Online Makespan Scheduling Under Scenarios

Ekin Ergen
Technical University of Berlin, Germany

Abstract

We consider a natural extension of online makespan scheduling on identical parallel machines by introducing scenarios. A scenario is a subset of jobs, and the task of our problem is to find a global assignment of the jobs to machines so that the maximum makespan under a scenario, i.e., the maximum makespan of any schedule restricted to a scenario, is minimized.

For varying values of the number of scenarios and machines, we explore the competitiveness of online algorithms. We prove tight and near-tight bounds, several of which are achieved through novel constructions. In particular, we leverage the interplay between the unit processing time case of our problem and the hypergraph coloring problem both ways: We use hypergraph coloring techniques to steer an adversarial family of instances proving lower bounds for our problem, which in turn leads to lower bounds for several variants of online hypergraph coloring.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online scheduling, scenario-based model, online algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.27

Related Version *Full Version*: `https://arxiv.org/abs/2507.04016`

Funding The author was 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).

Acknowledgements I would like to thank Martin Skutella for inspiring me to consider this problem, for his invaluable suggestions and for the insightful discussions. I am also grateful to the anonymous reviewers for their careful reading and valuable feedback.

1 Introduction

We study a natural extension of *online makespan scheduling*, also known as *online load balancing*, one of the most well-known problems in the scope of online optimization. In online makespan scheduling, we know a number m of identical parallel machines in advance, in addition to which jobs j are revealed one at a time along with their processing times $p_j \geq 0$. Our task is to assign the jobs to a machine as soon as they are revealed, solely under the knowledge of the preceding jobs, so as to minimize the *makespan*, i.e., the maximum sum of processing times any machine is assigned to. This problem was initially studied by Graham almost 60 years ago, who proved that assigning every job to a currently least loaded machine is a $(2 - \frac{1}{m})$-competitive algorithm [13]. It was observed that no better deterministic algorithm can exist for $m \in \{2, 3\}$ [13, 9], while for larger values of m, such a tight ratio is still not known.

We extend online makespan scheduling by a *scenario-based model*: We are given a number m of machines. There is a known number K of job sets (called *scenarios*) $S_k \subseteq \{1, \dots, n\}$, $k \in [K]$, over a common ground set $\{1, \dots, n\}$. Although the number of scenarios is known, neither the number of jobs nor their processing times p_j ($j \in [n]$) are known in advance. The ground set $[n]$ is revealed in increasing order of jobs; more precisely, we are incrementally informed of $p_1, \dots, p_j$ as well as $S_k \cap \{1, \dots, j\}$ for increasing j. The task is to find a global assignment $\tau \colon \{1, \dots, n\} \to \{1, \dots, m\}$ of the jobs to machines that delivers a reasonable solution restricted to every scenario. To be more precise, the objective is to minimize the makespan of the *worst-case scenario*, i.e., the scenario that yields the highest makespan. As in the classical online scheduling problem, a job j must be assigned irrevocably to a machine $\tau(j)$ as soon as it is revealed. In essence, we are solving a finite number of instances of Online Makespan Scheduling over the same ground set of jobs simultaneously.

The simplicity of the scenario model offers a variety of interpretations and applications. For instance, we can view the scenarios as distinct services that are provided in m (number of machines) sites to n (number of jobs) customers, where the customers are revealed one by one with their desired services and need to be assigned to one of the sites immediately. Another interpretation of our problem may concern a fair balancing of goods according to multiple agents in the following setting: Each of the K agents are interested in partitioning a subset S_k, $k \in \{1, \dots, K\}$ of n goods with weights p_j among m buckets, where the partition should be as balanced for all agents as possible (i.e., we would like to minimize the maximum bucket load for any agent). Here, the goods are revealed one by one together with the agents that are interested in them.

Moreover, even special cases of online makespan scheduling under scenarios correspond to interesting questions in their own regard. The special case $p_j \equiv 1$ of unit processing times is related to some cases of discrepancy minimization as well as hypergraph coloring.

In the large array of previous research on combinatorial optimization under scenarios, it was shown for several offline problems that optimizing under a number of scenarios is computationally harder than their underlying single-scenario counterparts. Accordingly, our goal in this work is to gain a better understanding of the intersection of scenario-based models and competitiveness, in particular how competitiveness diminishes under a growing number of scenarios.

1.1 Our Contribution

Implementing a miscellany of ideas to analyze competitiveness bounds for the varying numbers m of machines and K of scenarios, we achieve several tight bounds and further nontrivial bounds. An overview of our results, depending on the numbers m and K, can be seen in Table 1. Several of the entries are obtained via a generalization of Graham's List Scheduling Algorithm which we extend using the pigeonhole principle.

By a simple example, one can show that for any number $m \geq 2$ of machines and (at least) 3 scenarios, no algorithm with a competitive ratio better than 2 can exist, even for jobs with unit processing times. To underline the competitiveness gap between two and three scenarios, we contrast the aforementioned result with an algorithm for two machines and two scenarios, which is one of our main results.

▶ **Theorem 1.** *There exists a* $5/3$*-competitive algorithm for* Online Makespan Scheduling under Scenarios *for* $m = K = 2$.

This algorithm as well as its analysis combine techniques well-known from algorithms for online makespan scheduling with novel ideas that concern the scenario-based properties of a schedule. We first alter the definition of competitive ratio by allowing only specific lower bounds for an offlie optimum, which allows us to make structural modifications and restrictions on *worst-case* instances. What sets our analysis apart from previous work is that we alternate between forward and reverse engineering: Assuming that our algorithm obeys some fixed rules, we explore what we may assume about a well-structured worst-case instance. In spite of seemingly strong restrictions that we impose, the ratio of our algorithm is not far from the best achievable bound by any deterministic algorithm, which we show is at least $\frac{9+\sqrt{17}}{8} \approx 1.640$. Altogether, we obtain a near-complete picture of competitiveness for $m = 2$.

The remainder of the paper demonstrates a disparity between increasing the number m of machines and increasing the number K of scenarios. For a fixed but arbitrary m, we show that there exists a sufficiently large number $K(m)$ of scenarios and a family of instances with $K(m)$ scenarios, on which no deterministic online algorithm can achieve a ratio better than m. Furthermore, we can even choose all instances to have unit processing times $p_j \equiv 1$. This result is particularly noteworthy, considering that any online algorithm on m machines is trivially m-competitive: Indeed, the online output is at most the total load and an offline optimum is at least the average load, and these are by at most a factor of m apart.

