots, n-1$, jobs in F_j are executed before those in F_{j+1}.
2. For each scenario, any optimal solution schedules two jobs from each set F_j ($j = 1, \dots, n$) on each machine, i.e., for any optimal partitioning $J_1, \dots, J_m$ of the jobs, we have $|J_i \cap F_j \cap S_k| = 2$ for every $i \in [m], j \in [n], k \in [K]$. $\square$

Claim 1 For $Q_l > 4mn^2 a_{\max} + \sum_{j=l+1}^{n} m(4j-1)Q_j$, the above assumptions are satisfied.

Proof For the first condition to be fulfilled, we merely need $Q_j > Q_{j+1} + a_{j+1}$ for all j.

For the second, note that we have $|S_k \cap F_j| = 2m$ for each $k \in \{1,2,3\}$ and $j \in \{1, \dots, n\}$ (two white, two black, $2(m-2)$ gray), which is a necessary condition for the second assumption. Now it suffices to prove that any solution satisfying the second property has strictly less weight than one that does not.

Any schedule that satisfies the second property places jobs from F_j to have completion times $2j-1$ and $2j$. Since the weights of the jobs in F_j are upper bounded by $Q_j + a_j$, such solutions cost at most

$$m \cdot \sum_{j=1}^{n} ((2j-1) + 2j)(Q_j + a_j) \leq \sum_{j=1}^{n} m(4j-1)(Q_j + a_{\max}). \tag{5}$$

Now consider a schedule that does not satisfy the second property and let ℓ be the first index where it is violated, i.e., in every scenario, all machines are assigned two jobs from F_j each for $j < \ell$; and there exists a scenario k where a machine m' is assigned at least three jobs from F_ℓ, i.e., $|F_\ell \cap S_k \cap J_{m'}| \geq 3$. For this scenario, the machine m' contributes to the weight by at least

$$\sum_{j=1}^{\ell-1} ((2j-1) + 2j)Q_j + (2\ell - 1)Q_\ell + 2\ell Q_\ell + (2\ell + 1)Q_\ell.$$

Here, we take the sum only over jobs in $F_1 \cup \ldots \cup F_\ell$. The last summand corresponds to the third job from F_ℓ that has completion time $2\ell + 1$ by the minimality of ℓ. Similarly, the contribution of the other machines and jobs in $F_1 \cup \ldots \cup F_{\ell-1}$

is at least $(m-1)\cdot\sum_{j=1}^{\ell-1}((2j-1)+2j)Q_j$, so that the contribution of the jobs in $F_1\cup\ldots\cup F_\ell$ add up to

$$m\cdot\sum_{j=1}^{\ell-1}((2j-1)+2j)Q_j+(2\ell-1)Q_\ell+2\ell Q_\ell+(2\ell+1)Q_\ell \tag{6}$$

$$=\sum_{j=1}^{\ell-1}m(4j-1)Q_j+6\ell Q_\ell. \tag{7}$$

On the other hand, we know that $|S_k\cap F_\ell|=2m$, i.e., there are $2m-3$ other jobs in $S_k\cap F_\ell$. At most $m-1$ of these jobs can be completed at time $2\ell-1$ because there are only $m-1$ other machines. The remaining $m-2$ jobs are completed at time at least 2ℓ. These two cases together contribute at least another

$$\big((m-1)(2\ell-1)+(m-2)2\ell\big)Q_\ell=(4m\ell-6\ell-m+1)Q_\ell \tag{8}$$

to the weight. Considering the contribution of $F_1\cup\ldots\cup F_{\ell-1}\cup(F_j\cup J_{m'})$ (lower bounded by (7)) as well as that of the remaining jobs (lower bounded by (8)), the weight of the violating schedule is at least

$$\sum_{j=1}^{\ell-1}m(4j-1)Q_j+\big((4m\ell-6\ell-m+1)+6\ell\big)Q_\ell \tag{9}$$

$$=\sum_{j=1}^{\ell}m(4j-1)Q_j+Q_\ell \tag{10}$$

$$>\sum_{j=1}^{\ell}m(4j-1)Q_j+4mn^2a_{\max}+\sum_{j=\ell+1}^{n}m(4j-1)Q_j \tag{11}$$

$$>\sum_{j=1}^{n}m(4j-1)(Q_j+a_{\max}). \tag{12}$$

Comparing (5) and (12) yields a contradiction, finishing the proof of the claim.

The exact values of the Q_j are not relevant for the proof. We are rather interested in their existence, which is evident given Claim 1. One can further verify that e.g. the sequence $Q_j=(4mna_{\max})^{n-j}$ satisfies the inequality in Claim 1, meaning that the numbers Q_j can be chosen polynomially large in size of the input of Partition-3.

Since each job has unit processing time, the subsets F_ℓ correspond to a decomposition of the schedule into what we refer to as *blocks*, in which elements of F_ℓ are completed in time $2\ell-1$ resp. 2ℓ in any scenario. Therefore the contribution of the elements of F_ℓ to the sum of completion times is independent of those in $F_{\ell'}$ with $\ell' \neq \ell$.

Before continuing with the reduction, we make some observations about the $2m-4$ gray jobs. By the claim above, there is no optimal solution where these jobs are assigned to the same machine as a non-gray job, for if this were the case, then there would exist a scenario where either one job or three jobs would be executed by this machine. Hence without loss of generality, we may assume that gray jobs are assigned to machines 3 to *m*. In any optimal solution, these machines then contribute the same amount of weight, which we neglect from now on.

Let us fix a set F_ℓ of our partitioning of the jobs and consider the first two machines where no gray jobs are scheduled: For any assignment of the jobs, there is one scenario where two black jobs appearing in that scenario are assigned to the same machine. This is because two of the three black jobs are assigned to the same machine, and there exists one scenario where both these jobs are included. Given that the black jobs cost more than the white ones and would ideally be completed at time $2\ell-1$, this creates an excess weight in this particular scenario. More precisely, let without loss of generality black jobs appearing in scenario S_1 both be assigned to the first machine. In this case, the ℓ-th block contributes

$$(2\ell-1)\left(Q_\ell+a_\ell\right)+2\ell\left(Q_\ell+a_\ell\right)+(2\ell-1)Q_\ell+2\ell Q_\ell=(8\ell-2)Q_\ell+(4\ell-1)a_\ell.$$

In the scenarios S_2 and S_3 this contribution is

$$2(2\ell-1)\left(Q_\ell+a_\ell\right)+2\cdot 2\ell Q_\ell=(8\ell-2)Q_\ell+(4\ell-2)a_\ell.$$

Note that the blocks in the scenarios S_2 and S_3 are optimal and the scenario S_1 costs a_ℓ more than these scenarios. The three scenarios are shown Fig. 2.

For $k \in \{1,2,3\}$ let A_k be the set of blocks ℓ that cost more in the *k*-th scenario. For instance, the block in Fig. 2 is an element of A_1. We immediately observe that $A_1 \dot\cup A_2 \dot\cup A_3 = \{1,\dots,n\}$. Let c_j denote the contribution of the *j*-th job in the objective function and $c(J') := \sum_{j\in J'} c_j$ given $J' \subseteq [n]$. Then, we have

$$c(S_k)=\sum_{\ell=1}^{n} c(F_\ell\cap S_k)=\sum_{\ell=1}^{n}((8\ell-2)Q_\ell+(4\ell-2)a_\ell)+\sum_{\ell\in A_k} a_\ell.$$

Since $c(S_1)+c(S_2)+c(S_3)=3\sum_{\ell=1}^{n}((8\ell-2)Q_\ell+(4\ell-2)a_\ell)+\sum_{\ell=1}^{n} a_\ell$, we have

$$\max_{k\in\{1,2,3\}} c(S_k)\geq\sum_{\ell=1}^{n}((8\ell-2)Q_\ell+(4\ell-2)a_\ell)+\frac{1}{3}\sum_{\ell=1}^{n} a_\ell.$$

This inequality is tight if and only if there is a partition of the indices $1,\dots,n$ of the job subsets $F_1,\dots,F_\ell$ into the three sets A_1, A_2, A_3 such that

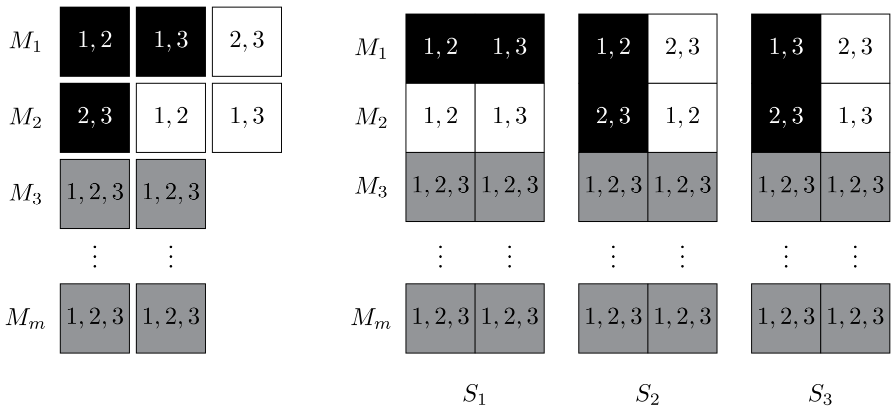


Fig. 2 Left: The assignment to machines described above. Right: The blocks for scenarios S_1, S_2 and S_3, respectively. The numbers denote the indices of the scenarios that the jobs belong to

$\sum_{\ell \in A_1} a_\ell = \sum_{\ell \in A_2} a_\ell = \sum_{\ell \in A_3} a_\ell = \frac{1}{3}\sum_{\ell=1}^{n} a_\ell$, i.e., if the set $a_1, \ldots, a_n$ is a YES-instance of Partition-3. □

This reduction establishes that MinMaxSTC is weakly NP-hard. For the case with a constant number of machines, the weak NP-hardness cannot be strengthened to strong NP-hardness (unless P=NP) since on a fixed number of machines and scenarios an optimal solution can be found in pseudopolynomial time by dynamic programming. Moreover, via standard rounding techniques, one can obtain an FPTAS for MinMaxSTC.

