

# Total Completion Time Scheduling Under Scenarios

Thomas Bosman<sup>1</sup>, Martijn van Ee<sup>2</sup>, Ekin Ergen<sup>3</sup>, Csanád Imreh<sup>4</sup>, Alberto Marchetti-Spaccamela<sup>5,6</sup>, Martin Skutella<sup>3</sup>, and Leen Stougie<sup>1,6,7</sup>

<sup>1</sup> Vrije Universiteit Amsterdam, The Netherlands  
tbosman@gmail.com

<sup>2</sup> Netherlands Defence Academy, Den Helder, The Netherlands  
M.v.Ee.01@mindef.nl

<sup>3</sup> Technische Universität Berlin, Germany  
ergen@math.tu-berlin.de, martin.skutella@tu-berlin.de

<sup>4</sup> Our co-author Csanád Imreh tragically passed away on January 5th, 2017.

<sup>5</sup> Università di Roma “La Sapienza”, Italy  
alberto@diag.uniroma.it

<sup>6</sup> CWI, Amsterdam, The Netherlands,

<sup>7</sup> Erable, INRIA, France  
leen.stougie@cwi.nl

**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 interesting generalizations of this classical problem involving 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 [11].

We study an interesting generalization of this classical scheduling problem where the input in addition specifies a set of  $K$  scenarios  $\mathcal{S} = \{S_1, \dots, S_K\}$ , with  $S_k \subseteq J$ ,  $k = 1, \dots, K$ . 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. In particular, 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. In fact, scenarios have been introduced to model discrete distributions over parameter values in Stochastic Programming [8], or as samples in Sampling Average Approximation algorithms for stochastic problems with continuous distributions over parameter values [20]. In Robust Optimization [6], 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 [7]: 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 [26]. 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 [21].

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 [16,21], and the metric minimum spanning tree problem with a scenario defining the subset of vertices to be in the tree [7]. Scenario versions of NP-hard combinatorial optimization problems were also considered in the literature such as, for example, set cover [1] and the traveling salesperson problem [28].

**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 refer to the survey [26] and references therein for an overview of scheduling under scenarios.

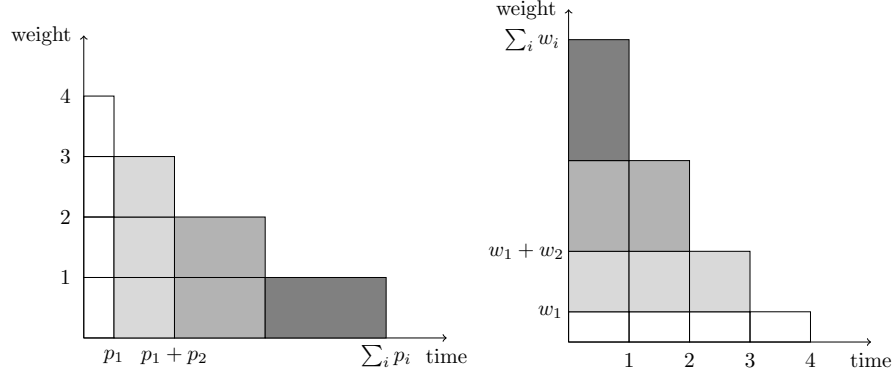
Closest to the problems considered in this paper is the work of Feuerstein et al. [13] who also consider scenarios given by subsets of jobs and develop approximation algorithms as well as non-approximability results for minimizing the makespan on identical parallel machines, both for the MinMax and the 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 [29], 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 [3] present algorithmic and computational complexity results for several single machine scheduling problems. Mastrolilli, Mutsanas, and Svensson [22] 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 [18] 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 [17] 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 [2] as well as Bougeret, Jansen, Poss, and Rohwedder [9] 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  $\Gamma$  jobs fail and require additional processing time. The considered objective function is to minimize the makespan: [9] proposes approximate algorithms for identical and unrelated parallel machines while [2] analyses online algorithms in this setting.

## 2 Preliminaries

We start by defining the problem formally, denoting the set of integers  $\{1, \dots, \ell\}$  simply by  $[\ell]$ . We are given a set of jobs  $J = [n]$ , machines  $M = [m]$ , non-negative job processing times  $p_j$ ,  $j \in [n]$ , and scenarios  $\mathcal{S} = \{S_1, \dots, S_K\}$ , where  $S_k \subseteq J$ ,



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

$k \in [K]$ . We assume that the jobs are ordered by *non-increasing* processing times  $p_1 \geq p_2 \geq \dots \geq p_n$ .<sup>8</sup> The task is to find a machine assignment, that is, a map  $\varphi: [n] \rightarrow [m]$ , or equivalently, a partitioning of the jobs  $J_1, \dots, J_m$  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  $j \geq j'$ , and the contribution of jobs in  $J_i$  to the total completion in scenario  $k \in [K]$  is thus:

$$\sum_{j' \in J_i \cap S_k} \sum_{j \in J_i \cap S_k: j' \leq j} p_j = \sum_{j \in J_i \cap S_k} p_j \cdot |\{j' \in J_i \cap S_k : j' \leq j\}| \quad (1)$$

**Observation 1 ([12])** *The above total unweighted completion time problem is equivalent to the total weighted completion time with jobs of unit length of weight  $w_j := p_j$  that are processed in reverse order, i.e., in order of non-increasing weights.*

Indeed, this equivalence between total completion time for unweighted jobs scheduled in SPT order and total *weighted* completion time for unit length jobs scheduled in order of non-decreasing weight was first observed by Eastman, Even, and Issacs [12]. The idea behind the equivalence (1) is best seen from a so-called 2-dimensional Gantt-chart; see Figure 1 and the work of Goemans and Williamson [14], Megow and Verschae [23], or Cho, Shmoys, and Henderson [10].

In the introduction of this paper we prefer to present our results in terms of the more commonly known unweighted version, but for the remainder of this paper it is somewhat easier to argue about the weighted unit-processing time

<sup>8</sup> In view of the SPT rule, this ordering might seem counterintuitive. But it turns out to be convenient as we argue below in Observation 1.

	$m = 2$	$m \in O(1)$	$m$ 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 $w_j \in O(1)$
$K$ part of input	strongly NP-hard [13], no $(2 - \varepsilon)$ -approx [13], 2-approx	strongly NP-hard [13]	strongly NP-hard [13]

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

version; hence the ordering of jobs introduced above. The objective of (MinMaxSTC) is then to minimize

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

whereas MinAvgSTC aims to minimize

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

In the sequel in the MinAvgSTC we neglect the constant  $1/K$ -term and minimize the sum over all scenarios.