To find the suitable number $K(m)$, we interpret the instances as hypergraphs, passing to a problem we call Online Makespan Hypergraph Coloring(m) instead (cf. Section 1.2). Strikingly, such a family can be created using hypertrees on which no deterministic algorithm can achieve a coloring without a monochromatic edge simultaneously.

▶ **Theorem 2.** *Let $m \in \mathbb{N}$. There exists a number $K(m)$ such that for any $r < m$, no deterministic algorithm for Online Makespan Hypergraph Coloring(m) is r-competitive, even when restricted to hypertrees with $K(m)$ hyperedges.*

The statement of Theorem 2 is best possible in the sense that the numbers $K(m)$ cannot be replaced by a global K for which the incompetitiveness holds for all m (cf. Theorem 5).

The proof of Theorem 2 can be found in the full version of the paper. To showcase some of our techniques, we sketch a proof for a similar statement for $K = 3$ in Section 4. We utilize a subhypergraph on 7 nodes as a gadget and obtain a relatively compact construction, both in terms of the hypergraph and the case distinction that we apply to analyze it.

An advantage of our approach while proving Theorem 2 is that it only features hyperforests. More precisely, we implicitly show that for every $m \in \mathbb{N}$, there is a number $n \le (2m+1) \uparrow\uparrow 5$ of nodes[1] such that no deterministic online algorithm on n nodes that uses m colors can avoid a monochromatic hyperedge. In other words, in order to color a hyperforest with an arbitrary number n of nodes online without monochromatic edges, we need $\Omega(\mathrm{slog}_5 n)$ colors, even if the restrictions $e \cap \{1, \dots, j\}$ of hyperedges e to revealed nodes are known upon the revelation of nodes $\{1, \dots, j\}$. Here, the emphasis is on the additional knowledge of partial hyperedges: This makes an adversarial revelation of hyperedges more challenging and to the best of our knowledge, none of the previous lower bound constructions [15, 14] are applicable under such additional information. This result addresses a completely open problem from [15] and sets a non-constant lower bound for Online Hypergraph Coloring for hypertrees under the knowledge of partial hyperedges.

▶ **Corollary 3.** *Let $n \in \mathbb{N}$. There exists no $o(\mathrm{slog}_5 n)$-competitive deterministic algorithm for Online Hypergraph Coloring with nodes $v_1, \dots, v_n$ (revealed in this order) even if we restrict to hypertrees and additionally, the subsets $e \cap \{v_1, \dots, v_j\}$ of hyperedges $e \in E$ are known upon the assignment of node v_j.*

For unit processing times, $K = 3$ and varying m, we further propose a 2-competitive algorithm, which is best possible due to the lower bounds mentioned earlier. Although the special case of unit processing times is admittedly a restriction, the result still illustrates a contrast with the tight lower bound of m for a varying number of scenarios, which also holds already for instances with unit processing times.

[1] $F(x) := x \uparrow\uparrow 5 := x^{x^{x^{x^{x}}}}$ and $\mathrm{slog}_5(x) := F^{-1}(x)$.

Table 1 Overview of the competitive ratios obtained by our results. Recall that on m machines, any algorithm is trivially m-competitive.

m $\downarrow$ K $\rightarrow$	2	3	4+
2	> 1.640, $\leq 5/3$	$= 2$	$= 2$
3	≤ 2	≥ 2, $= 2$ for $p_j \equiv 1$	≥ 2 $= 3$ for $K \geq 233$ and $p_j \equiv 1$
4+	≥ 3	$\geq 2, < 4$ $= 2$ for $p_j \equiv 1$	$\geq 2, < K + 1$ $= m$ for K sufficiently large and $p_j \equiv 1$

1.2 Preliminaries

Throughout this paper, for $n \in \mathbb{N}$, we denote by $[n] := \{1, \ldots, n\}$ the set of numbers 1 through n. An instance $I = (n, (p_j)_{j \in [n]}, \{S_k\}_{k \in [K]})$ of Online Makespan Scheduling under Scenarios(m, K) (OMSS(m, K)) is parametrized by a number $m \in \mathbb{N}$ of machines and a number $K \in \mathbb{N}$ of scenarios, and defined as a tuple consisting of a number $n \in \mathbb{N}$ of jobs, a processing time $p_j \geq 0$ for each job j and job subsets $S_1, \ldots, S_K \subseteq [n]$, called *scenarios*. As we usually work with a single instance at a time, we often suppress this notation in the sequel as well as the parameters m and K, which are often specified per section.

For a set $S \subseteq [n]$, we denote $p(S) := \sum_{j \in S} p_j$. Our objective is to find a partitioning $J_1 \dot\cup \ldots \dot\cup J_m$ of the set of jobs that minimizes $\max_{k \in [K]} \max_{i \in [m]} p(J_i \cap S_k)$. In other words, we consider schedules that arise from restricting a partition to each of the scenarios, and our goal is then to minimize the makespan of the worst-case scenario, which we also call the *makespan of the schedule*. Furthermore, our instance is *online* in the sense that parameters m (number of machines) and K (the number of scenarios) are known in advance, but jobs $j \in [n]$ as well as their processing times p_j and the indices $k \in [K]$ such that $j \in S_k$ are revealed one job j at a time. Once the information of a job j is revealed in addition to already known information about the jobs in $[j-1]$, the job j must be assigned to a machine $i \in [m]$, which is an irrevocable decision.

In our analyses and arguments, we use the notion of a *completion time* of a job j. In classical scheduling, this is defined as the sum of processing times preceding (including) j that are on the same machine as j. In our setting, the restricted schedules are evaluated without idle times, meaning that such completion times might be different for different scenarios. Accordingly, we define the *completion time of job* $j \in J_i \cap S_k$ *in scenario* S_k as $C_j^k := \sum_{j' \in J_i \cap S_k \cap [j]} p_{j'}$ and the *completion time* as $C_j := \max_{k \in [K], j \in S_k} C_j^k$.

Online algorithms are often analyzed by their performance compared to the best possible outcome under full information. In existing literature, this comparison is quantified by the *competitive ratio*. An algorithm is said to have competitive ratio ρ if for every instance, it is guaranteed to output a solution that has an objective value which deviates from the value of an optimal solution under full information by at most a factor of ρ.