Theorem 6 There exists a pseudopolynomial time algorithm as well as a fully polynomial time approximation scheme for MinMaxSTC on a constant number of machines with a constant number of scenarios.

Proof In order to develop a pseudopolynomial algorithm, we first observe a bound on the contribution of the *i*-th machine to the objective function value of MinMaxSTC (cf. (1)):

$$\max_{k \in [K]} \sum_{j \in J_i \cap S_k} w_j \cdot |\{j' \in J_i \cap S_k : j' \leq j\}| \leq n^2 W,$$

where $W := \max_{j=1}^{n} w_j$.

We propose a dynamic program that, given integers z_{ik}, y_{ik} with $i \in [m]$ and $k \in [K]$, decides whether there is an assignment of the jobs such that the *i*-th machine has load y_{ik} and contributes exactly z_{ik} to the weighted sum of completion times of the *k*-th scenario. More precisely, for increasing $j \in \{1, \ldots, n\}$, the dynamic program stores assignments of the first *j* jobs

$$\Phi^j \begin{bmatrix} y_{11} & \cdots & y_{1K} & z_{11} & \cdots & z_{1K} \\ \vdots & & \vdots & & & \vdots \\ y_{m1} & \cdots & y_{mK} & z_{m1} & \cdots & z_{mK} \end{bmatrix} : [j] \to [m]$$

(e.g. as vectors) indexed by $2mK$ parameters with the property that this assignment fulfills

$$\sum_{j \in J_i \cap S_k} w_j \cdot |\{j' \in J_i \cap S_k : j' \leq j\}| = z_{ik} \tag{13}$$

and

$$|\{j' \in J_i \cap S_k : j' \leq j\}| = y_{ik} \tag{14}$$

for all $i \in [m]$ and $k \in [K]$, where J_i denotes the set of jobs assigned to the i-th machine as usual.

To allow ourselves a more compact notation, we denote these maps by $\Phi^j[YZ]$ whenever the matrix entries are insignificant or the matrices Y and Z are evident.

It holds that $y_{ik} \leq n$ for all $i \in [m], k \in [K]$. Moreover, since we are interested in an optimal schedule only, we can bound $z_{ik} \leq n^2W + 1$ for all $i \in [m]$, $k \in [K]$.

Given matrices $Y \in [n]^{m \times K}$, $Z \in \{1, \ldots, n^2W\}^{m \times K}$, an assignment $\Phi[YZ]$ does not have to exist in general. However, as we shall see later, matrices (YZ) without corresponding assignments will not be called during the algorithm.

On the other hand, two different assignments can correspond to the same parameters Y, Z. This, however, will not be an issue for our purposes: If two assignments $\varphi, \varphi' : [n'] \to [m]$ for some $n' \leq n$ both corresponded to the same parameters y_{ik} and z_{ik}, this implies that the loads and costs of the machines are identical for the two assignments on each scenario. Therefore the dynamic program, whose main purpose is to compute an optimal assignment $\Phi^n[Y, Z] : [n] \to [m]$, could use either of φ and φ' as partial assignments. Thus, we only store at most one assignment per matrix (YZ).

For $n = 1$, it is easy to decide whether a partial assignment for the given matrix (YZ) exists. This is the case if and only if

$$y_{ik} = \begin{cases} 1 & i = i', 1 \in S_k \\ 0 & \text{else} \end{cases}$$

and

$$z_{ik} = \begin{cases} w_1 & i = i', 1 \in S_k \\ 0 & \text{else} \end{cases}$$

for a fixed $i' \in [m]$. In this case, the partial assignment $\Phi^1[YZ] : [1] \to [m]$ maps the first and only job to the i'-th machine.

Now let $2 \leq n' \leq n$ be fixed but arbitrary. By induction, we may assume that we have defined the assignments

$$\Phi^j \begin{bmatrix} y_{11} & \cdots & y_{1K} & z_{11} & \cdots & z_{1K} \\ \vdots & & \vdots & & & \vdots \\ y_{m1} & \cdots & y_{mK} & z_{m1} & \cdots & z_{mK} \end{bmatrix} : [j] \to [m]$$

for all $j \leq n' - 1$ and all possible values of $(y_{ik})_{i\in[m],k\in[K]}, (z_{ik})_{i\in[m],k\in[K]}$ for which an assignment indeed exists so as to satisfy (13) and (14).

Now we consider the process of assigning the n'-th job to a machine, which would mean an extension of an assignment $\Phi^{n'-1}[Y'Z']$ which exists by induction for appropriate matrices $Y' = (y'_{ik})_{i\in[m],k\in[K]}$ and $Z' = (z'_{ik})_{i\in[m],k\in[K]}$.

Assigning the n'-th job extends the map $\Phi^{n'-1}[Y'Z']$ and changes the load and contribution of the machine to which we assign n' to the objective function value. That is, the resulting assignment would be encoded by parameters $Y = (y_{ik})$ and $Z = (z_{ik})$ with entries $y_{ik} = y'_{ik} + 1$ and $z_{ik} > z'_{ik}$ whenever $n' \in J_i \cap S_k$, and $y_{ik} = y'_{ik}$, $z_{ik} = z'_{ik}$ otherwise. To be more precise, the increase of the contribution is given by

$$z_{ik} - z'_{ik} = w_{n'} \cdot |\{j' \in J_i \cap S_k : j' \leq n'\}| = w_{n'} \cdot (y'_{ik} + 1).$$

Our dynamic program checks for all possible $Y = (y_{ik})_{i\in[m],k\in[K]}$ and $Z = (z_{ik})_{i\in[m],k\in[K]}$ (with y_{ik} and z_{ik} bounded as above) if there exists an $m' \leq m$ such that the map

$$\Phi^{n'-1}[Y'Z'] : [n'-1] \to [m]$$

exists, where the matrices Y' and Z' are defined as

$$Y' = \begin{bmatrix} y_{11} & \cdots & y_{1K} \\ \vdots & & \vdots \\ y_{m'1} - |\{n'\} \cap S_1| & \cdots & y_{m'1} - |\{n'\} \cap S_K| \\ \vdots & & \vdots \\ y_{m1} & \cdots & y_{mK} \end{bmatrix}$$

and

$$Z' = \begin{bmatrix} z_{11} & \cdots & z_{1K} \\ \vdots & & \vdots \\ z_{m'1} - w_{n'} y_{m'1} |\{n'\} \cap S_1| & \cdots & z_{m'K} - w_{n'} y_{m'K} |\{n'\} \cap S_K| \\ \vdots & & \vdots \\ z_{m1} & \cdots & z_{mK} \end{bmatrix}$$

If this is the case, we define $\Phi^{n'}[YZ] : [n'] \to [m]$ as

$$\Phi^{n'}[YZ](j) = \begin{cases} \Phi^{n'-1}[Y'Z'](j) & j < n' \\ m' & j = n' \end{cases}$$

If such an $m' \leq m$ does not exist, we do not define $\Phi^{n'}[YZ]$. This way, $\Phi^{n'}[YZ]$ is defined by the algorithm if and only if it is the extension of another defined assignment $\Phi^{n'-1}[Y'Z']$ by the n'-th job. By induction, the algorithm therefore defines an assignment if and only if the corresponding schedule with given y_{ik}, z_{ik} is indeed realizable.

Note that deciding whether m' exists takes linearly many iterations in *m* by linear search. Moreover, there are at most $O(n^{mK} \cdot (n^2W)^{mK}) = O((n^3W)^{mK})$ different maps $\Phi^{n'}[YZ]$ for a fixed $n' \leq n$ and matrices *Y*, *Z*. For all possible n', there are $O(n(n^3W)^{mK})$ maps. Given all possible $\Phi^{n'-1}[YZ]$, computing $\Phi^{n'}[YZ]$ takes *O*(*m*) time and therefore the total runtime of computing the state space is $O(mn(n^3W)^{mK})$.

Given the dynamic program, we can solve the scheduling problem as follows: Starting with $y_{\max} = z_{\max} = 1$ and incrementing $z_{\max}$ and y_{max} upwards in this order, we enumerate all matrices [*YZ*] with $\sum_{i=1}^{m} y_{ik} = |S_k|$ for all $k \in [K]$ and $\max_{k=1}^{K} \sum_{i=1}^{m} z_{ik} = z_{\max}$, checking every time whether $\Phi^n[YZ]$ is defined. Once a number $z_{\max}$ is found for which an appropriate $y_{\max}$ as well as the matrix [*YZ*] as above exist, we output the schedule $\Phi^n[YZ]$. The runtime of this procedure is dominated by that of the dynamic program. Since the runtime for the dynamic program is polynomial in *n* and *W*, this yields a pseudopolynomial algorithm. Its running time is

$$O(mn(n^3W)^{mK}). \tag{15}$$

Next, we show how to turn the pseudopolynomial algorithm into an FPTAS. Let $\mathcal{I} = (w_1, \ldots, w_n, S_1, \ldots, S_K)$ be an instance of MinMaxSTC and $\varepsilon > 0$. For a number ρ to be determined later, we define $w'_j := \left\lceil \frac{w_j}{\rho} \right\rceil$. We apply the algorithm given above to the instance $\mathcal{I}' = (w'_1, \ldots, w'_n, S_1, \ldots, S_K)$ and output the (exact) solution given by this algorithm for the instance $\mathcal{I}'$. Let $J'_1, \ldots, J'_m$ be the partitioning of the jobs given by an optimal schedule of the instance $\mathcal{I}$. Let alg denote the value of the output of our proposed FPTAS that returns the optimal solution for the instance $\mathcal{I}'$, and opt the optimal value for the instance $\mathcal{I}$. Then we have, using (1),

$$\mathrm{alg} = \max_{k \in [K]} \sum_{i=1}^{m} \sum_{j \in J'_i \cap S_k} w_j \cdot |\{j' \in J'_i \cap S_k : j' \leq j\}| \tag{16}$$

$$\leq \max_{k \in [K]} \sum_{i=1}^{m} \sum_{j \in J'_i \cap S_k} w'_j \cdot \rho \cdot |\{j' \in J'_i \cap S_k : j' \leq j\}| \tag{17}$$

$$\leq \max_{k \in [K]} \sum_{i=1}^{m} \sum_{j \in J_i \cap S_k} w'_j \cdot \rho \cdot |\{j' \in J_i \cap S_k : j' \leq j\}| \tag{18}$$

$$< \max_{k\in[K]} \sum_{i=1}^{m} \sum_{j\in J_i\cap S_k} (w_j+\rho)\cdot |\{j'\in J_i\cap S_k : j'\le j\}| \tag{19}$$

$$= \mathrm{opt} + \rho\cdot \max_{k\in[K]} \sum_{i=1}^{m} \sum_{j\in J_i\cap S_k} |\{j'\in J_i\cap S_k : j'\le j\}| \le \mathrm{opt} + \rho m n^2 \tag{20}$$