### 3 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 problem is APX-hard, i.e., there is no PTAS, unless  $P=NP$ ; see Section 4. 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 4 we present a simple algorithm that constructs an ‘ideal’ schedule for the entire set of jobs simultaneously minimizing the total

	$m = 2$	$m \in O(1)$	$m$ part of input
$K = 2$	poly	poly	poly
$3 \leq K \in O(1)$	poly	poly	poly if $w_j \in O(1)$
$K$ part of input	strongly NP-hard [13], no PTAS [15,19], $5/4$ -approx [25,27]	strongly NP-hard [13], no PTAS [15,19], $(3/2-1/2m)$ -approx [25,27]	strongly NP-hard [13], no PTAS [15,19], $(3/2-1/2m)$ -approx [25,27]

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

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 5. 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. But then again, we present a more sophisticated dynamic program that solves the problem efficiently if all job processing times are bounded by a constant.

MinAvgSTC with constantly many scenarios  $K \geq 3$  is studied in Section 6. 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 always 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 unit job processing times. In this case we obtain an efficient algorithm with running time  $m^{h(K)} \cdot \text{poly}(n)$  for some function  $h$ .

Several of our results, in particular for MinMaxSTC, can be generalized to the Min–Max Regret framework; see Appendix 8.6 for details. Moreover, due to space restrictions, most proofs are deferred to the appendix.

## 4 Scheduling Under Arbitrary $K$ is Hard, but $K = 2$ is Easy

### 4.1 NP-hardness for 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 of 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 [13] can be adapted to similar inapproximability results for our problems. Also these results already hold for unweighted unit length jobs. The proof is deferred to Appendix 8.1.

**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.*

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; see Appendix 8.1 for a short proof.

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

### 4.2 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 2.** *For the MinMaxSTC and the MinAvgSTC problem with two scenarios, 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.*

*Proof.* We show how to assign the jobs in *non-increasing* order of their weights in order to be optimal for both scenarios. As mentioned above, we want to assign

the next job to a machine that, in each scenario, belongs to the least loaded machines in terms of the number of jobs already assigned to it. 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.

Initially, both vectors  $s_1$  and  $s_2$  are zero vectors, since no jobs have been assigned yet. When assigning job  $j$ , let  $\mu_k$  be the lowest entry (lowest numbered machine) equal to zero in vector  $s_k$ . Moreover, let  $\nu_k$  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  $\mu_1$  and increase  $\mu_1$  by 1.
- If  $j \in S_2 \setminus S_1 \rightarrow$  assign  $j$  to machine  $\mu_2$  and increase  $\mu_2$  by 1.
- If  $j \in S_1 \cap S_2 \rightarrow$  assign  $j$  to machine  $\nu_1 = \nu_2$  and decrease both  $\nu_1$  and  $\nu_2$  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  $(1, \dots, 1, 0, \dots, 0)$ . Reset  $\mu_1 = 1$ ,  $\nu_1 = m$ ,  $\mu_2 = \mu_2 + m - \nu_2$ , and  $\nu_2 = m$ . Do analogously if  $s_2 = \mathbb{1}$ .

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  $\nu_1$  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.  $\square$

## 5 The MinMax Version

### 5.1 Constant number of machines

In view of Theorem 2, the next theorem establishes a complexity gap for MinMaxSTC when going from two to three scenarios.

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

The proof can be found in Appendix 8.2. It reduces a variant of Partition, in which one is to partition a set of numbers into three sets of equal sum instead



of two. This reduction establishes that MinMaxSTC is weakly NP-hard. For the case  $m \in O(1)$ , 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. Details are given in Appendix 8.2.

**Theorem 4.** *There exists a pseudopolynomial algorithm as well as a fully polynomial time approximation scheme for MinMaxSTC on a constant number of machines with a constant number of scenarios.*

The dynamic program runs in time  $O(m(m^2n^3W)^{mK})$ , where  $W$  denotes the largest (integer) job weight. The rounding for FPTAS redefines this weight to be  $1 + mn^2/\varepsilon$ , i.e., the runtime of the FPTAS is in  $O(m(m^2n^3(mn^2/\varepsilon))^{mK})$ , indeed polynomial in  $n$  and  $1/\varepsilon$  assuming that  $m$  and  $K$  are constant.

An immediate consequence of the reduction that yields Theorem 3 is the NP-hardness of the robust version of scheduling with regret, as mentioned in [26], even when it is restricted to the parallel machine case. Similarly, the FPTAS given in Theorem 4 can easily be adapted to this model. A more elaborate discussion on the relations between our model and the scheduling with regret model can be found in Appendix 8.6.

## 5.2 Any number of machines

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 be easily generalized to the case of weights bounded by some constant. The DP leading to the following theorem is based on enumeration of machine configurations.

**Theorem 5.** *If the number of scenarios  $K$  is constant, and all jobs have unit weights, then MinMaxSTC and MinAvgSTC can be solved to optimality in polynomial time on any number of machines.*

The proof, which can be found in Appendix 8.3, suggests an algorithm with runtime  $O(mn^{2(2^K+K)})$ .

## 6 The MinAvg Version

By Theorem 2, MinAvgSTC is solvable in polynomial time in the case of two scenarios and by Theorem 5 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 (DP) that solves MinAvgSTC in polynomial time for any constant number of scenarios if there is also a constant number of machines; see Section 6.1.

In Section 6.2, 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, it is not clear yet if the techniques carry over to the more general case of job weights bounded by a constant.

### 6.1 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 higher weight, 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 6.** *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 DP, 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  $\ell$ . Equivalently,  $X(j-1, \ell)$  is the matrix obtained from  $X(j)$  by diminishing all positive entries in row  $\ell$  of  $X(j)$  by 1. It follows that  $X(j-1, \ell)$  has entries  $x_{ik}(j-1)$  that satisfy  $x_{\ell k}(j-1) = x_{\ell k}(j) - \mathbf{1}_{j \in S_k}$  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  $f_0(\mathbf{0}) = 0$  (where  $\mathbf{0}$  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  $(n+1)^{m \times K}$ , which, because  $m$  and  $K$  are constants, implies that the DP runs in polynomial time.  $\square$

The proof implies that the problem can be solved in time  $O(mKn^{mK+1})$ , which is polynomial given that  $m$  and  $K$  are constants in this particular case.

## 6.2 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 5, 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  $m^{h(K)} \cdot n$  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.* MinAvgSTC has an optimal solution such that for every scenario  $k \in [K]$  and each  $j \in [n]$ , the  $j$  largest jobs 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

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

We call the term on the left-hand side *full disbalance* of the schedule. To adjust the DP to run in polynomial time for any number of machines under the conjecture, we first observe that to compute a DP 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  $\ell$ , it means that in the partial schedule  $\ell_k$  jobs have been scheduled on this

machine in scenario  $k$ . We then simply represent a partial schedule by storing the number of machines that have configuration  $\ell$ . We consider the space of *states* that say how many machines have a certain configuration. We can further compress this space by storing the smallest number of jobs on any machine in a given scenario separately. That is

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

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

$$y_\ell = \sum_{i | x_{ik} - z_k = \ell_k, \quad k \in [K]} 1, \quad \ell \in \mathcal{C}.$$

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 time algorithm. Since the DP-computation for each state in each phase (job assignment) is  $O(m)$  and there are  $n$  consecutive job assignments in SPT-order in the DP, 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, 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.