We also consider the special case of unit processing times $p_j \equiv 1$. In this special case, we may view our jobs as nodes, our scenarios as hyperedges containing a subset of the nodes and the partition $J_1 \dot\cup \ldots \dot\cup J_m$ as a coloring $c \colon [n] \to [m]$ of the nodes. Then, our problem can be formulated as an online hypergraph coloring problem, in which we color a hypergraph $H = (V, E)$ with a fixed number of colors $1, \ldots, m$ and seek to minimize $\max_{i \in [m], e \in E} |\{c^{-1}(i)\} \cap e|$. The nodes $j \in [n]$ are revealed one at a time in increasing order along with partial hyperedges $e \cap [j]$ $(e \in E, j \in [n])$.[2] We shall name this problem Online Makespan Hypergraph Coloring(m) (OMHC(m)) and address it in Section 4.

[2] According to this definition, hyperedges can have size 1, although such hyperedges can and will be omitted as they pose no difference with respect to the objective function.

1.3 Related Work

The problem that we introduce in this paper lies in the intersection of several well-known problems, which we would like to mention briefly.

Optimization under Scenarios. In recent years, scenario-based models have received significant attention. An instance of a scenario model over a discrete optimization problem is often given as an instance of the underlying problem; and in addition, subsets of the underlying instance (called *scenarios*) are specified. The objective function involves the objective values that are attained restricted to each scenario; e.g., the maximum of these values over scenarios or their average.

Although a significant portion of the literature is rather recent, some well-known problems have been scenario-based problems in disguise. For instance, the famous exact matching problem ([16]), which asks whether a given graph whose edges are colored blue or red has a perfect matching with exactly k blue edges for a given k, is equivalent to the matching problem under two scenarios, whereas a stochastic scenario-based model for matchings has recently been studied in [4]. Other examples include bin packing ([7, 5]), metric spanning tree ([3]) and the traveling salesperson problem ([24]).

Among the scenario-based problems with the widest array of studies is machine scheduling with its many variations, see [21] for an extensive overview. The closest to our problem is [10], in which the worst-case makespan is being minimized for parallel machine scheduling on two machines and a variable number of scenarios. Although an instance under a variable number of scenarios is not approximable with a ratio better than the trivial bound of 2, a constant number of scenarios allows a PTAS.

In some variants of scenario scheduling, e.g., in [6], a complexity gap is observed between two and three scenarios. Interestingly, we obtain a distinction between two and three scenarios as well, at least when we restrict to the special case of two machines.

Online Scheduling. As it is impossible to capture the vast amount of literature on all online scheduling problems, let us restrict to online parallel machine scheduling with the objective of minimizing the makespan. This is precisely the special case $K = 1$ of our problem.

The problem was first addressed in Graham's seminal work [13] from which we also draw inspiration in multiple ways. The competitive ratio of $2 - 1/m$ for m machines was not beaten for three decades, until it was improved slightly in [12]. Graham's List Scheduling Algorithm is evidently best possible for $m = 2$, and it was observed to be best possible for $m = 3$ as well [9]. In the following years, a stream of research narrowed the gap between upper and lower bounds for competitive ratios depending on m ([2, 17, 1]), although for $m \geq 4$, still no tight bound is known. The state-of-the-art results are Fleischer and Wahl's 1.9201-competitive algorithm ([11]) and an asymptotic lower bound of 1.88 due to Rudin III ([18]). For $m = 4$, the best known lower bound is $\sqrt{3} \approx 1.732$, due to Rudin III and Chandrasekaran ([19]).

Discrepancy Minimization, Hypergraph Coloring. A problem similar to Online Makespan Hypergraph Coloring, often called Online Hypergraph Coloring, has been studied extensively over the years. In this problem, the objective is to use the smallest number of colors while avoiding a *monochromatic hyperedge*, i.e., a hyperedge whose nodes all attain the same color. For hypertrees with n nodes, an $O(\log(n))$-competitive algorithm is known to be best possible ([14]), while for general graphs with n nodes, no $o(n)$-competitive algorithm can exist ([15]). However, this standard model differs from ours, as a hyperedge is revealed only once all of its nodes have been revealed. This nuance deems it much easier to construct families of instances to achieve lower bounds. Moreover, due to differences in their respective objective functions, this problem does not seem to have a direct relation to Online Makespan Hypergraph Coloring. However, our approach has implications for a variant of Online Hypergraph Coloring in which partial hyperedges are known (see Section 1.1).

The reader may also notice similarities to the well-known discrepancy minimization problem (see e.g. [23, 8]): Indeed, in the usual setting of discrepancy minimization, we have $m = 2$ and the objective is to minimize the difference $\max_{e\in E} ||\{c^{-1}(1)\} \cap e| - |\{c^{-1}(2)\} \cap e||$. If all hyperedges $e \in E$ have the same size, this objective function is even equivalent to ours. In this regard, the problem that we study generalizes the ℓ_∞ norm online discrepancy minimization [22] in uniform set systems as well.

2 Extending Graham's List Scheduling

Our first observation is a simple algorithm that generalizes Graham's List Scheduling Algorithm for OMSS(m, K) and performs well when there are too many machines to force the schedule to create significant discrepancy. Our algorithm beats the trivial bound of m if $m > K + 1$.

We first observe that if the number m of machines is large enough, we can apply the pigeonhole principle to find a machine that is not among the most loaded machines of any scenario. For $s \in [m-1]$ and $k \in [K]$, we call a machine i *s-favorable with respect to the k-th scenario* if there are at least s distinct machines $i' \neq i$ that each have at least the load of machine i in the k-th scenario, i.e., $p(J_{i'} \cap S_k) \geq p(J_i \cap S_k)$.

▶ **Observation 4.** *There exists a machine i that is $\left(\left\lceil \frac{m}{K} \right\rceil - 1\right)$-favorable with respect to every scenario $k \in [K]$.*

Proof. There are at most s machines per scenario that are not s-favorable. Let $s = \left\lceil \frac{m}{K} \right\rceil - 1$. Since $m > s \cdot K$, there must be a machine i which is not non-s-favorable with respect to any scenario. ◀

In light of this observation, we propose the following algorithm: Upon the assignment of each job j, we find a machine i that is $(\lceil m/K \rceil - 1)$-favorable with respect to all scenarios k with $j \in S_k$ and assign the job j to the machine i.

▶ **Theorem 5.** *The above algorithm is $\left(\frac{m-1}{\left\lceil \frac{m}{K} \right\rceil} + 1\right)$-competitive for $m > K$.*

Notice that the aforementioned competitive ratio is strictly smaller than $K + 1$.

This algorithm is also applicable in the special case where every job appears in at most k scenarios for a fixed $k \in \mathbb{N}$, yielding a ratio of $(m-1)/\left\lceil \frac{m}{k} \right\rceil + 1$. Furthermore, plugging in $K = 1$ yields the well-known competitive ratio of $2 - 1/m$ of Graham's List Scheduling Algorithm.