On the other hand, we know that $\mathrm{opt} \ge \max_{j=1}^{n} w_j =: W$. Therefore,

$$\frac{\mathrm{alg}}{\mathrm{opt}} < 1 + \frac{\rho m n^2}{\mathrm{opt}} \le 1 + \frac{\rho m n^2}{W}$$

We set $\rho := \frac{W\varepsilon}{mn^2}$. Then we have $\frac{\mathrm{alg}}{\mathrm{opt}} \le 1+\varepsilon$. Moreover, it holds that

$$w_j' = \left\lceil \frac{w_j m n^2}{W\varepsilon} \right\rceil \le \frac{mn^2}{\varepsilon} + 1.$$

The rounding for the FPTAS redefines the largest weight W to be $1 + {}^{mn^2}/_{\varepsilon}$. Inserted into (15) yields the runtime of the FPTAS $O\big(mn\big(n^3({}^{mn^2}/_{\varepsilon})\big)^{mK}\big)$, indeed polynomial in *n* and ${}^{1}/_{\varepsilon}$ assuming that *m* and *K* are constant.

$\square$

5 The MinAvg Version

By Theorem 3, MinAvgSTC is solvable in polynomial time in the case of two scenarios and by Theorem 4 in case of a constant number of scenarios and bounded job weights. For general job weights, however, we need to design a different dynamic program that solves MinAvgSTC in polynomial time for any constant number of scenarios if there is also a constant number of machines; see Section 5.1.

In Section 5.2, we present a conjecture that, if true, leads to a polynomial time dynamic programming algorithm for *any* number of machines. We prove the conjecture for the special case of unit job weights which results in an efficient algorithm for MinAvgSTC in this case that is faster than the one given in the previous section as a function of the number of jobs, but slower as a function of the number of machines.

Moreover, the techniques can be adapted to the more general case that the number of distinct job weights are bounded by a constant. We sketch how this can be achieved.

5.1 Constant Number of Machines

We first describe the case of a constant number of machines. Recall that the objective function for MinAvgSTC is defined as

$$\sum_{k=1}^{K}\sum_{i=1}^{m}\sum_{j\in J_i\cap S_k} w_j\cdot|\{j'\in J_i\cap S_k : j'\leq j\}|,$$

with J_i the set of jobs assigned to machine *i*.

It is clear from the objective function that the contribution of some job *j* to the cost of a solution depends only on the assignment of that job and any jobs with a lower index, i.e., jobs $1,\dots,j-1$, to the various machines. In particular, if we want to compute the contribution of job *j* to the cost of a solution in some schedule, it is sufficient to know the following quantities for each $i\in[m], k\in[K]$

$$x_{ik}(j-1) := |\{j'\in J_i\cap S_k : j'<j\}|,$$

which together form a *state* of the scheduling process.

If job *j* gets assigned to machine *i* in that schedule, its contribution to the overall cost would then be

$$\sum_{k\in[K]:j\in S_k} w_j(1+x_{ik}(j-1)).$$

In light of these observations, one can derive a dynamic program, leading to the following theorem.

Theorem 7 The MinAvgSTC problem with constant number of machines and constant number of scenarios can be solved in polynomial time.

Proof We define a state in a dynamic programming decision process as a partial schedule at the moment the first *j* jobs have been assigned and encode this by a $m\times K$ matrix *X*(*j*), with $x_{ik}(j)=|\{j'\in J_i\cap S_k : j'\leq j\}|$.

This leads to a simple dynamic program, using the following recursion, where we use $f_j(X(j))$ to denote the minimum cost associated with the first *j* jobs in any schedule that can be represented by *X*(*j*):

$$f_j(X(j)) = \min_{\ell\in[m]}\left\{f_{j-1}(X(j-1,\ell)) + \sum_{k\in[K]:j\in S_k} w_j(1+x_{\ell k}(j-1))\right\},$$

where $X(j-1,\ell)$ is the matrix $X(j-1)$ from which *X*(*j*) is obtained by assigning job *j* to machine ℓ. Equivalently, $X(j-1,\ell)$ is the matrix obtained from *X*(*j*) by diminishing all positive entries in row ℓ of *X*(*j*) by 1. It follows that $X(j-1,\ell)$ has entries $x_{ik}(j-1)$ that satisfy $x_{\ell k}(j-1)=x_{\ell k}(j)-1_{j\in S_k}$ for all *k*, and $x_{ik}(j-1)=x_{ik}(j)$ for all $i\neq\ell$, and all *k*. Therefore, the computation of each state can be done in time *O*(*mK*).

We initialize $f_0(0)=0$ (where 0 denotes the all-zero matrix) and set $f_0(X)=\infty$ for any other possible *X*. Thus in each of the *n* phases (partial job assignments) the number of possible states is bounded by $(n+1)^{m\times K}$. This all leads to a running time of $O(mKn^{mK+1})$, which is polynomial given that *m* and *K* are constants in this particular case. □

5.2 Any Number of Machines

In this section we develop another efficient algorithm for MinAvgSTC on an arbitrary number of machines with a constant number of scenarios and unit job weights.

In contrast to the dynamic program presented in the proof of Theorem 4, the running time is linear in *n*, the number of jobs, but polynomial in *m* with the power a function of *K*, that is, the running time is of the form $O(nm^{h(K)})$ for some function *h*.

More importantly, the technique that we use here is new and we believe it can be generalised to arbitrary job weights. For the time being it remains a fascinating open question whether MinAvgSTC can even be solved efficiently for arbitrary job weights.

As we will explain, this is true under the assumption that the following conjecture holds, which we do believe but can only prove for the special case of unit job weights.

Conjecture 1 For every instance, MinAvgSTC has an optimal solution such that for every scenario $k\in[K]$ and each $j\in[n]$, the jobs $1,\dots,j$ are assigned to the machines in such a way that the difference in number of jobs assigned to each pair of machines is bounded by a function *g*(*K*) of *K* only, or more formally

$$\max_{j\in[n],k\in[K]}\left\{\max_{i\in[m]}|\{j'\in J_i\cap S_k : j'\le j\}| - \min_{i\in[m]}|\{j'\in J_i\cap S_k : j'\le j\}|\right\}\le g(K).$$

We call the term on the left-hand side *full disbalance* of the schedule. To adjust the dynamic program in the proof of Theorem 7 to run in polynomial time for any number of machines under the conjecture, we first observe that to compute a recursion step, the order of the rows in each matrix *X* representing a partial schedule is irrelevant: we only need to know how many machines have a certain number of jobs assigned to them under each scenario, not exactly which machines.

A first step to encode partial schedules is using vectors $\ell\in\mathcal{C}\subseteq\{0,\dots,n\}^K$, where we call $\mathcal{C}$ the set of machine configurations. If a machine has configuration ℓ, it means that in the partial schedule ℓ_k jobs have been scheduled on this machine in scenario *k*.

We then simply represent a partial schedule by the number of machines that have configuration ℓ.

We consider the space of *states* that say how many machines have a certain configuration. We can further reduce this space by storing the smallest number of jobs on any machine in a given scenario separately, that is,

$$z_k=\min_{i\in[m]}x_{ik},\ k\in[K].$$

The crucial observation now is that under Conjecture 1, there is an optimal solution such that for any partial schedule corresponding to that solution, $0\le x_{ik}-z_k\le g(K)$.

We may therefore take $\mathcal{C}$ to be $\{0,\dots,g(K)\}^K$, i.e., the excess over z_k, so that $\mathcal{C}$ has constant size for constant K and define

$$y_\ell := |\{i \in [m] : x_{ik} - z_k = \ell_k \text{ for all } k \in [K]\}|$$

for all $\ell \in \mathcal{C}$. Since the entries of y are bounded by m and the entries of z are bounded by n, the possible number of values for a pair (y, z) in the encoding above is $(m+1)^{(g(K)+1)^K}(n+1)^K$, yielding a polynomial size for the state space. Since the dynamic programming computation for each state in each phase (job assignment) takes $O(m)$ steps and there are n consecutive job assignments in SPT-order in the dynamic program, this yields a polynomial time algorithm.

It is still open whether there exists an upper bound on the full disbalance of the schedule depending only on K. In the following subsection, we affirm this statement for the case where all jobs have unit weights. Note that, in this case, the j largest jobs are not well-defined. We therefore prove the stronger statement that for any ordering of the jobs, the conjecture holds. To do so, we utilize the power of integer programming, combining the theory of Hilbert bases with techniques of an algorithmic nature.

5.3 Proof of Conjecture 1 for Unit Weights

In the following, we prove the unit-weight case of Conjecture 1 in a constructive manner. More precisely, our goal is to prove the following theorem, which implies the case of Conjecture 1 where the jobs have unit weights.