**Theorem 7.** *Conjecture 1 holds for unit weights and unit processing times, where the jobs are given by an arbitrary (but fixed) order and the number  $m$  of machines is part of the input.*

The proof can be found in Appendix 8.4.

One may wonder whether  $g(K)$  can be strengthened to be polynomial in  $K$ . In the case of arbitrary weights, one can establish exponential lower bounds, for which we refer to Appendix 8.5.

## 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 DPs underlying the results of Theorems 5 and 6 easily generalize to this version. But Theorem 7 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 [4] 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. Moreover, we would like to thank the anonymous referees for their valuable feedback 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.

## References

1. Marek Adamczyk, Fabrizio Grandoni, Stefano Leonardi, and Michal Włodarczyk. When the optimum is also blind: a new perspective on universal optimization. In *44th International Colloquium on Automata, Languages and Programming, ICALP*, volume 80 of *LIPICs*, pages 35:1–35:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
2. Susanne Albers and Maximilian Janke. Online makespan minimization with budgeted uncertainty. In Anna Lubiw and Mohammad R. Salavatipour, editors, *Algo-*

- rithms and Data Structures - 17th Int. Symposium, WADS 2021*, volume 12808 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 2021.
3. Mohamed Ali Aloulou and Federico Della Croce. Complexity of single machine scheduling problems under scenario-based uncertainty. *Oper. Res. Lett.*, 36(3):338–342, 2008.
  4. Amitai Armon and Uri Zwick. Multicriteria global minimum cuts. *Algorithmica*, 46(1):15–26, 2006.
  5. Per Austrin, Johan Hastad, and Venkatesan Guruswami.  $(2 + \epsilon)$ -SAT is NP-hard. In *Proceedings of 55th Annual Symposium on Foundations of Computer Science*, pages 1–10. IEEE, 2014.
  6. Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*, volume 28 of *Princeton Series in Applied Mathematics*. Princeton University Press, 2009.
  7. Dimitris Bertsimas, Patrick Jaillet, and Amedeo R. Odoni. A priori optimization. *Operations Research*, 38(6):1019–1033, 1990.
  8. John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.
  9. Marin Bougeret, Klaus Jansen, Michael Poss, and Lars Rohwedder. Approximation results for makespan minimization with budgeted uncertainty. *Theory Comput. Syst.*, 65(6):903–915, 2021.
  10. Woo-Hyung Cho, David B. Shmoys, and Shane G. Henderson. SPT optimality (mostly) via linear programming. *Oper. Res. Lett.*, 51(1):99–104, 2023.
  11. Richard W. Conway, William L. Maxwell, and Louis W. Miller. *Theory of Scheduling*. Addison-Wesley Publishing Company, 1967.
  12. Willard L. Eastman, Shimon Even, and I. M. Isaacs. Bounds for the optimal scheduling of  $n$  jobs on  $m$  processors. *Management Science*, 11(2):268–279, 1964.
  13. Esteban Feuerstein, Alberto Marchetti-Spaccamela, Frans Schalekamp, René Sitters, Suzanne van der Ster, Leen Stougie, and Anke van Zuylen. Minimizing worst-case and average-case makespan over scenarios. *Journal of Scheduling*, pages 1–11, 2016.
  14. Michel X. Goemans and David P. Williamson. Two-dimensional gantt charts and a scheduling algorithm of Lawler. *SIAM J. Discret. Math.*, 13(3):281–294, 2000.
  15. Johan Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.
  16. Nicole Immorlica, David Karger, Maria Minkoff, and Vahab S Mirrokni. 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*, pages 691–700. Society for Industrial and Applied Mathematics, 2004.
  17. Adam Kasperski, Adam Kurpisz, and Pawel Zieliński. Parallel machine scheduling under uncertainty. In *Advances in Computational Intelligence - 14th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 74–83. Springer, 2012.
  18. Adam Kasperski and Pawel Zieliński. Single machine scheduling problems with uncertain parameters and the OWA criterion. *Journal of Scheduling*, 19:177–190, 2016.
  19. Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O’Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.

20. Anton J Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002.
21. Panos Kouvelis and Gang Yu. *Robust discrete optimization and its applications*. Kluwer Academic Publishers, 1997.
22. Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. Single machine scheduling with scenarios. *Theoretical Computer Science*, 477:57–66, 2013.
23. Nicole Megow and José Verschae. Dual techniques for scheduling on a machine with varying speed. *SIAM Journal on Discrete Mathematics*, 32(3):1541–1571, 2018.
24. Alexander Schrijver. *Theory of Linear and Integer programming*. Wiley-Interscience, 1986.
25. Andreas S. Schulz and Martin Skutella. Scheduling unrelated machines by randomized rounding. *SIAM J. Discret. Math.*, 15(4):450–469, 2002.
26. Dvir Shabtay and Miri Gilenson. A state-of-the-art survey on multi-scenario scheduling. *European Journal of Operational Research*, 2022.
27. M. Skutella. *Approximation and Randomization in Scheduling*. PhD thesis, Technische Universität Berlin, 1998.
28. Martijn van Ee, Leo van Iersel, Teun Janssen, and René Sitters. A priori tsp in the scenario model. *Discrete Applied Mathematics*, 250:331–341, 2018.
29. Jian Yang and Gang Yu. On the robust single machine scheduling problem. *Journal of Combinatorial Optimization*, 6(1):17–33, 2002.

## 8 Appendix

### 8.1 Proofs of Propositions 1 and 2

For Proposition 1 we give sketches of the proof, because of similarly proven inapproximability results in [13].

*Proof (of Proposition 1: Sketches).* For MinMaxSTC we can use essentially the same reduction as in the proof of Theorem 1 in [13]. Consider a set of scenarios with  $2\ell + 1$  jobs each, where in each scenario the jobs can be partitioned in a perfectly balanced way. By the hardness of hypergraph balancing [5], it is NP-hard to find a solution where none of the scenarios puts all jobs to one of the machines. In a balanced solution the cost is  $\sum_{i=1}^{\ell} i + \sum_{i=1}^{\ell+1} i = (\ell + 1)^2$ . If we put all jobs on the same machine then the cost is  $\sum_{i=1}^{2\ell+1} i = (2\ell + 1)(\ell + 1)$ .