3 Near-Tight Bounds for $m = 2$ Machines

When the partitioning is among two machines, a clear line can be drawn: Any algorithm is trivially 2-competitive, and if we have at least 3 scenarios, this is best possible.

▶ **Observation 6.** *For $m = 2$ and $K \geq 3$, there exists no r-competitive algorithm for any $r < 2$, even for the special case of unit processing times.*

Proof. For all jobs j below, we assume $p_j = 1$. The first instance $\mathcal{I}_1$ is given as follows: $n = m + 1$, $S_1 = \{1, 3, \ldots, m + 1\}$, $S_2 = \{2, 3, \ldots, m + 1\}$.

We consider an arbitrary algorithm that places the first jobs 1 and 2 to two distinct machines, without loss of generality, to the first and second, respectively. In this case, the makespan at the end must be 2, as two jobs $(j, j') \neq (1, 2)$ must be assigned to the same machine. However, the following assignment, which places the first two jobs together would have yielded a completion time of 1: $J_1 = \{1, 2\}, J_i = \{i + 1\}$ for $2 \leq i \leq m$. Hence, no algorithm that places two unit weight jobs in $S_1 \setminus S_2$ resp. $S_2 \setminus S_1$ to two separate machines can attain a competitive ratio better than 2.

In light of this, we consider another instance $\mathcal{I}_2$, on which assigning the first two jobs to the same machine leads to a failure. It is given by $n = m + 2$, $S_1 = \{1, 3\}$, $S_2 = \{2, 4, 5, 6, \ldots, m + 2\}$, $S_3 = \{3, 4, 5, \ldots, m + 2\}$. The first two jobs are indeed identical to those of $\mathcal{I}_1$.

By our considerations about the instance $\mathcal{I}_1$, we may assume that the first two jobs are assigned to the same machine, the first one without loss of generality. However, any assignment that places the first two jobs to the same machine admits makespan 2: Otherwise, due to the third scenario, the jobs $\{3, \ldots, m + 2\}$ must occupy one machine each. The job j that is placed to the first machine has a common scenario with either job 1 (if $j = 3$), or with job 2 (otherwise). ◀

On the other hand, this statement cannot be extended to $K = 2$. To show this, we present a minimalistic yet insightful algorithm (Algorithm 1) for makespan scheduling under $K = 2$ scenarios and $m = 2$ machines. Our general idea is to fix a simple rule for assigning *double-scenario jobs*, i.e., jobs in $S_1 \cap S_2$, and partition the *single-scenario jobs* $j \in S_1 \triangle S_2$ so that the machines are reasonably balanced even after upcoming hypothetical double-scenario jobs.

▶ **Rule 7.** *If $j \in S_1 \cap S_2$, then assign the job j to a machine i such that the makespan of the schedule was previously attained by the other machine $3 - i$. In case of a tie, select the machine that received the job $j - 1$ ($j = 1$ is assigned to the first machine if applicable).*

Throughout our analysis, we only use two types of lower bounds for the offline optimum: Processing times p_j of selected jobs j whose completion times equal the makespan, and the average load of a selected scenario S_k. Hence, while searching for the worst-case sequence of the upcoming double-scenario jobs, we may replace the actual definition of competitiveness with that of a proxy one:

▶ **Definition 8.** *We define the* proxy competitive ratio *of a schedule as*

$$\rho = \frac{\max_{i,k \in \{1,2\}} p(S_k \cap J_i)}{\max\left\{\max_{k \in \{1,2\}} \left\{\frac{p(S_k)}{2}\right\}, \max_{j \in [n] \cap \operatorname{argmax}_{j \in [n]}\{C_j\}} \{p_j\}\right\}},$$

where C_j denotes the completion time of the job j.

The proxy competitive ratio is an upper bound on the competitive ratio. The lower bounds that we use are already well-known; e.g., Graham's List Scheduling Algorithm is analyzed by bounding the proxy competitive ratio from above. We take this approach, however, one step further and restrict ourselves to analyzing only some instances that attain the largest possible proxy competitive ratio (see next Observation). This is significantly more difficult for the competitive ratio in the usual sense, as a witness to having good or bad offline optima highly depends on how the processing times of jobs fit together numerically.

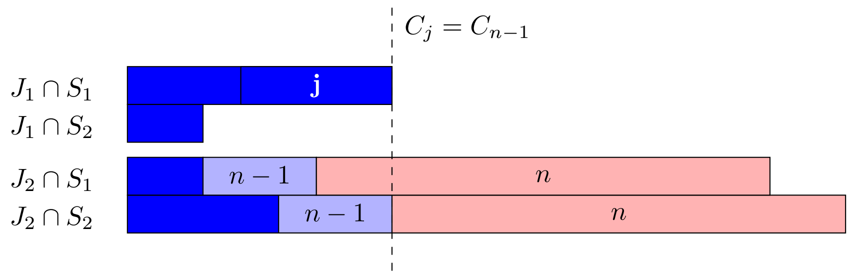

Figure 1 Illustrative description of Observation 9. Processing times p_j are given by the widths. Given the dark blue schedule, a worst outcome turns out to be the light blue job $n-1$ with a processing time p_{n-1} such that $C_{n-1} = C_j$, followed by the light red job n with $p_n = p(S_1 \cap [n-1])$.

▶ **Observation 9** (cf. Figure 1). *Let $\mathcal{A}$ be an online algorithm for OMSS$(2,2)$ obeying Rule 7. Assume that $\mathcal{A}$ has a proxy competitive ratio of at most $\frac{5}{3}$ on instances where*
(i) *$n \in S_1 \cap S_2$, and*
(ii) *$\mathcal{A}$ outputs a schedule with the following property: If $j \in [n]$ is the largest index with $j \in S_1 \triangle S_2$, then $C_j = C_{n-1}$.*

Then, $\mathcal{A}$ is $\frac{5}{3}$-competitive.

The above observation motivates the following parameter of significant value.

▶ **Definition 10.** *Consider a schedule where, without loss of generality, the makespan is attained at $S_1 \cap J_1$. The* anticipation *α of this schedule is given by the ratio*

$$\alpha := \begin{cases} \frac{p(J_1 \cap S_1)}{\max\{p(J_1 \cap S_2), p(J_2 \cap S_1) + p(J_1 \cap S_1) - p(J_2 \cap S_2)\}} & \text{if } p(J_2 \cap S_2) > p(J_2 \cap S_1), \\ 0 & \text{else.} \end{cases}$$

By convention, we assume $\frac{0}{0} = 1$.