Theorem 8 Conjecture 1 holds for unit weights (and unit processing times), where the jobs are given by an arbitrary fixed ordering and the number m of machines is part of the input. The full disbalance is then bounded by

$$4K \cdot f(K) = 4K\left(2^{2^{K+1}}K! + 1\right)^{2\cdot 2^K} \cdot 2^{2^{K+1}+K+1}K! + 4K\sqrt{K}\cdot 2^{K-1}.$$

5.3.1 Proof for $m = 2$

Initially, we show the statement for $m = 2$ as opposed to m being an arbitrary number of machines that is part of the input.

To this end, we first prove a weaker statement regarding the (non-full) disbalance at the end of the schedule. We shall define the *final disbalance under scenario k* as

$$d_k := \max_{i\in[m]} |J_i \cap S_k| - \min_{i\in[m]} |J_i \cap S_k|$$

and the *final disbalance* as $d := \max_k d_k$. It is straightforward to see that

$$d = 0 \Leftrightarrow d_k = 0 \ \forall k \Leftrightarrow \sum_{k=1}^{K} d_k = 0.$$

Lemma 1 *For an instance of MinAvgSTC with unit weights (and unit processing times), the following hold:*

(a) The optimal solutions are precisely those that minimize $\sum_{i=1}^{m}\sum_{k=1}^{K}|J_i \cap S_k|^2$.
(b) If $m = 2$, the optimal solutions are precisely those that minimize $\sum_{k=1}^{K} d_k^2$.

Proof (a) The objective value for a solution equals

$$\sum_{i=1}^{m}\sum_{k=1}^{K}\frac{|J_i \cap S_k|\cdot(|J_i \cap S_k|+1)}{2} = \frac{1}{2}\sum_{i=1}^{m}\sum_{k=1}^{K}|J_i \cap S_k|^2 + \frac{1}{2}\sum_{i=1}^{m}\sum_{k=1}^{K}|J_i \cap S_k|$$
$$=\frac{1}{2}\sum_{i=1}^{m}\sum_{k=1}^{K}|J_i \cap S_k|^2 + \frac{1}{2}\sum_{k=1}^{K}|S_k|$$

because $J_1\dot{\cup}\dots\dot{\cup}J_m$ is a partitioning of the jobs. The second summand does not depend on the solution $J_1\dot{\cup}\dots\dot{\cup}J_m$, hence this value is minimized if and only if $\sum_{i=1}^{m}\sum_{k=1}^{K}|J_i \cap S_k|^2$ is minimized.

(b) By (a), for $m = 2$ a solution is optimal if and only if

$$\sum_{k=1}^{K}(|J_1 \cap S_k|^2 + |J_2 \cap S_k|^2)$$
$$=\sum_{k=1}^{K}\frac{((|J_1 \cap S_k| + |J_2 \cap S_k|)^2 + (|J_1 \cap S_k| - |J_2 \cap S_k|)^2)}{2}$$
$$=\frac{1}{2}\sum_{k=1}^{K}(|S_k|^2 + d_k^2).$$

is minimized. Once again, $|S_k|$ does not depend on the solution; therefore, this term is minimized if and only if $\sum_{k=1}^{K} d_k^2$ is minimized. □

This criterion automatically singles out a class of instances whose optimal solutions can be described by their final disbalances, as in the following lemma.

Lemma 2 *For an instance of MinAvgSTC on two machines with unit weights (and unit processing times) which has a solution with final disbalance zero, the optimal solutions are precisely those that have final disbalance zero.*

Next, we argue how Lemma 1 implies an upper bound on the final disbalances of optimal solutions of unit-weight instances.

Lemma 3 *For any optimal solution of an instance of MinAvgSTC with unit weights (and unit processing times), it holds that*

$$d_k \leq \sqrt{K} \cdot 2^{K-1} forall\ k \in [K],$$

i.e., the final disbalance d of resulting schedules with respect to any scenario is bounded by a function only dependent of the number K of scenarios.

Proof Let an optimal solution $\varphi : [n] \to [m]$ be given to an instance with unit weights and processing times. Assume without loss of generality that in the first scenario, the solution restricted to machines 1 and 2 admits a final disbalance strictly larger than $\sqrt{K} \cdot 2^{K-1}$, i.e.,

$$|J_1 \cap S_1| - |J_2 \cap S_1| > \sqrt{K} \cdot 2^{K-1}$$

for $J_i = \varphi^{-1}(i)$ $(i = 1, 2)$. Our goal is to apply a redistribution of the jobs assigned to machines 1 and 2 such that the objective value is strictly decreased after the distribution, which contradicts the optimality of the initial solution. Since the assignments to the remaining machines stay unchanged, it suffices to show that the objective value of MinAvgSTC restricted to the first two machines strictly decreases after the redistribution. Hence, for the remainder of this proof, we may assume that $m = 2$.

For the redistribution of the jobs, notice that jobs that appear in the same scenarios are indistinguishable for our problem. Thus, we obtain at most 2^K disjoint subsets by grouping jobs according to precisely which subset of scenarios the jobs appear in. Within each subset we order the jobs in any fixed order, and assign them in this list order to the two machines, always assigning the next job within the subset to the least loaded machine, leading to an (almost) even load per machine per subset of equivalent jobs.

Formally, we redefine J_1 and J_2 by Algorithm 1.

Algorithm 1 A distribution of jobs according to the scenarios they appear in

```
$J_1, J_2 \leftarrow \emptyset$
**for** $I \subseteq [K]$, s.t. $S_I := \bigcap_{k \in I} S_k \setminus \bigcup_{k \notin I} S_k \neq \emptyset$ **do**
    Let $S_I =: \{j_1^I, \ldots, j_{q_I}^I\}$
    $J_1 \leftarrow J_1 \cup \{j_i^I \in S_I : i + |I| \text{ odd}\}$
    $J_2 \leftarrow J_2 \cup \{j_i^I \in S_I : i + |I| \text{ even}\}$
**end for**
```

For a solution obtained this way, we immediately observe that the final disbalance under scenario $k \in [K]$, which we denote by d'_k, satisfies $d'_k \leq 2^{K-1}$, for each k. As we know that $d_1 > \sqrt{K} \cdot 2^{K-1}$, we obtain

$$\sum_{k=1}^{K} d_k^2 \geq d_1^2 > (\sqrt{K} \cdot 2^{K-1})^2 = K \cdot 2^{2K-2} \geq \sum_{k=1}^{K} (d'_k)^2,$$

giving a contradiction to φ being an optimal solution by Lemma 1(b). □

Recall that the *full disbalance f* of a schedule is given by $f = \max_{k \in [K]} f_k$, where

$$f_k := \max_{j\in[n]} \left\{ \max_{i\in[m]} |\{j' \in J_i \cap S_k : j' \leq j\}| - \min_{i\in[m]} |\{j' \in J_i \cap S_k : j' \leq j\}| \right\}.$$

We propose an algorithm that provides a proof of the conjecture for the unit weight case and $m = 2$. This algorithm will be used as a subroutine in an algorithm that provides a proof of the unit weight case with arbitrarily many machines.

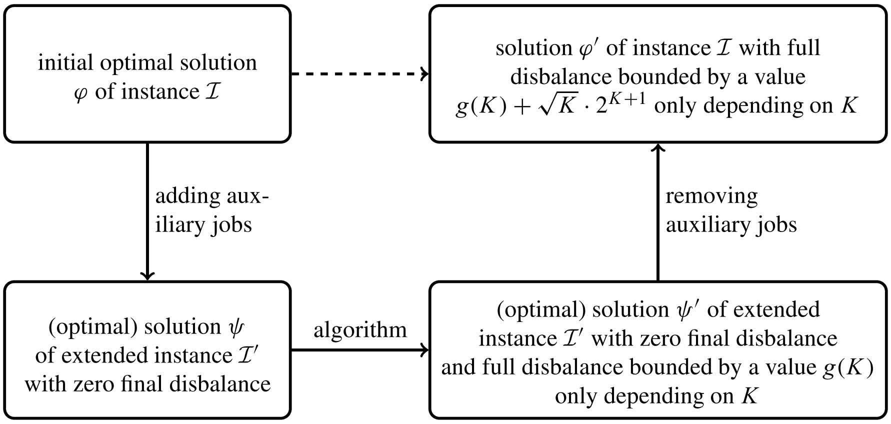
Theorem 9 There exists an algorithm that takes a unit-weight instance of MinAvg-STC on $m = 2$ machines as well as an optimal solution of this instance as input, and outputs an optimal solution with full disbalance at most

$$\left(2^{2^{K+1}} K! + 1\right)^{2\cdot 2^K} \cdot 2^{2^{K+1}+K+1} K! + \sqrt{K} \cdot 2^{K-1}.$$

Proof We describe an algorithm that produces the solution as in the conjecture, starting from an arbitrary optimal solution.

Due to some restrictions of the techniques that we apply, we aim to start with a solution $\varphi : [n] \to \{1, 2\}$ that has zero final disbalance under each scenario, which is equivalent to $\sum_{k=1}^{K} d_k = 0$, though it is evident that such a solution need not exist. However, by Lemma 3, we know that the final disbalance of any optimal solution is at most $\sqrt{K} \cdot 2^{K-1}$. To obtain a solution $\psi : [n + \sum_{k=1}^{K} d_k] \to \{1, 2\}$ with zero final disbalance, we add d_k auxiliary jobs that appear only in scenario *k* for $k = 1, \ldots, K$. An optimal solution of this extended instance now has zero final disbalance at the end of the schedule for every scenario by Lemma 2. We apply our algorithm to this extended optimal solution to obtain a solution $\psi' : [n + \sum_{k=1}^{K} d_k] \to \{1, 2\}$ and remove the auxiliary jobs at the end, obtaining an assignment $\varphi' : [n] \to \{1, 2\}$.