For the MinAvg version we adapt the reduction in the proof of Theorem 6 in [13] 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 jobs. If the scenario is in the cut then the cost is 2, if it is not in the cut then the jobs are on the same machine and the cost is 3. Thus the objective value is 3 times the number of edges minus the size of the cut. 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 known inapproximability of MAX CUT [15,19].  $\square$

*Proof (of Proposition 2).* The result is an immediate consequence of the following well known approximation result for the classical machine scheduling problem without scenarios: If all jobs are assigned to machines independently and uniformly at random, the expected total weighted completion time of the resulting schedule is at most a factor  $3/2 - 1/2m$  away from the optimum; see, e.g., [25,27]. In our scenario scheduling model, this upper bound holds, in particular, for each single scenario which yields a randomized  $(3/2 - 1/2m)$ -approximation algorithm by linearity of expectation. Furthermore, this algorithm can be derandomized using standard techniques.  $\square$

## 8.2 Proofs of Theorems 3 and 4

*Proof (Proof of Theorem 3).* We reduce Partition-3 (Partition into 3 subsets instead of 2) to MinMaxSTC. Let  $\{a_1, \dots, a_n\}$  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  $(2m+2) \cdot n$  jobs, grouping them in (disjoint) subsets  $T_1, \dots, T_n$  of size  $2m+2$ . For  $j = 1, \dots, n$ , we define the weights of the jobs in  $T_j$  as follows:

1. Three of the jobs have weight  $Q_j$ , where  $(Q_j)_{j \in \{1, \dots, n\}}$  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.
2. 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.
3. 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  $T_j$  are executed before those in  $T_{j+1}$ .
2. Any optimal solution schedules two jobs from each phase on each machine for each scenario.

*Claim.* For  $Q_l > 4mn^2 a_{\max} + \sum_{j=l+1}^n m \cdot (4j-1) \cdot Q_j$ , 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  $|S_i \cap T_j| = 2m$  for each  $i \in \{1, 2, 3\}$  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  $T_j$  to have finishing times  $2j-1$  and  $2j$ . Since the weights of the jobs in  $T_j$  are upper bounded by  $Q_j + a_j$ , such solutions cost at most



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

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, the two machines are assigned two jobs from  $T_j$  each for  $j < l$ ; and there exists a scenario where one machine is assigned at least three jobs from  $T_l$ . For this scenario, the machine with three jobs contributes to the weight by at least

$$\sum_{j=1}^{l-1} ((2j-1) + 2j) \cdot Q_j + (2l-1) \cdot Q_l + 2l \cdot Q_l + (2l+1) \cdot Q_l. \quad (3)$$

Here, we take the sum only over elements of  $T_1 \cup \dots \cup T_l$ . The last summand corresponds to the third job from  $T_l$  that has finishing time  $2l+1$  by the minimality of  $l$ . Similarly, the contribution of the other machines and jobs in  $T_1 \cup \dots \cup T_{l-1}$  is at least  $(m-1) \cdot \sum_{j=1}^{l-1} ((2j-1) + 2j) \cdot Q_j$ . On the other hand, we know by  $|S_i \cap T_j| = 2m$  that there must be  $2m-3$  other jobs in this scenario. At most  $m-1$  of these jobs can be completed at time  $2l-1$  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

$$((m-1) \cdot (2l-1) + (m-2) \cdot 2l) \cdot Q_l = (4ml - 6l - m + 1) \cdot Q_l \quad (4)$$

to the weight. Considering the machine loaded with at least three jobs (lower bounded by (3)) as well as the other machines (lower bounded by (4)), the weight of the violating schedule is at least

$$\sum_{j=1}^{l-1} m \cdot (4j-1) \cdot Q_j + ((4ml - 6l - m + 1) + 6l) \cdot Q_l \quad (5)$$

$$= \sum_{j=1}^l m \cdot (4j-1) \cdot Q_j + Q_l \quad (6)$$

$$> \sum_{j=1}^l m \cdot (4j-1) \cdot Q_j + 4mn^2 a_{\max} + \sum_{j=l+1}^n m \cdot (4j-1) \cdot Q_j \quad (7)$$

$$> \sum_{j=1}^n m \cdot (4j-1) \cdot (Q_j + a_{\max}). \quad (8)$$

Comparing (2) and (8) yields a contradiction, finishing the proof of the claim.  $\square$

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 8.2. One can further

verify that e.g. the sequence  $Q_j = (4mna_{\max})^{n-j}$  satisfies the inequality in Claim 8.2, 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  $T_j$  correspond to a decomposition of the schedule into what we refer to as *blocks*, in which elements of  $T_j$  are completed in time  $2j - 1$  resp.  $2j$  in any scenario. Therefore the contribution of the elements of  $T_j$  to the sum of completion times is independent of those in  $T_{j'}$  with  $j' \neq j$ .

Before continuing with the reduction, we make some observations about the  $2k - 4$  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 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  $2j - 1$ , 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  $j$ -th block contributes

$$\begin{aligned} & (2j - 1) \cdot (Q_j + a_j) + 2j \cdot (Q_j + a_j) + (2j - 1) \cdot Q_j + 2j \cdot Q_j \\ &= (8j - 2) \cdot Q_j + (4j - 1) \cdot a_j. \end{aligned}$$

In the scenarios  $S_2$  and  $S_3$  this contribution is

$$2 \cdot (2j - 1) \cdot (Q_j + a_j) + 2 \cdot 2j \cdot Q_j = (8j - 2) \cdot Q_j + (4j - 2) \cdot a_j.$$

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

For  $i \in \{1, 2, 3\}$  let  $A_i$  be the set of blocks  $j$  that cost more in the  $i$ -th scenario. For instance, the block in Figure 2 is an element of  $A_1$ . We immediately observe that  $A_1 \dot{\cup} A_2 \dot{\cup} A_3 = \{1, \dots, n\}$ . This yields for  $c(J') := \sum_{j \in J'} c_j$  given  $J' \subseteq [n]$  that

$$c(S_i) = \sum_{j=1}^n c(T_j \cap S_i) = \sum_{j=1}^n ((8j - 2) \cdot Q_j + (4j - 2) \cdot a_j) + \sum_{j \in A_i} a_j.$$

Since  $c(S_1) + c(S_2) + c(S_3) = 3 \sum_{j=1}^n ((8j - 2) \cdot Q_j + (4j - 2) \cdot a_j) + \sum_{j=1}^n a_j$ , we have

$$\max_{i=1,2,3} c(S_i) \geq \sum_{j=1}^n ((8j - 2) \cdot Q_j + (4j - 2) \cdot a_j) + \frac{1}{3} \sum_{j=1}^n a_j.$$

$M_1$	<table><tr><td>1, 2</td><td>1, 3</td><td>2, 3</td></tr></table>	1, 2	1, 3	2, 3
1, 2	1, 3	2, 3		
$M_2$	<table><tr><td>2, 3</td><td>1, 2</td><td>1, 3</td></tr></table>	2, 3	1, 2	1, 3
2, 3	1, 2	1, 3		
$M_3$	<table><tr><td>1, 2, 3</td><td>1, 2, 3</td></tr></table>	1, 2, 3	1, 2, 3	
1, 2, 3	1, 2, 3			
	$\vdots \quad \vdots$			
$M_k$	<table><tr><td>1, 2, 3</td><td>1, 2, 3</td></tr></table>	1, 2, 3	1, 2, 3	
1, 2, 3	1, 2, 3			