In Figure 1, we have $p(J_2 \cap S_2 \cap [n-2]) > p(J_2 \cap S_1 \cap [n-2])$ and the second term of the denominator corresponds to the load $p(J_2 \cap S_1 \cap [n-1])$, i.e., the load of $J_2 \cap S_1$ after the $(n-1)$-st job is assigned. In general, anticipation measures the minimum discrepancy of a machine with respect to different scenarios after machine 2 is loaded with double-scenario jobs just enough to attain the makespan of the schedule together with machine 1. In our algorithm, one of the goals is to keep $\alpha \leq 2$ in order to brace ourselves for upcoming double-scenario jobs. The invariant which we maintain to this end is the following.

▶ **Invariant 11.** *The schedule satisfies the following assertions:*
(i) *Its proxy competitive ratio is bounded by $\rho \leq 5/3$.*
(ii) *The schedule has anticipation bounded by $\alpha \leq 2$.*
(iii) *Let k, i be given so that $J_i \cap S_k$ admits the makespan in the schedule. If the schedule is* dominated by the i-th machine*, i.e., $p(J_i \cap S_{k'}) > p(J_{3-i} \cap S_{k''})$ for all $k', k'' \in \{1,2\}$, then we have $p(J_i \cap S_k) \leq 2p(J_i \cap S_{3-k})$.*

The last property might appear somewhat unnatural at the first glance, though it is crucial for bounding the anticipation $\alpha \leq 2$ in the subsequent iterations.

Algorithm 1 shows how a single job j is assigned given a schedule of the first $j-1$ jobs where Invariant 11 is maintained. To analyze the algorithm, we prove that instances satisfying the properties in Observation 9 are always assigned so that the first $n-1$ jobs maintain Invariant 11 and the last job results in a proxy competitive ratio of at most $5/3$.

Algorithm 1 The assignment of a single job j given a $\frac{5}{3}$-competitive schedule that satisfies Invariant 11.

```
1: Rename machines and scenarios so that $(J_1 \cap S_1)$ is maximal
2: if $j \in S_1 \setminus S_2$ then
3:     return 2
4: end if
5: if $j \in S_2 \setminus S_1$ then
6:     if assigning $j$ to the first machine satisfies Invariant 11 then
7:         return 1
8:     else
9:         return 2
10:    end if
11: end if
12: if $j \in S_1 \cap S_2$ then
13:     return $\text{argmin}_{i=1,2} \max_{k=1,2}\{p(J_i \cap S_k)\}$
14: end if
```

A significant portion of the analysis is focused on proving that if line 9 is executed, Invariant 11 is indeed maintained. We refer the reader to the full version of the paper for further details.

In spite of the restrictions that arise from both the fixed double-scenario rule and how we evaluate lower bounds, we find out that our algorithm in the previous subsection is not far from being best possible.

▶ **Theorem 12.** *For $m = 2$ and $K = 2$, there exists no deterministic algorithm with competitive ratio strictly smaller than $\frac{9+\sqrt{17}}{8} \approx 1.640$.*

Proof. To ease the notation once again, let $X := 3 + \sqrt{17}$.

The first four jobs are given by $1, 2 \in S_1 \setminus S_2$, $3, 4 \in S_2 \setminus S_1$, $p_1 = p_2 = p_3 = p_4$. Any algorithm with competitive ratio better than 2 must assign these jobs so that the makespan equals 1 at the end; otherwise, we may stop revealing further jobs so that the offline optimum equals 1, while the makespan equals 2.

The fifth job fulfills $5 \in S_1 \setminus S_2$ and $p_5 = X$. Without loss of generality, it is assigned to the first machine.

The sixth job is given by $6 \in S_2 \setminus S_1$, $p_6 = \frac{X}{2}$. Now there are two options:

(i) If the sixth job is also assigned to the first machine, we reveal the seventh job with $7 \in S_2 \setminus S_1$ and $p_7 = X$. This job cannot be assigned to the first machine, as the makespan would then be $1 + \frac{3X}{2}$, while the offline optimum is X. Indeed, this would imply a ratio of at least

$$\frac{1 + \frac{9+3\sqrt{17}}{2}}{3 + \sqrt{17}} = \frac{9 + \sqrt{17}}{8}.$$

In particular, we obtain $p(S_1 \cap J_1 \cap [7]) = p(S_2 \cap J_2 \cap [7]) = 1 + X$.

(ii) If the sixth job is assigned to the second machine, we reveal the seventh job as $7 \in S_1 \cap S_2$ and $p_7 = \frac{X}{2}$. Due to similar reasoning to the other case, the seventh job must be assigned to the second machine and we again have $p(S_1 \cap J_1 \cap [7]) = p(S_2 \cap J_2 \cap [7]) = 1 + X$.

See Figure 2 for the first seven jobs in each of the cases.

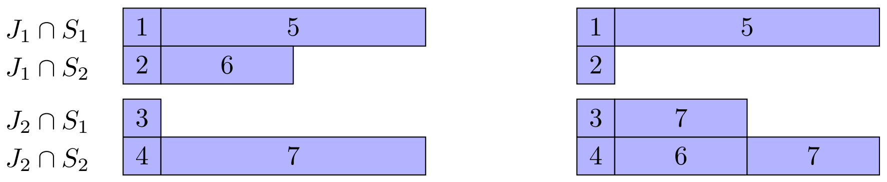

Figure 2 The first seven jobs in cases (i) and (ii) of the proof of Theorem 12, respectively. In both cases, an eighth job $8 \in S_1 \cap S_2$ forces the schedules to the desired lower bound of competitiveness.

In both of these cases, we reveal the eighth job with $8 \in S_1 \cap S_2$ and $p_8 = 2 + \frac{3X}{2}$. No matter where this job is assigned, the makespan is $3 + \frac{5X}{2}$, whereas the optimal solution $\{1, \ldots, 7\} \dot\cup \{8\}$ achieves a makespan of $2 + \frac{3X}{2}$. Therefore, the competitive ratio is bound to be at least

$$\frac{3 + \frac{5X}{2}}{2 + \frac{3X}{2}} = \frac{6 + 5(3 + \sqrt{17})}{4 + 3(3 + \sqrt{17})} = \frac{9 + \sqrt{17}}{8},$$

as desired. ◀

Notice that the aforementioned family of instances fails to achieve small competitive ratios for reasons similar to those that require Property (iii) in Invariant 11.