The following diagram demonstrates our reduction onto instances with a solution that satisfies $\sum_{k=1}^{K} d_k = 0$.


initial optimal solution φ of instance $\mathcal{I}$
solution φ' of instance $\mathcal{I}$ with full disbalance bounded by a value $g(K) + \sqrt{K} \cdot 2^{K+1}$ only depending on K
adding auxiliary jobs
removing auxiliary jobs
(optimal) solution ψ of extended instance $\mathcal{I}'$ with zero final disbalance
algorithm
(optimal) solution ψ' of extended instance $\mathcal{I}'$ with zero final disbalance and full disbalance bounded by a value $g(K)$ only depending on K

By Lemma 3, removing the auxiliary jobs increases each f_k by at most $\sqrt{K} \cdot 2^{K-1}$; therefore the full disbalance $f(\varphi')$ of the solution φ' is at most $g(K) + \sqrt{K} \cdot 2^{K-1}$ if the full disbalance of the solution φ is $g(K)$. One also easily observes that the resulting solution is an optimal one: □

Claim 2 The solution φ' as described above is optimal.

Proof As observed in the Lemma 1 above, it suffices to show that

$$\sum_{k=1}^{K} d_k(\varphi')^2 \leq \sum_{k=1}^{K} d_k(\varphi)^2.$$

By our assumptions, we have $d_k(\psi') = 0$ for every $k \in [K]$. Moreover, removing auxiliary jobs can create at most as much additional final disbalance for scenario *k* as there are auxiliary jobs appearing in scenario *k* (which is $d_k(\varphi)$); in total, we observe $d_k(\varphi') \leq d_k(\psi') + d_k(\varphi) = d_k(\varphi)$. Taking squares of this equation and summing over each *k*, we obtain the desired inequality. □

Hence, in the following, we may assume that there exists a solution φ with $\sum_{k=1}^{K} d_k(\varphi) = 0$. In fact, precisely the optimal solutions satisfy this by Lemma 2.

For an arbitrary but fixed numbering of all possible subsets *S* of the *K* scenarios, i.e., a bijection $\sigma : 2^{\{1,\ldots,K\}} \to \{1, \ldots, 2^K\}$, let $M \in \{0,1\}^{K \times 2^K}$ be given by entries

$$M_{k\sigma(S)} := \begin{cases} 1 & k \in S, \\ 0 & \text{else.} \end{cases}$$

This matrix helps us to compute how many jobs are assigned to each machine for each scenario. Let $x, y \in \mathbb{Z}_{\geq 0}^{2^K}$ encode the number of jobs of each profile in machines 1 and 2, respectively. That is, for $S \subseteq [K]$, the $\sigma(S)$-th entry of *x* resp. *y* denotes the number of jobs appearing precisely in scenarios $k \in S$ in machine 1 resp. 2. Then, the vectors $Mx, My \in \mathbb{Z}_{\geq 0}^{K}$ have entries corresponding to the number of jobs appearing in each scenario, that is, their respective *k*-th entry denotes the number of jobs on machine 1 resp. 2 that appear in scenario $k \in [K]$.

A solution $\psi : [n] \to \{1, 2\}$ is optimal if and only if $Mx = My$. Motivated by this observation, we consider the polyhedral cone

$$C := \{(x, y) \in \mathbb{R}_{\geq 0}^{2 \cdot 2^K} : Mx - My = 0\}.$$

The set of optimal solutions is given by a subset of the lattice points $C_I := C \cap \mathbb{Z}^{2 \cdot 2^K}$. It is easy to observe that *C* is a pointed rational convex cone, that is, we have

$$C = \text{cone}(r^1, \ldots, r^{q(K)})$$

where $r^1, \dots, r^{q(K)}$ are the extreme rays of *C*. Indeed, as there are 2^{K+1} inequalities defining *C*, there can be at most $2^{2^{K+1}}$ subsystems that can define an extreme ray, therefore the number of extreme rays is bounded by $q(K) \leq 2^{2^{K+1}}$.

Claim 3 For the polyhedral cone *C*, there exist extreme rays $r^p \in \mathbb{Z}^{2^{K+1}}$ for $p \in [q(K)]$ such that $\|r^p\|_\infty \leq K!$.

Proof In the following, let *N* denote the matrix $(M \quad -M)^T \in \{-1, 0, 1\}^{K \times (2 \cdot 2^K)}$.

A fixed edge of the cone *C* is given by $\{v = (x, y) \in C : N'(x, y) = 0\}$ for a $(K-1) \times 2^K$ submatrix N' of *N* with full rank. Since the system $N'v = 0$ is underdefined, we may assume that all variables of *v* except *K* variables indexed $\alpha_1, \dots, \alpha_K$ are zero.[2] We further restrict the system of equalities to the potentially nonzero variables $v' := (v_{\alpha_1}, \dots, v_{\alpha_K})^T$ and consider the system $N''v' = 0$ instead, where N'' consists of the columns of N' indexed $\alpha_1, \dots, \alpha_K$.

In order to describe a point on the ray, we would like to add an equality $\gamma^T v' = 1$ to the system

$N''v' = 0$ of equalities. Here, γ^T must be the restriction of a row of the matrix *N* to its $\alpha_1, \dots, \alpha_K$-th components which is linearly independent from the rows of the matrix N''. Furthermore, we would like to ensure that the solution of the resulting system lies in *C*, that is, it also satisfies the (restricted) inequalities describing the cone that are not necessarily tight.

One candidate for the vector γ is the α-th unit vector e^α for a suitable index α. Indeed, by basis extension, there exists an index α such that the unit vector e^α is linearly independent from the rows of the matrix N''. Moreover, the equality $(e^\alpha)^T v' = 1$ simply means $v'_\alpha = 1$, which does not violate any inequalities (as the inequality $v'_\alpha \geq 0$ could not have been tight).

To prove the claim, we define the extreme ray corresponding to the fixed edge of the cone as the unique solution v' to the system

$$(N^\alpha)^T v' := \begin{pmatrix} N'' \\ (e^\alpha)^T \end{pmatrix} v' = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix},$$

extended by zeroes so as to obtain *v* as described above.

We have already argued that *v* lies on the edge that we have fixed. Its *k*-th entry is given by $\frac{\det N_k^\alpha}{\det N^\alpha}$ by Cramer's rule ([29], Chapter 5.3) for every $k \in \{\alpha_1, \dots, \alpha_K\}$ (and 0 otherwise). Here N_k^α is the matrix that is obtained by replacing the *k*-th column by the vector $(e^\alpha)^T \in \mathbb{R}^K$. In particular, the vector $(\det N^\alpha) \cdot v$ is an integral vector on the ray. Therefore, it suffices to show that this vector has entries bounded by *K*!, that

[2] In fact, we can reduce this number to $K-1$ instead of *K* such that the statement is still true, but then the only feasible solution would be the zero vector, which we want to avoid.

is, $|\det N_k^\alpha| \leq K!$ for all k. However, this is true because by the definition of determinant (cf. [29], Chapter 5.2), we have

$$|\det N_k^\alpha| = \left| \sum_{\tau \in \mathcal{S}_K} \operatorname{sgn}(\tau) \prod_{i=1}^{n} (N_k^\alpha)_{i\tau(i)} \right| \leq \sum_{\tau \in \mathcal{S}_K} \left| \operatorname{sgn}(\tau) \prod_{i=1}^{K} (N_k^\alpha)_{i\tau(i)} \right| \leq K!$$

since N_k^α is a square matrix of size K with entries in $\{0, 1, -1\}$. □

An implication of C being a pointed cone is that the *irreducible* lattice points of C_I, that is, those that cannot be written as the sum of two nonzero points in C_I, build a Hilbert basis B. In other words, every point in C_I (and hence the encoding of every optimal solution) is an integer linear combination of the set of irreducible vectors in C_I,

In the context of our problem, these vectors correspond to partial solutions with final disbalance zero that do not have subsolutions with final disbalance zero.

Claim 4 For any vector $b \in B$ of the Hilbert basis, we have $\|b\|_\infty \leq 2^{2^{K+1}} K!$.

Proof It is well-known (cf. [30]) that each element b of B can be written as a linear combination $b = \sum_{p=1}^{q(K)} \lambda_p r^p$ for suitable $\lambda_p \in [0, 1]$ and extreme rays r^p of C, $p \in [q(K)]$.

Hence

$$\|b\|_\infty = \left\| \sum_{p=1}^{q(K)} \lambda_p r^\ell \right\|_\infty \leq \sum_{p=1}^{q(K)} \lambda_p \|r^p\|_\infty \leq 2^{2^{K+1}} K!$$

by Claim 3. □

Claim 5 For the Hilbert basis B, it holds that $|B| \leq \left(2^{2^{K+1}} K! + 1\right)^{2 \cdot 2^K}$.

Proof By Claim 4, each entry in a basis vector admits integral values between 0 and $2^{2^{K+1}} K!$. There are hence $2^{2^{K+1}} K! + 1$ choices for each of the $2 \cdot 2^K$ entries. □

Now consider an optimal schedule (i.e., one with final disbalance zero) and the corresponding vector $v \in C_I$. Then, by definition of a Hilbert basis, we find a decomposition

$$v = \sum_{b \in B} \lambda_b b \tag{21}$$

with $\lambda_b \in \mathbb{N}$ and $\|b\|_\infty \leq 2^{2^{K+1}} K!$. This means that we can partition jobs in our schedule into $\sum_{b \in B} \lambda_b$ sets each of which has final disbalance zero. These job subsets correspond to the elements of the basis B; for each $b \in B$, there are λ_b subsets that correspond to the support of the vector b.

The core idea of the algorithm claimed in the statement of the theorem is to exploit the fact that $|B|$ is bounded by a function of K. Since λ_b are not necessarily bounded by a function of K, our aim is to bound the sum of full disbalances of the λ_b classes that contribute to the same $b \in B$ at all times.

To describe the algorithm, we first consider a fixed $b \in B$. Let $\tau : [n] \to 2^{[K]}$, $j \mapsto \{k \in [K] : j \in S_k\}$ map each job to its type. Then each job j contributes to the vector v by a unit vector; more precisely by $e_{\sigma(\tau(j))}$ if it is assigned to the first machine and by $e_{2^K+\sigma(\tau(j))}$ if it is assigned to the second. Our algorithm takes the jobs that contribute to the summand $\lambda_b b$ of the sum (21) and assigns each job to the copy of b with the smallest possible index in which $\tau(j)$ "fits". This will ensure that every basis element contributes at most as much as the size of a single basis element to the disbalance at all times.