$M_1$	<table><tr><td>1, 2</td><td>1, 3</td></tr></table>	1, 2	1, 3
1, 2	1, 3		
$M_2$	<table><tr><td>1, 2</td><td>1, 3</td></tr></table>	1, 2	1, 3
1, 2	1, 3		
$M_3$	<table><tr><td>1, 2, 3</td><td>1, 2, 3</td></tr></table>	1, 2, 3	1, 2, 3
1, 2, 3	1, 2, 3		
	$\vdots \quad \vdots$		
$M_k$	<table><tr><td>1, 2, 3</td><td>1, 2, 3</td></tr></table>	1, 2, 3	1, 2, 3
1, 2, 3	1, 2, 3		

$S_1$

<table><tr><td>1, 2</td><td>2, 3</td></tr></table>	1, 2	2, 3
1, 2	2, 3	
<table><tr><td>2, 3</td><td>1, 2</td></tr></table>	2, 3	1, 2
2, 3	1, 2	
<table><tr><td>1, 2, 3</td><td>1, 2, 3</td></tr></table>	1, 2, 3	1, 2, 3
1, 2, 3	1, 2, 3	
$\vdots \quad \vdots$		
<table><tr><td>1, 2, 3</td><td>1, 2, 3</td></tr></table>	1, 2, 3	1, 2, 3
1, 2, 3	1, 2, 3	

$S_2$

<table><tr><td>1, 3</td><td>2, 3</td></tr></table>	1, 3	2, 3
1, 3	2, 3	
<table><tr><td>2, 3</td><td>1, 3</td></tr></table>	2, 3	1, 3
2, 3	1, 3	
<table><tr><td>1, 2, 3</td><td>1, 2, 3</td></tr></table>	1, 2, 3	1, 2, 3
1, 2, 3	1, 2, 3	
$\vdots \quad \vdots$		
<table><tr><td>1, 2, 3</td><td>1, 2, 3</td></tr></table>	1, 2, 3	1, 2, 3
1, 2, 3	1, 2, 3	

$S_3$

**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.

This inequality is tight if and only if there is a partition of  $\{1, \dots, n\}$  into the three sets  $A_1, A_2, A_3$  such that  $\sum_{j \in A_1} a_j = \sum_{j \in A_2} a_j = \sum_{j \in A_3} a_j = \frac{1}{3} \sum_{j=1}^n a_j$ , i.e. if the set  $a_1, \dots, a_n$  is a YES-instance of Partition-3.  $\square$

Next, we prove the FPTAS for MinMaxSTC on a constant number of machines and a constant number of scenarios. To this end, we first find a pseudopolynomial algorithm, to which the standard rounding techniques are applied.

*Proof (Proof of Theorem 4).* In order to develop a pseudopolynomial algorithm, we first observe that the contribution of the  $i$ -th machine to the objective function value of MinMaxSTC is bounded by

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

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, the dynamic program stores assignments

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

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

$$\sum_{j \in J_i \cap S_k} w_j \cdot |\{j' \in J_i \cap S_k : j' \leq j\}| = z_{ik} \quad (9)$$

and

$$|\{j' \in J_i \cap S_k : j' \leq n'\}| = y_{ik} \quad (10)$$

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 evidently given by the matrices  $Y$  and  $Z$ .

The parameters  $y_{ik}$  are bounded by  $n$  and  $z_{ik}$  can each be bounded by  $n^2W+1$  because we are interested in an optimal schedule. Note that given matrices  $Y \in [n]^{m \times K}$ ,  $Z \in \{1, \dots, n^2W\}^{m \times K}$ , 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  $\varphi, \varphi': [n'] \rightarrow [m]$  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  $\Phi^n[Y, Z]: [n] \rightarrow [m]$ , could use either of  $\varphi$  and  $\varphi'$  as partial assignments, therefore 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

$$y_{ik} = \begin{cases} 1 & i = i', 1 \in J_k \\ 0 & \text{else} \end{cases} \quad (11)$$

and

$$z_{ik} = \begin{cases} w_1 & i = i', 1 \in J_k \\ 0 & \text{else} \end{cases} \quad (12)$$

for a fixed  $i' \in [m]$ . In this case, the partial assignment  $\Phi^1[YZ]: [1] \rightarrow [m]$  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

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

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 (10) and (9).

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

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  $Y = (y_{ik})_{i \in [m], k \in [K]}$  and  $Z = (z_{ik})_{i \in [m], k \in [K]}$  if there exists an  $m' \leq m$  such that the map

$$\Phi^{n'-1}[Y'Z'] :=: [n'-1] \rightarrow [m] \quad (13)$$

where the matrices  $Y'$  and  $Z'$  are defined as

$$Y' = \begin{bmatrix} y_{11} & \dots & y_{1K} \\ \vdots & & \vdots \\ y_{m'1} - |\{n'\} \cap S_1| & \dots & y_{m'1} - |\{n'\} \cap S_K| \\ \vdots & & \vdots \\ y_{m1} & \dots & y_{mK} \end{bmatrix} \quad (14)$$

and

$$Z' = \begin{bmatrix} z_{11} & \dots & z_{1K} \\ \vdots & & \vdots \\ z_{m'1} - w_{n'} \cdot y_{m'1} \cdot |\{n'\} \cap S_1| & \dots & z_{m'1} - w_{n'} \cdot y_{m'1} \cdot |\{n'\} \cap S_K| \\ \vdots & & \vdots \\ m1 & \dots & z_{mK} \end{bmatrix} \quad (15)$$

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

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

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. The recursion base  $n = 1$  is given in (12).

Note that deciding whether  $m'$  exists takes linearly many calls in  $m$  by linear search. Moreover, there are at most polynomially many different maps in  $n$ ,  $m$  and  $W$ .

Given the dynamic program, we can solve the scheduling problem as follows: Starting with  $y_{\max} = z_{\max} = 1$  and incrementing upwards, 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  $\Phi^n[YZ]$  has nontrivial domain. Once such a matrix  $Z$  is found, we output the schedule  $\Phi^n[YZ]$ . Since there are polynomially many matrices  $[YZ]$  in  $n$  and  $W$ , this yields a pseudopolynomial algorithm.

We next show how to turn the pseudopolynomial algorithm into an FPTAS.