It can also be shown through a simple array of instances that, no algorithm that satisfies Rule 7 can beat the bound of 5/3, for which we refer the reader to the full version of the paper.

4 Tight Bounds for Sufficiently Many Scenarios

One of our main results (Theorem 2) states that for any number $m \in \mathbb{N}$ of machines, there exists a number K of scenarios such that no deterministic algorithm for OMSS(m, K) can have a competitive ratio less than the trivial upper bound m, even for unit processing times.

Due to space constraints, we merely sketch a proof for the special case of $m = 3$. While doing so, we showcase key ideas of the general argument, although we forfeit the restriction onto hypertrees for the sake of a more clearly structured description. In the context of scheduling, this is indeed not a significant change.

▶ **Theorem 13.** *For OMHC*(3)*, there exists no* r*-competitive online algorithm for any* $r < 3$*, even when restricted to instances with at most* 233 *hyperedges of size at most* 3 *each.*

The problem OMHC(m), which we formulated in Section 1.2, is equivalent to the unit processing time case of OMSS(m, K) with K part of the input, so that the lower bounds are immediately implied for the latter as well.

To prove Theorem 13, we provide a family of instances on which any online algorithm is forced to output a monochromatic hyperedge of size 3. For each of the instances, we also maintain (offline) colorings $c\colon [j] \to \{\text{blue}, \text{red}, \text{yellow}\}$, with the property that if two nodes v, v' are contained in a common hyperedge, we have $c(v) \neq c(v')$. The latter coloring suggests an offline optimal value of 1 for each of the instances, implying the competitive ratio of 3 immediately. The main difference between the online and offline colorings is, as expected, that we may change the offline coloring completely at all times as long as its makespan equals 1. Among other modifications, we often employ a permutation $\pi\colon \{\text{red}, \text{blue}, \text{yellow}\} \to \{\text{red}, \text{blue}, \text{yellow}\}$ on connected components to realize this possibility. We must trace the offline coloring simultaneously because the hyperedges are often defined adversarially according to both colorings of the current hypergraph.

To differentiate between the online and offline colorings, we denote by $J_i \subseteq V$ the subset of nodes assigned to the online color i for $i \in \{1, 2, 3\}$. Right before the revelation of a node j, we implicitly mean $J_i \cap [j - 1]$ whenever we mention J_i for some $i \in \{1, 2, 3\}$. Inspired by our original scheduling problem, we call the objective value $\max_{e \in E} \max_{i \in \{1,2,3\}} |e \cap J_i|$ of an instance its *makespan*. Without loss of generality, we truncate an instance as soon as the online solution is forced to obtain a makespan of 3.

Our gadget: Seven-node subhypergraph $\mathcal{S}$. Initially, we study a subhypergraph $\mathcal{S}$ that we use as a building block throughout our construction. The hypergraph $\mathcal{S}$ is given by the nodes $V(\mathcal{S}) = \{1, \ldots, 7\}$ and hyperedges

$$E(\mathcal{S}) = \big\{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}, \{2, 4\}, \{4, 5\}, \{2, 5\}, \{2, 4, 5\}, \\ \{4, 6\}, \{6, 7\}, \{4, 7\}, \{4, 6, 7\}, \{3, 6\}, \{3, 7\}\big\}.$$

The hyperedges have size at most 3 and the set $E(\mathcal{S})$ is closed under taking subsets of size at least 2, i.e., if $e \in E(\mathcal{S})$, $|e| = 3$ and $e' \subseteq e$ with $|e'| = 2$, then $e' \in E(\mathcal{S})$. The hypergraph $\mathcal{S}$ together with the maximal hyperedges $e_1, \ldots, e_5$ is given in Figure 3.

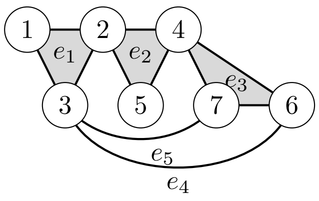

Figure 3 The hypergraph $\mathcal{S}$ with the maximal hyperedges $e_1, \ldots, e_5$.

We first assume that the seven nodes in $V(\mathcal{S})$ are assigned to exactly two out of the three online colors. Since the online colors are initially symmetric, let us assume that we use the online colors 1 and 2. By a careful case distinction, we may observe:

▶ **Proposition 14.** *For any partitioning $V(\mathcal{S}) = J_1 \dot\cup J_2$ of the nodes that admits a makespan of at most 2, there exists a node coloring $c \colon V(\mathcal{S}) \to \{\text{blue}, \text{red}, \text{yellow}\}$ such that there are two hyperedges $e \in E(\mathcal{S}[J_1]), e' \in E(\mathcal{S}[J_2])$ with $c(v) \in \{\text{red}, \text{blue}\}$ (or another subset of $\{\text{blue}, \text{red}, \text{yellow}\}$ of size 2) for all $v \in e \cup e'$.*

In other words, we have induced monochromatic hyperedges of size 2 with two different colors, both with respect to the online coloring. Applying this repeatedly, we can further force the algorithm to a monochromatic hyperedge of size 3, which we show next.

Pigeonholing the two-color case. In our construction, we would like to create *copies* of the graph $\mathcal{S}$ successively, which we denote by $\mathcal{S}(t)$ for incrementing $t \in \mathbb{N}$.

A straightforward application of the pigeonhole principle yields that, if we have seven subhypergraphs $\mathcal{S}(1), \ldots, \mathcal{S}(7)$ that are each partitioned onto two online colors, at least three of them (without loss of generality $\mathcal{S}(1)$, $\mathcal{S}(2)$, $\mathcal{S}(3)$) must be partitioned onto the same two online colors. Again without loss of generality, these two online colors are 1 and 2. Considering Proposition 14, we can permute the colorings of the three copies $\mathcal{S}(1)$, $\mathcal{S}(2)$ and $\mathcal{S}(3)$ each offline so that for any offline color $u \in \{\text{blue}, \text{red}, \text{yellow}\}$, there exists a copy $\mathcal{S}(t)$ with two (online) monochromatic hyperedges $e_i^t \in E(\mathcal{S}(t))$ $(i \in \{1, 2\})$ of size 2 avoiding the color u. The hypergraph in Figure 4 restricted to nodes of the form j_s^i illustrate these edges.