More precisely, let $B = \{b_1, \ldots, b_{|B|}\}$ and let $\psi : [n] \to \{1, 2\}$ be an assignment with zero final disbalance which corresponds to a vector $v = \sum_{p=1}^{|B|} \lambda_{b_p} b_p \in C_I$ and let $b_p^1, \ldots, b_p^{\lambda_{b_p}}$ denote copies of b_p.

Algorithm 2 computes an assignment $\psi' : [n] \to \{1, 2\}$ that we claim satisfies the desired properties.

Algorithm 2 Assignment of jobs via Hilbert basis. In the function ASSIGN(j, v), the vector v is *passed by reference*, that is, any alteration of the vectors b_p^ℓ within the function ASSIGN(j, v) is also kept after the execution of the function

```
  Input: A decomposition $v = \sum_{b\in B} \lambda_b b$ of a vector $v$ that corresponds to an optimal assignment $\psi: [n] \to \{1, 2\}$.
  Output: An optimal assignment $\psi': [n] \to \{1, 2\}$ with bounded full disbalance.
for $j = 1$ to $n$ do
    $\psi'(j) \leftarrow$ ASSIGN($j, v$)
end for

ASSIGN($j, v$):
for $p = 1$ to $|B|$ do
    for $\ell = 1$ to $\lambda_{b_p}$ do
        if $b_p^\ell - e_{\sigma(\tau(j))} \geq 0$ then
            $b_p^\ell \leftarrow b_p^\ell - e_{\sigma(\tau(j))}$
            if $\ell$ is odd then return 1
            else return 2
            end if
        else if $b_p^\ell - e_{2^K+\sigma(\tau(j))} \geq 0$ then
            $b_p^\ell \leftarrow b_p^\ell - e_{2^K+\sigma(\tau(j))}$
            if $\ell$ is odd then return 2
            else return 1
            end if
        end if
    end for
end for
```

To observe that the algorithm works correctly, we first note that it assigns all jobs to a machine eventually, which follows from the way the vector v and its summands are defined. Further, we observe that the assignment is balanced.

Claim 6 At any point of the algorithm, we have $b_p^\ell \leq b_p^{\ell'}$ for $\ell < \ell', p \in [|B|]$.

Proof We prove the claim via induction over the number j of assigned jobs. At the beginning of the algorithm, b_p^ℓ and $b_p^{\ell'}$ both equal b_p, therefore the claim holds.

Let j be an arbitrary job that is assigned to a machine at the p-th iteration of the second inner loop and the ℓ'-th iteration of the innermost loop. By induction, we may assume that $b_p^\ell \leq b_p^{\ell'}$ right before the j-th iteration of the outermost loop.

For the inequality not to hold after the assignment of the j-th job, $b_p^{\ell'}$ must have strictly decreased in the iteration that assigns the j-th job. That is, unless the j-th job is assigned to a machine in the ℓ'-th inner loop, the inequality is maintained.

If the job j is indeed assigned in the ℓ'-th inner loop, then it was not in the ℓ-th iteration, although this iteration was reached since $\ell < \ell'$. Therefore it must hold that $(b_p(\ell))_{\sigma(\tau(j))} = (b_p(\ell))_{2^K+\sigma(\tau(j))} = 0$ before (and after) the j-th iteration of the outermost loop. On the other hand, we have $(b_p^{\ell'})_{\sigma(\tau(j))}, (b_p^{\ell'})_{2^K+\sigma(\tau(j))} \geq 0$ at the end of the j-th iteration, for else the assignment for j would have happened later. Since all other entries of b_p^ℓ and $b_p^{\ell'}$ stay constant during this iteration, the claim follows. □

We note that the assignment is carried out depending on the parity of the copy. The entry-wise monotonicity of the b_p^ℓ in the parameter ℓ implies that, for every type $T \subsetneq [K]$, jobs appearing precisely in scenarios $S_k, k \in T$ (which correspond to entries of $\sigma(T)$ or $2^K + \sigma(T)$) are assigned to alternating parities of copies, and therefore to the two machines in a round-robin manner. Therefore, the total contribution of the jobs assigned by *all* copies $b_p^1, \dots, b_p^{\lambda_p}$ combined to the total disbalance does not exceed the size of a single copy b_p. To be more precise, we have for every $j \in [n]$ and $k \in [K]$:

$$\begin{aligned}&\max_{i\in\{1,2\}} |\{j' \in J_i \cap S_k \cap B_p : j' \leq j\}| - \min_{i\in\{1,2\}} |\{j' \in J_i \cap S_k \cap B_p : j' \leq j\}|\\ \leq &\max_{i\in\{1,2\}} |\{j' \in J_i \cap S_k \cap B_{p\ell(j)} : j' \leq j\}| - \min_{i\in\{1,2\}} |\{j' \in J_i \cap S_k \cap B_{p\ell(j)} : j' \leq j\}|\end{aligned}$$

where B_p resp. $B_{p\ell(j)}$ denotes the set of jobs that are assigned during the outer iteration p resp. both iterations $(p, \ell(j))$. The size of $B_{p\ell(j)}$ is bounded by $\|b_p\|_1$ by the construction of the algorithm.

Therefore, we have

$$\begin{aligned}\max\nolimits_{j\in[n],k\in[K]} \{&\max\nolimits_{i\in\{1,2\}} |\{j' \in J_i \cap S_k : j' \leq j\}| \\ &- \min\nolimits_{i\in\{1,2\}} |\{j' \in J_i \cap S_k : j' \leq j\}|\}\end{aligned} \tag{22}$$

$$\begin{aligned}\leq \sum\nolimits_{p=1}^{|B|} \max\nolimits_{j\in[n],k\in[K]} \{&\max\nolimits_{i\in\{1,2\}} |\{j' \in J_i \cap S_k \cap B_p : j' \leq j\}| \\ &- \min\nolimits_{i\in\{1,2\}} |\{j' \in J_i \cap S_k \cap B_p : j' \leq j\}|\}\end{aligned} \tag{23}$$

$$\begin{aligned}\leq \sum\nolimits_{p=1}^{|B|} \max\nolimits_{j\in[n],k\in[K]} \{&\max\nolimits_{i\in\{1,2\}} |\{j' \in J_i \cap S_k \cap B_{p\ell(j)} : j' \leq j\}| \\ &- \min\nolimits_{i\in\{1,2\}} |\{j' \in J_i \cap S_k \cap B_{p\ell(j)} : j' \leq j\}|\}\end{aligned} \tag{24}$$

$$\leq \sum_{p=1}^{|B|} \|b_p\|_1 \leq \sum_{p=1}^{|B|} 2 \cdot 2^K \cdot \|b_p\|_\infty \tag{25}$$

$$\leq \left(2^{2^{K+1}} K! + 1\right)^{2 \cdot 2^K} \cdot 2 \cdot 2^K \cdot 2^{2^{K+1}} K! \tag{26}$$

The last inequality holds by Claims 4 and 5. This inequality together with the reduction at the beginning finishes the proof.

5.3.2 Proof of Theorem 8

It remains to prove the statement of Theorem 8 for an arbitrary number *m* of machines. To deduce this result from the previous case, we use Algorithm 2 as a subroutine to apply on two machines $i_1, i_2 \in [m]$. This way, we obtain full disbalance bounded by a function *f*(*K*) when restricted to the machines i_1 and i_2 (cf. proof of Theorem 9). We call this subroutine *equalizing* and denote by $\text{equalize}(i_1, i_2)$ when applied to machines i_1 and i_2. We equalize two machines at a time and prove that for a suitable function value *g*(*K*) depending on the number *K* of scenarios as well as a suitable choice of pairs of machines to equalize, the procedure eventually terminates with the desired outcome.

Let $L_k := \frac{|S_k|}{m}$ be the average load of a machine for scenario $k \in [K]$. Then the procedure can be described by Algorithm 3.

Algorithm 3 Equalizing two machines

```
while there exists $i \in [m]$, $k \in [K]$ with $||J_i \cap S_k| - L_k| > 2K \cdot f(K)$ do
    if $|J_i \cap S_k| > L_k$ then
        equalize($i$, $\text{argmin}_{i' \in [m]} |J_{i'} \cap S_k|$)
    else
        equalize($i$, $\text{argmax}_{i' \in [m]} |J_{i'} \cap S_k|$)
    end if
end while
```

By construction, the algorithm outputs a solution with full disbalance at most $4K \cdot f(K)$ if it terminates, because then we have for any two machines $i_1, i_2 \in [m]$ and for any scenario $k \in [K]$:

$$\begin{aligned} ||J_{i_1} \cap S_k| - |J_{i_2} \cap S_k|| &= |(|J_{i_1} \cap S_k| - L_k) - (|J_{i_2} \cap S_k| - L_k)| \\ &\leq ||J_{i_1} \cap S_k| - L_k| + ||J_{i_2} \cap S_k| - L_k| \leq 2 \cdot 2K \cdot f(K) \end{aligned}$$

Therefore, it suffices to show that the procedure terminates.

Claim 7 The sum $\mathrm{L} := \sum_{i=1}^{m} \sum_{k=1}^{K} ||J_i \cap S_k| - L_k|$ strictly decreases at each iteration of the procedure.