Let  $\mathcal{I} = (w_1, \dots, w_n, S_1, \dots, S_K)$  be an instance of MinMaxSTC and  $\varepsilon > 0$ . For a number  $\rho$  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  $\mathcal{I}' = (w'_1, \dots, w'_n, S_1, \dots, S_K)$  and output the (exact) solution given by this algorithm for the instance  $\mathcal{I}'$ . Let  $J'_1, \dots, J'_m$  be the partitioning of the jobs given by an optimal schedule of the instance  $\mathcal{I}'$ . Let  $\text{alg}$  denote the value of the output of our proposed FPTAS that returns the optimal solution for the instance  $\mathcal{I}'$ , and  $\text{opt}$  the optimal value for the instance  $\mathcal{I}$ . Then we have

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

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

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

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

$$= \text{opt} + \rho \cdot \max_{k \in [K]} \sum_{i=1}^m \sum_{j \in J_i \cap S_k} |\{j' \in J_i \cap S_k : j' \leq j\}| \leq \text{opt} + \rho mn^2 \quad (20)$$

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

$$\frac{\text{alg}}{\text{opt}} < 1 + \frac{\rho mn^2}{\text{opt}} \leq 1 + \frac{\rho mn^2}{W}$$

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

$$w'_j = \left\lceil \frac{w_j mn^2}{W\varepsilon} \right\rceil \leq \frac{mn^2}{\varepsilon} + 1,$$

therefore the algorithm has running time polynomial in  $m$ ,  $n$  and  $\frac{1}{\varepsilon}$ .  $\square$

### 8.3 Proof of Theorem 5

We say that job  $j \in J$  is of *type*  $T = \{k \in [K] \mid j \in S_k\}$ . For  $T \subseteq [K]$ , the subset of jobs of type  $T$  is denoted by  $J(T)$  and their number by  $n_T := |J(T)|$ . Notice that two jobs in  $J(T)$  are indistinguishable as they occur in exactly the same subset of scenarios and have the same (unit) weight.

Given an assignment  $\phi: [n] \rightarrow [m]$  of the machines to the jobs, a *machine configuration* is a vector  $q^{(\phi)} = (q_T^{(\phi)})_{T \subseteq [K]}$  with  $0 \leq q_T^{(\phi)} \leq n_T$ , meaning that,

for every  $T \subseteq [K]$ , exactly  $q_T^{(\phi)}$  jobs of type  $T$  are assigned to a machine. For the sake of simplicity, we shall suppress the assignment in the following and denote  $q^{(\phi)}$  simply by  $q$ .

For scenario  $k \in K$ , the number of jobs scheduled on a machine with configuration  $q$  is  $n(q, k) := \sum_{T \subseteq [K]: k \in T} q_T$ , and the total completion time of machine configuration  $q$  in scenario  $k$  is  $c(q, k) := \frac{1}{2}n(q, k)(n(q, k) + 1)$ ; we define  $c_q := (c(q, k))_{k \in [K]}$ .

The set of all possible machine configurations is denoted by  $Q$ . The number of machine configurations is

$$|Q| = \prod_{T \subseteq [K]} (n_T + 1) \leq (n + 1)^{2^K},$$

and thus polynomial in the input size. A feasible solution, i.e., an assignment of jobs to machines, can be encoded by specifying, for every machine configuration  $q$ , the number of machines  $y_q$  of configuration  $q$ . More precisely, a feasible solution is given by a vector  $y$  satisfying

$$\begin{aligned} \sum_{q \in Q} y_q &= m \\ \sum_{q \in Q} q_T y_q &= n_T \quad \text{for all } T \subseteq [K] \end{aligned}$$

The total completion time of solution  $y$  for scenario  $k$  is  $\sum_q c(q, k) y_q$ .

*Proof (of Theorem 5).* We show how to efficiently solve MinAvgSTC and MinMaxSTC by dynamic programming. In the following we consider tuples  $(m', \pi, d)$  with  $\pi = (\pi_T)_{T \subseteq [K]}$ ,  $d = (d_k)_{k \in [K]}$ , and

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

Notice that the number of such tuples is

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

which is polynomial in the input size. For each tuple  $(m', \pi, d)$  define  $A(m', \pi, d)$  to be *true* if there is an assignment of  $\pi_T$  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

$$A(0, \pi, d) = \begin{cases} \text{true} & \text{if } \pi_T = 0 \text{ for every } T \text{ and } d_k = 0 \text{ for every } k, \\ \text{false} & \text{otherwise.} \end{cases} \quad (21)$$

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

$$A(m', \pi, d) = \bigvee_{q \in Q: q \leq \pi} A(m' - 1, \pi - q, d - c_q); \quad (22)$$

here we set  $A(m' - 1, \pi - q, d - c_q)$  to *false* if  $d - c_q \not\geq 0$ . With (21) and (22), 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.  $\square$

Notice that the dynamic program given in the proof above can be simplified for MinAvgSTC by only storing the total objective function value over all scenarios  $\sum_{k \in [K]} d_k$  in the state space. Since we present a considerably faster algorithmic approach for MinAvgSTC in Section 6, we do not elaborate on this any further.

#### 8.4 Proof of Theorem 7

In the special case of unit weights (and unit processing times), we can prove that there is an optimal solution to every instance such that the full disbalance is bounded by a function only depending on the number  $K$  of scenarios.

To this end, we first prove a weaker statement regarding the 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  $d := \max_k d_k$ . If the final disbalance (under a scenario  $k$ ) refers to a solution  $\varphi$  which is not clear from the context, we denote the respective final disbalance by  $d(\varphi)$  ( $d_k(\varphi)$ ).

**Lemma 1.** *For an instance of MinAvgSTC 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.*

*Proof.* Let  $L_k := \frac{S_k}{m}$  be the average load per machine in the  $k$ -th scenario. Then the machines have load  $L_k + a_{k,1}, \dots, L_k + a_{k,m}$  for suitable rational numbers  $a_{k,1}, \dots, a_{k,m}$  with  $\sum_{i=1}^m a_{k,i} = 0$ . The objective value for this solution then equals

$$\sum_{k=1}^K \sum_{i=1}^n \frac{(L_k + a_{k,i})(L_k + a_{k,i} + 1)}{2} = \sum_{k=1}^K \frac{1}{2} (L_k^2 + L_k + a_{k,i}^2), \quad (23)$$

which is minimized if and only if  $a_{k,i} = 0$  for all  $k \in [K]$  and  $i \in [m]$ . This is the case if and only if  $|J_i \cap S_k| = L_k$  for all  $i \in [m]$ .  $\square$



**Lemma 2.** *For any optimal solution of an instance of MinAvgSTC with unit weights and unit processing times, it holds that*

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

*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  $\varphi: [n] \rightarrow [m]$  be given to an instance with unit weights and processing times. Assume for the sake of contradiction that in the first scenario, the solution restricted to machines 1 and 2 (without loss of generality) admit 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 have the same scenario profile are indistinguishable for our problem. Thus, we obtain disjoint subsets by sorting jobs according to their scenario profile. 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 to the least loaded machine within the subset, leading to an (almost) even load per machine per subset of equivalent jobs.