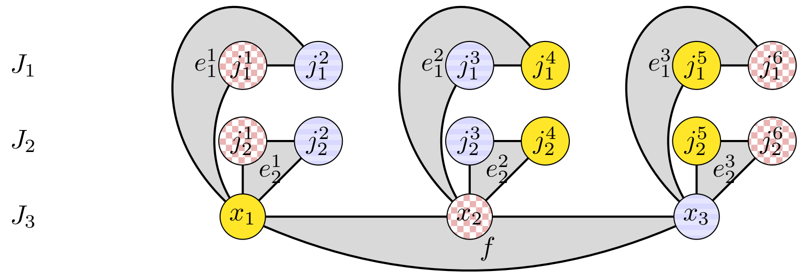

Figure 4 All fifteen relevant nodes together with the hyperedges within them. The heights of the vertices encode their online color (J_1, J_2, J_3), while their printed colors denote their offline color. Any assignment of the node x_3, given the other nodes and induced hyperedges, leads to a makespan of 3.

Once the six edges in question are found, one can insert nodes x_1, x_2, x_3 in this order, revealing hyperedges as depicted in Figure 4. To avoid a monochromatic edge, both x_1 and x_2 must be assigned the third online color. Keeping this in mind, it follows that x_3 cannot be assigned an online color without creating a monochromatic edge with respect to the online coloring. Together with the fact that the offline coloring indeed has makespan 1, we obtain:

▶ **Lemma 15.** *Any algorithm that assigns at least 7 copies of $\mathcal{S}$ to at most two online colors each is no better than 3-competitive.*

Subhypergraph $\mathcal{S}$ on Three Online Colors: The Palettes. It remains to consider the case where copies of the subhypergraph $\mathcal{S}$ are partitioned among the three colors, i.e., its nodes $\{1, \dots, 7\}$ are partitioned in nonempty sets $J_1 \dot\cup J_2 \dot\cup J_3$.

▶ **Lemma 16.** *For every online coloring $J_1 \dot\cup J_2 \dot\cup J_3$ of the nodes $V(\mathcal{S}) = \{v_1, \dots, v_7\}$ for nonempty sets $J_1 \dot\cup J_2 \dot\cup J_3$, there is an offline coloring $c\colon V(\mathcal{S}) \to \{\text{red}, \text{blue}, \text{yellow}\}$ such that two vertices exist with different online colors but the same offline color.*

In our construction, we exploit this property as follows: Let us consider a copy $\mathcal{S}(t)$ and fix two of its nodes $v_i(t) \in J_i$, $v_{i'}(t) \in J_{i'}$ provided by Lemma 16 as well as a third node $v_{i''}(t) \in J_{i''}$, where $i'' \notin \{i, i'\}$. By their choice, the nodes $v_i(t)$ ($i \in \{1, 2, 3\}$) admit at most two offline colors and avoid a third offline color $C \in \{\text{blue}, \text{red}, \text{yellow}\}$.

In this case, the subhypergraph $\mathcal{S}(t)$ is called a C *palette* (i.e., *yellow palette*, *blue palette* or *red palette*) and the corresponding nodes $v_i(t)$ are called *palette nodes*. In particular, $c(v_i(t)) \neq C$ holds for the palette nodes of a C palette. The choice of the name is motivated by the fact that, if we connect a new node w to e.g. yellow palette nodes, we are still allowed to extend the offline coloring by $c(w) = \text{yellow}$, while maintaining that the makespan of the offline coloring is 1.

The Construction. Below is a summary of our construction for the case where Lemma 15 is not applicable. Essentially, we create copies of the subhypergraph $\mathcal{S}$ which we connect with the existing hypergraph. The online color of the node v is denoted by $\varphi(v)$ and its (current) offline color by $c(v)$.

(i) We create up to 13 copies of the subhypergraph $\mathcal{S}$. By Lemma 15, we may assume that at least 7 of the copies are assigned to all three online colors (we truncate this step as soon as 7 such copies can be found). We handpick 7 such copies of $\mathcal{S}$, calling them $\mathcal{S}(1), \dots, \mathcal{S}(7)$.

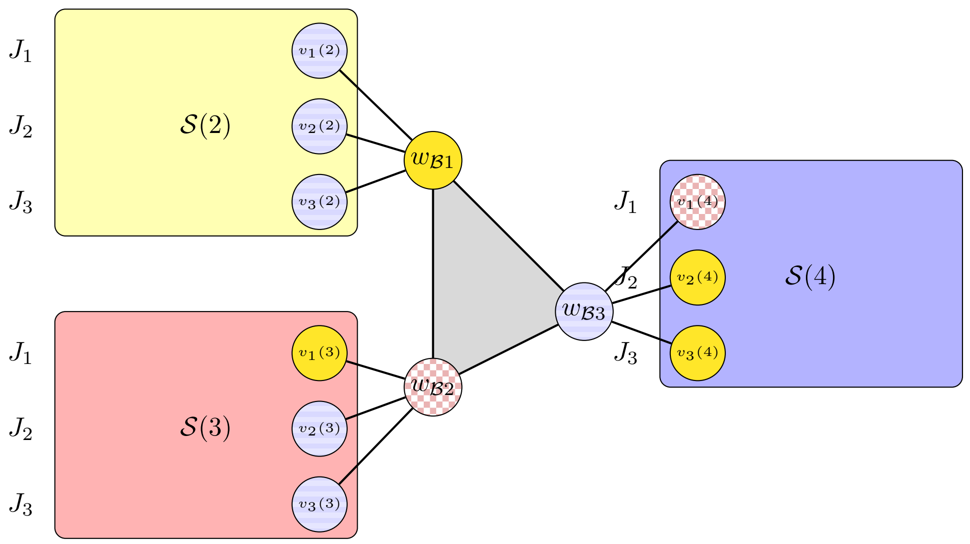

Figure 5 An example constellation of the three palettes together with the nodes $w_{\mathcal{B}t}$. The y-axis in the palettes denote the online colorings of the palette nodes respectively. The offline colors of the palette nodes might look differently, but they are never the same color as the palette, which is depicted as the color of the rectangles. The online coloring of the nodes $w_{\mathcal{B}i}$ are not yet known.

(ii) We permute the offline coloring on each copy of $\mathcal{S}$ so that $\mathcal{S}(1)$, $\mathcal{S}(4)$ and $\mathcal{S}(7)$ are yellow palettes, $\mathcal{S}(2)$ and $\mathcal{S}(5)$ are blue palettes, and $\mathcal{S}(3)$ and $\mathcal{S}(6)$ are red palettes.