Proof After the subroutine $\text{equalize}(i_1, i_2)$ is applied, all summands of L indexed by $i \notin \{i_1, i_2\}$ stay constant. Let k be the scenario because of which the subroutine was applied (i.e. one has $||J_{i_1} \cap S_\mathrm{k}| - L_\mathrm{k}| > 2K \cdot f(K)$ or

$||J_{i_2} \cap S_{\mathrm{k}}| - L_{\mathrm{k}}| > 2K \cdot f(K))$. Then the summands $||J_{i_1} \cap S_{\mathrm{k}}| - L_{\mathrm{k}}|$ resp. $||J_{i_2} \cap S_{\mathrm{k}}| - L_{\mathrm{k}}|$ were each decreased by at least $\frac{1}{2}||J_{i_1} \cap S_{\mathrm{k}}| - L_{\mathrm{k}}| - f(K)$ resp. $\frac{1}{2}||J_{i_2} \cap S_{\mathrm{k}}| - L_{\mathrm{k}}| - f(K)$. For $k' \neq \mathrm{k}$, the increase of each summand $||J_{i_1} \cap S_{k'}| - L_{k'}|$ resp. $||J_{i_2} \cap S_{k'}| - L_{k'}|$ is at most *f*(*K*). Therefore, the total decrease is at least

$$\begin{aligned}
&\frac{1}{2}||J_{i_1} \cap S_{\mathrm{k}}| - L_{\mathrm{k}}| - f(K) + \frac{1}{2}||J_{i_2} \cap S_{\mathrm{k}}| - L_{\mathrm{k}}| - f(K) - 2(K-1) \cdot f(K) \\
>&\frac{1}{2} \cdot 4K \cdot f(K) - 2K \cdot f(K) = 0,
\end{aligned}$$

showing the claim.

The claim implies that the procedure must terminate after finitely many steps as we have $\mathrm{L} \geq 0$ throughout the procedure.

Theorem 8 can be used for the general case as well, granted that an algorithm equalizing two machines exists, so that the problem reduces to bounding full disbalance on two machines. In turn, techniques from Theorem 9 can be adapted to the slightly generalized case where the number of *pairwise distinct* job weights w_j are bounded by a function *h*(*K*) depending on *K*. To that end, one can obtain a generalized bound for the final disbalance which involves the different weights. Furthermore, Algorithm 2 can be adapted to distribute jobs of a certain weight *w* under the assumption that all jobs with weight $w' \geq w$ are distributed in a way that can be extended to an optimal solution, which uses the well-known resolution theorem that describes a polyhedron as a Minkowski sum of a polytope and a polyhedral cone [30]. The challenge of this approach is to determine the number of jobs (per subset $S \subseteq [K]$ of scenarios they appear in) and a the used inductive technique does not exclude a disbalance linearly increasing in the number of distinct weights, which is why the assumption of $|\{w_j : j \in [n]\}| \leq h(K)$ is necessary.

One may wonder whether *g*(*K*) can be strengthened to be polynomial in *K*. In the next subsection we show that the full disbalance can be exponential in *K* even in the case of two machines and two distinct job weights.

5.4 Exponential Lower Bound on Disbalance

Let us turn to the lower bound construction. First, we consider the following related question. Suppose we are given a $0-1$ matrix $A \in \{0,1\}^{n \times K}$ such that every column in A sums to c. We now want to know whether there exists a proper $n' \times K$ submatrix A' of A such that each column in A' sums to $c' < c$. If this is not the case, we call matrix A unsplittable.

For every unsplittable matrix *A* such that every column sums to *c*, we can create an instance of MinAvgSTC such that the disbalance of the schedule is *c*. To see this, note that we may create an instance on 2 machines, consisting of one job for every row in *A* with weight 1, and a scenario for each column in *A*, where such a job *j* belongs to scenario *k* if and only if there is a 1 in the corresponding entry. Furthermore, we create another set of *c* jobs appearing in every scenario with weight $1-\epsilon$. For ϵ small enough in the optimal solution the jobs corresponding to the rows of *A* should be assigned to the same machine, say machine 1, and the c additional jobs to machine 2. However, the dynamic program will schedule the rows of A first and therefore after having assigned all jobs corresponding to the rows of A it has to create a disbalance of c, since machine 2 will have been assigned no jobs yet.

We can now construct a family of instances showing that the disbalance of the schedule is at least exponential in K. Let q be some natural number and let I, J be the $q \times q$ identity respectively the all ones matrix, and let H be $J - I$. We define the following $q^2 \times 2q$ matrix.

$$A_q^2 = \begin{pmatrix} I & J \\ J & H \\ \vdots & \vdots \\ J & H \end{pmatrix},$$

where the submatrix (J H) is repeated $q-1$ times. Now the columns sum to $q^2 - q + 1$, but no proper submatrix has columns summing to the same number. To see this, note that if we take a row from the top (I J), we must take all of them, just to balance out the first q columns. But then the first q columns contain in total $q^2 - q$ fewer ones than the last q columns. As every row in the bottom has exactly one zero, it takes all of the bottom rows to make up the difference.

We can use this insight to work up the example. Let $J_{r,n}$ be the $r \times n$ all ones matrix. Let $B_q^2 = J_{q^2,2q} - A_q^2$. Then B_q^2 has no proper submatrix with uniform column sums either, but has every column summing to $q-1$. Hence, the following matrix is again unsplittable:

$$A_q^3 = \begin{pmatrix} B_q^2 & J_{q^2,q} \\ J_{q,2q} & H \\ \vdots & \vdots \\ J_{q,2q} & H \end{pmatrix},$$

where the submatrix $(J_{q,2q} \quad H)$ is repeated $q^2 - 2q + 2$ times. Note that the number of ones in B_q^2 is equal to $2q^2 - 2q$, whereas the number of ones in $J_{q^2,q}$ equals q^3. Since the difference is a multiple of q, we can add an integer number of copies of $(J_{q,2q} \quad H)$ to obtain the unsplittable matrix A_q^3. This matrix has columns summing to $q^3 - 2q^2 + 3q - 1$. Repeating this construction will result in matrix A_q^t, for $t \geq 4$, for which the following theorem holds.

Theorem 10 Using the construction above, we obtain the unsplittable matrix A_q^t for $t \geq 2$, satisfying the following properties:

1. The number of columns equals tq,
2. The number of rows is a polynomial in q of degree t, where the leading coefficient equals 1,
3. The columns sum to a polynomial in q of degree t, where the leading coefficient equals 1.

Proof We will prove the theorem using induction on *t*. We have already shown that matrix A_q^2 satisfies the required properties. Now, we assume the theorem holds for matrix A_q^{t-1}. Let the number of rows of A_q^{t-1} be equal to $q^{t-1} + p_1(q)$, and let the column sum be equal to $q^{t-1} + p_2(q)$, where $p_1(q)$ and $p_2(q)$ are polynomials in q of degree $t-2$.

Let $B_q^{t-1} = J_{q^{t-1}+p_1(q),(t-1)q} - A_q^{t-1}$. Since A_q^{t-1} is unsplittable, so is B_q^{t-1}. Each column of B_q^{t-1} sums to

$$q^{t-1} + p_1(q) - (q^{t-1} + p_2(q)) = p_1(q) - p_2(q),$$

which is a polynomial of degree $t-2$. In our construction, we would like to add $J_{q^{t-1}+p_1(q),q}$ to the right of B_q^{t-1}, and add a number of copies of $(J_{q,(t-1)q} \quad H)$ to balance the column sums. The resulting matrix is called A_q^t.

Note that the number of ones in B_q^{t-1} equals $(t-1)q(p_1(q) - p_2(q))$, whereas the number of ones in $J_{q^{t-1}+p_1(q),q}$ equals $q^t + qp_1(q)$. Since the difference between these numbers is a multiple of *q*, we can actually add an integer number of copies of $(J_{q,(t-1)q} \quad H)$ to balance the columns. Now, matrix A_q^t has *tq* columns, and the number of rows is equal to

$$q^{t-1} + p_1(q) + q^t + qp_1(q) - (t-1)q(p_1(q) - p_2(q)),$$

which is a polynomial in *q* of degree *t*, where the leading coefficient equals 1. Similarly, each column of A_q^t sums to

$$p_1(q) - p_2(q) + q^t + qp_1(q) - (t-1)q(p_1(q) - p_2(q)),$$

which is a polynomial in *q* of degree *t*, where the leading coefficient equals 1. Hence, A_q^t also satisfies the properties stated in the theorem. □

Since A_q^t has *tq* columns summing to $\Omega(q^t)$, we obtain an $\Omega(q^{K/q})$ lower bound.

6 Consequences for Regret Scheduling

In the proof of Theorem 5, we applied our reduction by building very specific instances of MinMaxSTC. This allows us to conclude that MinMaxSTC on a restricted class of instances is also NP-hard.

Corollary 1 *Let an instance of MinMaxSTC with unit processing times be called scenario-symmetric if for any permutation of the scenarios, the sets of unnumbered jobs are identical as sets of tuples of weight and scenarios, i.e., for any permutation* $\pi : [K] \to [K]$*, we have*

$$\{(w_i, \{k \in [K] : i \in S_k\})\}_{i\in[n]} = \{(w_i, \{\pi(k) \in [K] : i \in S_k\})\}_{i\in[n]}.$$

MinMaxSTC restricted to scenario-symmetric instances is NP-hard.

Proof In the proof of Theorem 5, the instance in the reduction is scenario-symmetric because gray jobs appear in all three scenarios, while the three black and three white jobs in each block appear in scenarios $\{1,2\}, \{1,3\}, \{2,3\}$ respectively, which is invariant under permutations of the three scenarios. Hence any polynomial-time algorithm for MinMaxSTC on scenario-symmetric instances implies a polynomial-time algorithm for Partition-3. □

We can thus relate our model to robust min-max regret scheduling with arbitrary weights and unit processing times [6]. Indeed, the latter model seeks to minimize

$$\max_{k=1}^{K}\left(G(\sigma,k)-\min_{\tau:[n]\to[m]}G(\tau,k)\right),$$

while our model seeks to minimize $\max_{k=1}^{K}(G(\sigma,k))$, where $G(\tau,k)$ denotes the sum of completion times of the assignment τ in scenario k. In general, these two objective functions do not have a one-to-one correspondence. However, under the assumption that the scenarios are scenario-symmetrical, optimal solutions of the two models coincide.