Formally, we redefine  $J_1$  and  $J_2$  as follows:

$J_1, J_2 \leftarrow \emptyset$   
**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  $S_I := \{j_1^I, \dots, j_{q_I}^I\}$   
      $J_1 \leftarrow J_1 \cup \{j_i^I \in S_I : i \text{ odd}\}$   
      $J_2 \leftarrow J_2 \cup \{j_i^I \in S_I : i \text{ even}\}$

For this solution, we immediately observe that the final disbalances  $d'_k$  under each scenario  $k$  satisfy  $d'_k \leq 2^{K-1}$  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  $\varphi$  being an optimal solution by the step (23) in the proof of Lemma 1.  $\square$

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

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

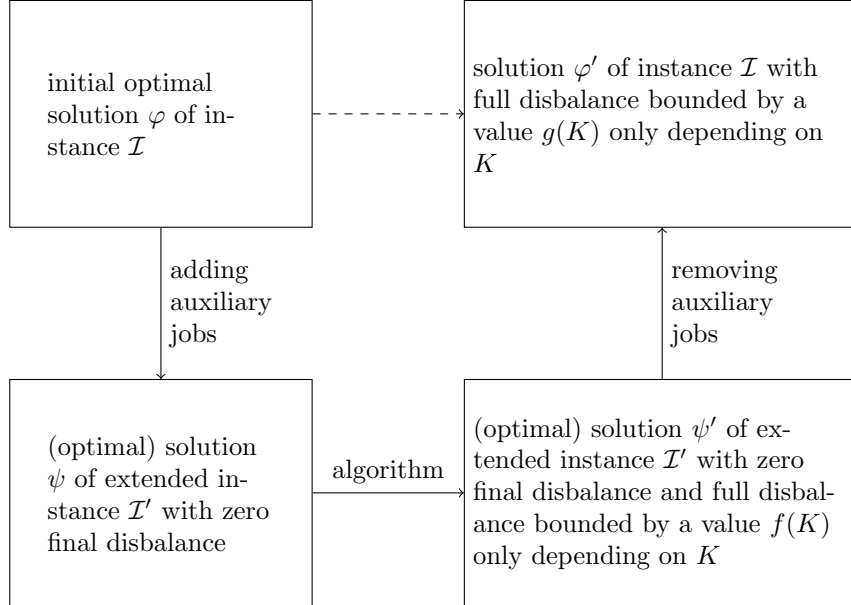
We propose an algorithm that provides a proof of the conjecture for the unit weight case and  $m = 2$ . This algorithm will be used as a subroutine in an algorithm that provides a proof of the unit weight case (with arbitrarily many machines).

**Proposition 3.** *There exists an algorithm that takes an instance of  $\text{MinAvgSTC}$  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*

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

*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  $\varphi: [n] \rightarrow \{1, 2\}$  that has zero final disbalance for each scenario, i.e.  $\sum_{k=1}^K d_k = 0$ , though it is evident that an optimal solution of an arbitrary instance need not satisfy this. However, by Lemma 2, we know that the final disbalance of any optimal solution is at most  $\sqrt{K} \cdot 2^{K-1}$ . To obtain a solution  $\psi: [n + \sum_{k=1}^K d_k] \rightarrow \{1, 2\}$  with zero final disbalance, we add  $d_k$  auxiliary jobs that appear only in scenario  $k$  for  $k = 1, \dots, K$ . An optimal solution of this extended instance now has zero final disbalance at the end of the schedule for every scenario by Lemma 1. We apply our algorithm to this extended optimal solution to obtain a solution  $\psi': [n + \sum_{k=1}^K d_k] \rightarrow \{1, 2\}$  and remove the auxiliary jobs at the end, obtaining an assignment  $\varphi': [n] \rightarrow \{1, 2\}$ . The following diagram demonstrates our reduction onto instances with a solution that satisfies  $\sum_{k=1}^K d_k = 0$ .



It is immediate that 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  $\varphi'$  is at most  $g(K) + \sqrt{K} \cdot 2^{K-1}$  if the full disbalance of the solution  $\varphi$  is  $g(K)$ . One also easily observes that the resulting solution is an optimal one:

*Claim.* The solution  $\varphi'$  as described above is optimal.

*Proof.* As observed in the step (23) the proof of 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.  $\square$

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

For a fixed bijection  $\sigma: 2^{\{1, \dots, K\}} \rightarrow \{1, \dots, 2^K\}$ , 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  $\psi: [n] \rightarrow \{1, 2\}$  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

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

where  $r^1, \dots, r^{q(K)}$  are the extreme rays of  $C$ . We may assume that  $r^p \in \mathbb{Z}^{2^{K+1}}$  and

$$\|r^p\|_\infty \leq \frac{(K!)^2}{2} \quad \text{for } p \in [q(K)]. \quad (24)$$

without loss of generality. 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  $q(K) \leq 2^{2^{K+1}}$ .

An implication of  $C$  being a pointed cone is that the irreducible lattice points of  $C_I$  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  $C_I$ , that is, those that cannot be written as the sum of two nonzero points in  $C_I$ . Combinatorially, these correspond to partial solutions with final disbalance zero that do not have subsolutions with final disbalance zero.

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

*Proof.* It is well-known (cf. [24]) that all elements of  $B$  can be written as a linear combination  $v = \sum_{p=1}^{q(K)} \lambda_p r^p$  for suitable  $\lambda_p \in [0, 1]$  and extreme rays  $r^p$  of  $C$  ( $\ell \in [q(K)]$ ). Hence

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

by (24). □

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

*Proof.* By Claim 8.4, each entry in a basis vector admits integral values between 0 and  $2^{2^{K+1}} \cdot (K!)^2$ . There are hence  $2^{2^{K+1}} \cdot (K!)^2 + 1$  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

$$v = \sum_{b \in B} \lambda_b b \tag{25}$$

with  $\lambda_b \in \mathbb{N}$  and  $\|b\|_\infty \leq 2^{2^{K+1}} \cdot \frac{(K!)^2}{2}$ . This means that we can decompose jobs in our schedule into  $\sum_{b \in B} \lambda_b$  classes each of which has final disbalance zero. These classes correspond to the basis elements  $b \in B$ , there are  $\lambda_b$  copies of such classes for each  $b \in B$ .

The core idea of the algorithm is to exploit the fact that  $|B|$  is bounded by a function of  $K$ . Since  $\lambda_b$  are not necessarily bounded by a function of  $K$ , the most important detail is to bound the sum of full disbalances of classes that contribute to the same  $b \in B$ .

To describe the algorithm, we first consider a fixed  $b \in B$ . Let  $\tau: [n] \rightarrow 2^S, j \mapsto \{s \in S: j \in s\}$  be the transversal to the scenario sets. 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 (25) according to the order  $\sigma$  and assigns each job to the class with the smallest possible index in which  $\tau(j)$  “fits”.