Corollary 2 *Scheduling with regret on $m \geq 2$ machines, $K = 3$ scenarios and unit processing times is NP-hard.*

Moreover, the dynamic program used in the proof of Theorem 6 can also be used to solve min-max regret scheduling by first computing $\min_{\tau:[n]\to[m]}G(\tau,k)$ (where k is trivially the only scenario) and then finding assignments to match every possible value of the objective value. It is then immediate that Theorem 6 can be applied as well, proving that robust min-max regret scheduling admits an FPTAS.

One can also consider scheduling with regret with respect to the sum over scenarios, that is, the model minimizing

$$\sum_{k=1}^{K}\left(G(\sigma,k)-\min_{\tau:[n]\to[m]}G(\tau,k)\right)=\sum_{k=1}^{K}G(\sigma,k)-\sum_{k=1}^{K}\min_{\tau:[n]\to[m]}G(\tau,k).$$

Here, the latter term does not depend on the solution. Therefore, it immediately follows that the optima of this problem coincide with those of MinAvgSTC.

7 Conclusion and Open Problems

We hope that our results inspire interest in the intriguing field of scenario optimization problems. There are some obvious open questions that we left unanswered. For example, is it true that MinMax versions are always harder than MinAvg versions? For researchers interested in exact algorithms and fixed parameter tractability (FPT) results we have the following question. For the scheduling problems that we have studied so far within the scenario model, we have seen various exact polynomial time dynamic programming algorithms for a constant number of scenarios K, but always with K in the exponent of a function of the number of jobs or the number of machines. Can these results be strengthened to algorithms that are FPT in K? Or are these problems W[1]-hard? Similarly, researchers interested in approximation algorithms may wonder how approximability is affected by introducing scenarios into a problem. We have given some first results here, but it clearly is a research area that has so far remained virtually unexplored.

Another interesting variation of the MinAvg version is obtained by assigning probabilities to scenarios (i.e., a discrete distribution over scenarios) with the objective to minimize the expected total (weighted) completion time. The dynamic programs underlying the results of Theorems 4 and 7 easily generalize to this version, but Theorem 8 does not.

We see as a main challenge to derive structural insights why multiple (constant number of) scenario versions are sometimes as easy as their single scenario versions, like the MinAvg versions of linear programming or of the min-cut problem [31] or of the scheduling problem that we have studied here, and for other problems, such as MinMaxSTC, become harder or even NP-hard.

Acknowledgements We would like to thank Bart Litjens, Sven Polak, Lluís Vena and Bart Sevenster for providing a counterexample for a preliminary version of Conjecture 1, in which $g(K)$ was a linear function. In addition, we would like to thank the anonymous referees for their many valuable comments and suggestions. Alberto Marchetti-Spaccamela was supported by the ERC Advanced Grant 788893 AMDROMA "Algorithmic and Mechanism Design Research in Online Markets", and the MIUR PRIN project ALGADIMAR "Algorithms, Games, and Digital Markets". Ekin Ergen and Martin Skutella were supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy — The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689). Leen Stougie was supported by NWO Gravitation Programme Networks 024.002.003, and by the OPTIMAL project NWO OCENW.GROOT.2019.015.

Author Contributions All authors contributed to Sections 1, 2 and 7. All authors except E.E. contributed to Section 3, Section 4 (Theorem 4), Section 5.1 and Section 5.2 (Conjecture 1). An earlier version of Theorem 5 was proven by C.I. which was simplified and generalized by E.E. and M.S.; the current version of the text was written by E.E. Theorem 6 was proven by A.M.S., M.S. and L.S.; the text was written by E.E. The partial proof of the conjecture and the proofs of the relevant lemmata are joint work of E.E. and M.S., written by E.E. Section 5.2. was written by T.B. and Section 6 was written by E.E.
All authors did multiple reviews at multiple times during the course of the work.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Conway, R.W., Maxwell, W.L., Miller, L.W.: Theory of Scheduling. Addison-Wesley Publishing Company, Boston (1967)
2. Birge, J.R., Louveaux, F.: Introduction to Stochastic Programming. Springer, New York (2011)
3. Kleywegt, A.J., Shapiro, A., Homem-de-Mello, T.: The sample average approximation method for stochastic discrete optimization. SIAM J. Optim. **12**(2), 479–502 (2002)
4. Ben-Tal, A., Ghaoui, L.E., Nemirovski, A.: Robust Optimization. Princeton Series in Applied Mathematics, vol. 28. Princeton University Press, Princeton (2009)
5. Bertsimas, D., Jaillet, P., Odoni, A.R.: A priori optimization. Oper. Res. **38**(6), 1019–1033 (1990)
6. Shabtay, D., Gilenson, M.: A state-of-the-art survey on multi-scenario scheduling. European Journal of Operational Research. (2022)
7. Kouvelis, P., Yu, G.: Robust Discrete Optimization and Its Applications. Kluwer Academic Publishers, London (1997)
8. Immorlica, N., Karger, D., Minkoff, M., Mirrokni, V.S.: On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 691–700 (2004)
9. Adamczyk, M., Grandoni, F., Leonardi, S., Wlodarczyk, M.: When the optimum is also blind: a new perspective on universal optimization. In: Chatzigiannakis, I., Indyk, P., Kuhn, F., Muscholl, A. (eds.) 44th International Colloquium on Automata, Languages and Programming. LIPIcs, vol. 80, pp. 35:1–35:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2017)
10. van Ee, M., van Iersel, L., Janssen, T., Sitters, R.: A priori TSP in the scenario model. Discret. Appl. Math. **250**, 331–341 (2018)
11. Feuerstein, E., Marchetti-Spaccamela, A., Schalekamp, F., Sitters, R., van der Ster, S., Stougie, L., van Zuylen, A.: Minimizing worst-case and average-case makespan over scenarios. J. Sched. **20**, 545–555 (2017)
12. Yang, J., Yu, G.: On the robust single machine scheduling problem. J. Comb. Optim. **6**(1), 17–33 (2002)
13. Aloulou, M.A., Croce, F.D.: Complexity of single machine scheduling problems under scenario-based uncertainty. Oper. Res. Lett. **36**(3), 338–342 (2008)
14. Mastrolilli, M., Mutsanas, N., Svensson, O.: Single machine scheduling with scenarios. Theoret. Comput. Sci. **477**, 57–66 (2013)
15. Kasperski, A., Zieliński, P.: Single machine scheduling problems with uncertain parameters and the OWA criterion. J. Sched. **19**, 177–190 (2016)
16. Kasperski, A., Kurpisz, A., Zieliński, P.: Parallel machine scheduling under uncertainty. In: Advances in Computational Intelligence - 14th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, pp. 74–83. Springer, Berlin, Heidelberg (2012)

17. Albers, S., Janke, M.: Online makespan minimization with budgeted uncertainty. In: Lubiw, A., Salavatipour, M.R. (eds.) Algorithms and Data Structures - 17th Int. Symposium, WADS 2021. Lecture Notes in Computer Science, vol. 12808, pp. 43–56. Springer, Cham (2021)
18. Bougeret, M., Jansen, K., Poss, M., Rohwedder, L.: Approximation results for makespan minimization with budgeted uncertainty. Theory Comput. Syst. **65**(6), 903–915 (2021)
19. Håstad, J.: Some optimal inapproximability results. J. ACM **48**(4), 798–859 (2001)
20. Khot, S., Kindler, G., Mossel, E., O'Donnell, R.: Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? SIAM J. Comput. **37**(1), 319–357 (2007)
21. Schulz, A.S., Skutella, M.: Scheduling unrelated machines by randomized rounding. SIAM J. Discret. Math. **15**(4), 450–469 (2002)
22. Skutella, M.: Approximation and randomization in scheduling. PhD thesis, Technische Universität Berlin (1998)
23. Scheduling independent tasks to reduce mean finishing time: Bruno, J.L., Jr., E.G.C., Sethi, R. Commun. ACM **17**, 382–387 (1974)
24. Eastman, W.L., Even, S., Isaacs, I.M.: Bounds for the optimal scheduling of n jobs on m processors. Manage. Sci. **11**(2), 268–279 (1964)
25. Goemans, M.X., Williamson, D.P.: Two-dimensional gantt charts and a scheduling algorithm of Lawler. SIAM J. Discret. Math. **13**(3), 281–294 (2000)
26. Megow, N., Verschae, J.: Dual techniques for scheduling on a machine with varying speed. SIAM J. Discret. Math. **32**(3), 1541–1571 (2018)
27. Cho, W., Shmoys, D.B., Henderson, S.G.: SPT optimality (mostly) via linear programming. Oper. Res. Lett. **51**(1), 99–104 (2023)
28. Austrin, P., Hastad, J., Guruswami, V.: $(2+\epsilon)$-SAT is NP-hard. In: Proceedings of 55th Annual Symposium on Foundations of Computer Science, pp. 1–10 (2014). IEEE
29. Strang, G.: Introduction to Linear Algebra. Wellesley-Cambridge Press, Wellesley, MA (2016)
30. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons, Chichester (1986)
31. Armon, A., Zwick, U.: Multicriteria global minimum cuts. Algorithmica **46**(1), 15–26 (2006)

Authors and Affiliations

Thomas Bosman[1] · Martijn van Ee[2] · Ekin Ergen[3] · Csanád Imreh[4] · Alberto Marchetti-Spaccamela[5] · Martin Skutella[3] · Leen Stougie[1,6]

✉ Ekin Ergen
ergen@math.tu-berlin.de

Thomas Bosman
tbosman@gmail.com

Martijn van Ee
M.v.Ee.01@mindef.nl

Alberto Marchetti-Spaccamela
alberto@diag.uniroma.it

Martin Skutella
martin.skutella@tu-berlin.de

Leen Stougie
leen.stougie@cwi.nl

1 Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

2 Netherlands Defence Academy, Den Helder, The Netherlands

3 Technische Universität Berlin, Berlin, Germany

4 University of Szeged, Szeged, Hungary

5 Universitá di Roma "La Sapienza", Rome, Italy

6 CWI, Amsterdam, The Netherlands