More precisely, let  $B = \{b_1, \dots, b_{|B|}\}$  and let  $\psi: [n] \rightarrow \{1, 2\}$  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  $b_p^1, \dots, b_p^{\lambda_{b_p}}$  denote copies of  $b_p$ . We describe the new assignment  $\psi': [n] \rightarrow \{1, 2\}$  as follows:

```

for  $j = 1$  to  $n$  do
  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
         $b_p^\ell \leftarrow b_p^\ell - e_{\sigma(\tau(j))}$ 
         $\psi(j) \leftarrow 1$ 
        continue
      else if  $b_p^\ell - e_{2\kappa + \sigma(\tau(j))} \geq 0$  then
         $b_p^\ell \leftarrow b_p^\ell - e_{2\kappa + \sigma(\tau(j))}$ 
         $\psi(j) \leftarrow 2$ 
        continue

```

*Claim.* At any point of the algorithm, we have  $b_p^\ell \leq b_p^{\ell'}$  for  $\ell < \ell'$ .

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

Let  $j$  be an arbitrary job that is assigned to a machine at the  $p$ -th iteration of the second inner loop and the  $\ell'$ -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. This is the only case where  $b_p^{\ell'}$  is decreased, and since the values of  $b_p^\ell$  and  $b_p^{\ell'}$  are monotonically decreasing throughout the algorithm, such jobs  $j$  are the only reasons that the claim could become violated for the first time. Since  $\ell < \ell'$ , the job  $j$  has not been assigned at the  $\ell$ -th iteration, meaning that  $(b_p(\ell))_{\sigma(\tau(j))} = (b_p(\ell))_{2\kappa + \sigma(\tau(j))} = 0$  at the  $j$ -th iteration of the outermost loop. On the other hand, we have  $(b_p(\ell))_{\sigma(\tau(j))}, (b_p(\ell))_{2\kappa + \sigma(\tau(j))} \geq 0$  at the end of the  $j$ -th iteration, for else the assignment for  $j$  would have happened later. Since all other entries of  $b_p^\ell$  and  $b_p^{\ell'}$  stay constant during this iteration, the claim follows.  $\square$

The monotonicity of the  $b_p^\ell$  in the parameter  $\ell$  implies that at any point of the algorithm, there is at most one  $\ell$  such that  $(b_p^\ell)_k - (b_p^\ell)_{k+2\kappa} \neq 0$  holds for some  $k \in [K]$ . This particular  $\ell =: \ell(j)$  is the index of the only copy whose corresponding jobs contribute to the current final disbalance. 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 middle iteration  $p$  resp. the two innermost iterations  $(p, \ell(j))$ . The size of the latter is bounded by  $\|b_p\|_1$  by the construction of the algorithm.

Therefore, we have

$$\max_{j \in [n], k \in [K]} \left\{ \max_{i \in \{1,2\}} |\{j' \in J_i \cap S_k : j' \leq j\}| - \min_{i \in \{1,2\}} |\{j' \in J_i \cap S_k : j' \leq j\}| \right\} \quad (26)$$

$$\leq \sum_{p=1}^{|B|} \max_{j \in [n], k \in [K]} \left\{ \max_{i \in \{1,2\}} |\{j' \in J_i \cap S_k \cap B_p : j' \leq j\}| - \min_{i \in \{1,2\}} |\{j' \in J_i \cap S_k \cap B_p : j' \leq j\}| \right\} \quad (27)$$

$$\leq \sum_{p=1}^{|B|} \max_{j \in [n], k \in [K]} \left\{ \max_{i \in \{1,2\}} |\{j' \in J_i \cap S_k \cap B_{p\ell(j)} : j' \leq j\}| - \min_{i \in \{1,2\}} |\{j' \in J_i \cap S_k \cap B_{p\ell(j)} : j' \leq j\}| \right\} \quad (28)$$

$$\leq \sum_{p=1}^{|B|} \|b_p\|_1 \leq \sum_{p=1}^{|B|} 2 \cdot 2^K \cdot \|b_p\|_\infty \leq \left( 2^{2^{K+1}} \cdot (K!)^2 + 1 \right)^{2 \cdot 2^K} \cdot 2 \cdot 2^K \cdot 2^{2^{K+1}} \cdot (K!)^2 \quad (29)$$

The last inequality holds by Claims 8.4 and 8.4. This inequality together with the reduction at the beginning finishes the proof.  $\square$

*Proof (Proof of Theorem 7).* We use the algorithm from Proposition 3 as a subroutine to apply on two machines, after which we obtain full disbalance bounded by a function  $f(K)$  when restricted to these two machines (cf. proof of Proposition 3). We call this subroutine *equalizing* and denote by **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 as follows:

**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**( $i, \operatorname{argmin}_{i' \in [m]} |J_{i'} \cap S_k|$ )  
     **else**  
         **equalize**( $i, \operatorname{argmax}_{i' \in [m]} |J_{i'} \cap S_k|$ )

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

*Proof.* After the subroutine **equalize**( $i_1, i_2$ ) is applied, all summands of  $\mathbf{L}$  indexed by  $i \notin \{i_1, i_2\}$  stay constant. Let  $\mathbf{k}$  be the scenario because of which the subroutine was applied (i.e. one has  $|J_{i_1} \cap S_{\mathbf{k}}| - L_{\mathbf{k}}| > 2K \cdot f(K)$  or  $|J_{i_2} \cap S_{\mathbf{k}}| - L_{\mathbf{k}}| > 2K \cdot f(K)$ ). Then the summands  $|J_{i_1} \cap S_{\mathbf{k}}| - L_{\mathbf{k}}$  resp.  $|J_{i_2} \cap S_{\mathbf{k}}| - L_{\mathbf{k}}$  were each decreased by at least  $\frac{1}{2}|J_{i_1} \cap S_{\mathbf{k}}| - L_{\mathbf{k}} - f(K)$  resp.  $\frac{1}{2}|J_{i_2} \cap S_{\mathbf{k}}| - L_{\mathbf{k}} - f(K)$ . For  $k' \neq \mathbf{k}$ , the increase 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

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

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

### 8.5 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  $\epsilon$  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 DP 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  $J_{m,n}$  be the  $m \times n$  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  $(J_{q, 2q} \ H)$  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  $(J_{q, 2q} \ H)$  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 8.** *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 proof 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  $(J_{q, (t-1)q} \ H)$  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  $(J_{q,(t-1)q} \ H)$  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 leading coefficient equals 1. Hence,  $A_q^t$  also satisfies the properties stated in the theorem.  $\square$

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

## 8.6 Consequences of our Results for Regret Scheduling

In the proof of Theorem 3, 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. we have for any permutation  $\pi: [K] \rightarrow [K]$*

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

*MinMaxSTC restricted to scenario-symmetric instances with unit processing times is NP-hard.*

*Proof.* In the proof of Theorem 3, 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.  $\square$

We can thus relate our model to robust min-max regret scheduling with arbitrary weights and unit processing times [26]. Indeed, the latter model seeks to minimize

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

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  $\tau$  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 4 can also be used to solve min-max regret scheduling by first computing  $\min_{\tau: [n] \rightarrow [m]} G(\tau, k)$  (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 4 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

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

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