(iii) We reveal a node $w_{\mathcal{A}}$ which is connected to palette nodes $v_i(1) \in V(\mathcal{S}(1)) \cap J_i$ for $i \in \{1,2,3\}$. The assignment to the online color $\varphi(w_{\mathcal{A}})$ leads to a monochromatic hyperedge $\{v_{\varphi(w_{\mathcal{A}})}(1), w_{\mathcal{A}}\}$ of online color $\varphi(w_{\mathcal{A}})$. We define $v_{\mathcal{A}} := v_{\varphi(w_{\mathcal{A}})}(1)$.

(iv) We reveal three nodes $w_{\mathcal{B}1}$, $w_{\mathcal{B}2}$ and $w_{\mathcal{B}3}$ that are connected by the hyperedge $\{w_{\mathcal{B}1}, w_{\mathcal{B}2}, w_{\mathcal{B}3}\}$ to each other. Moreover, for each $s \in \{1,2,3\}$, the node $w_{\mathcal{B}s}$ is connected to the palette nodes $v_i(s+1)$ for $i \in \{1,2,3\}$. There is an $i \neq \varphi(w_{\mathcal{A}})$ such that $\{v_i(s+1), w_{\mathcal{B}s}\}$ is a monochromatic edge of online color i for a suitable copy $\mathcal{S}(s)$, $s \in \{1,2,3\}$. For this s, we define $v_{\mathcal{B}} := v_i(s+1)$ and $w_{\mathcal{B}} := w_{\mathcal{B}s}$.

(v) We reveal further copies of $\mathcal{S}$ together with edges described below, until a copy $\mathcal{C}$ is assigned to all three online colors. In the copy $\mathcal{C}$, we denote the nodes $v_i(t)$ by $w_{\mathcal{C}t}$ for clarity. The nodes $w_{\mathcal{C}t}$ of the copy $\mathcal{C}$ are connected to the palettes $\mathcal{S}(5)$, $\mathcal{S}(6)$, $\mathcal{S}(7)$ by edges depending on their offline colors, in that nodes $w_{\mathcal{C}t}$ with $c(w_{\mathcal{C}t}) = C$ are connected to the palette nodes $v_i(s)$ for all $i \in \{1,2,3\}$ and $s \in \{5,6,7\}$ such that $\mathcal{S}(s)$ is a C palette, see Figure 6. As the nodes $w_{\mathcal{C}t}$ are assigned to all three online colors, a makespan of 2 is also achieved in the online color $i \notin \{\varphi(w_{\mathcal{A}}), \varphi(w_{\mathcal{B}})\}$. More precisely, there exists a monochromatic edge $\{v_i(s), w_{\mathcal{C}t}\}$ of the online color i. We rename $v_{\mathcal{C}} := v_i(s)$ and $w_{\mathcal{C}} := w_{\mathcal{C}t}$.

(vi) We permute the offline coloring in each of the connected components created in (iii), (iv) and (v) again, so that $c(v_j) = \text{blue}$ and $c(w_j) = \text{yellow}$ for $j \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$.

(vii) We add a final node v_n together with hyperedges $\{v_n, v_j, w_j\}$ for each $j \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$. Now, the online color $\varphi(v_n)$ that this node is assigned to admits a makespan of 3, while extending the offline coloring by $c(v_n) = \text{red}$ maintains an offline coloring with makespan 1.

Notice that we have maintained an offline coloring of the nodes with makespan 1 so far up to the very last node, and the last node n neighbors only blue and yellow nodes by construction and therefore the offline assignment $c(n) = \text{red}$ does not increase the makespan.

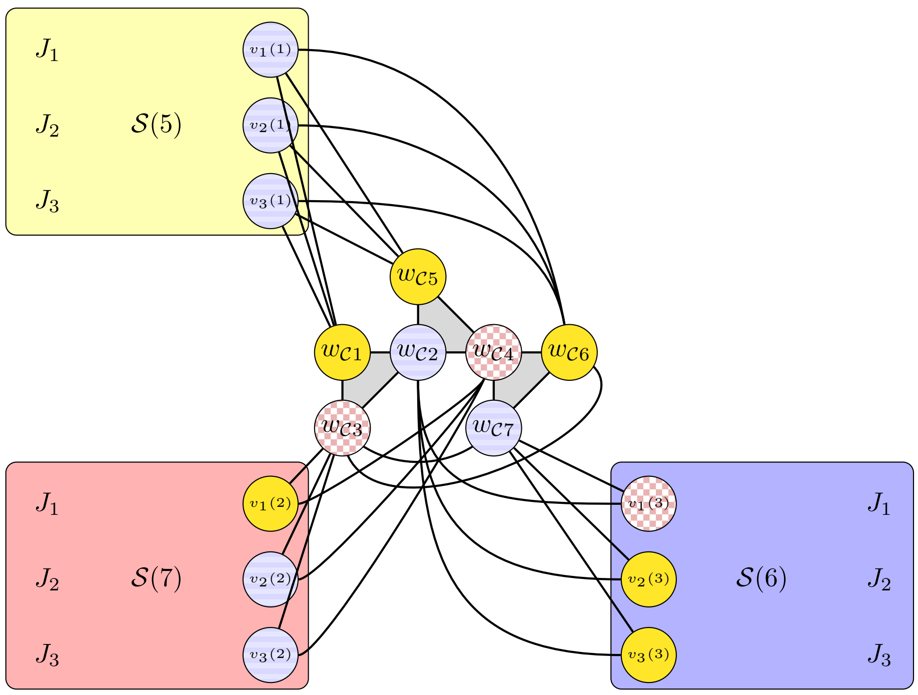

Figure 6 Partial sketch of subinstance $\mathcal{C}$.

In total, this yields that the offline optimal value is 1. A careful enumeration of introduced hyperedges reveals that 233 hyperedges suffice to create a monochromatic hyperedge for any of the cases demonstrated, which proves Theorem 13.

A Note on Theorem 2. Here, we abstain from presenting a proof of Theorem 2. Our construction in this general case is also based on incrementally creating monochromatic edges of each online color. However, we are faced with two fundamental challenges:

(i) The monochromatic hyperedges increase in size rather slowly, yet unless they have size of $m-1$, they are not strong enough to act as a palette for other hyperedges in the same sense as here.

(ii) To use our construction for Corollary 3, we restrict to hypertrees. This is a significant constraint, as it roughly implies that every palette may be used at most once.

Therefore, we include a more intricate recursive construction with much larger hypergraphs in the full version of the paper.

5 Tight Bounds For $K = 3$ and Unit Processing Times

In the previous section, we have established that for large enough K, there is no better-than-3-competitive algorithm for OMSS$(3, K)$, even for unit processing times. Let us reverse the parameters and ask: How competitively can we solve OMSS$(m, 3)$? This is not an easy question, indeed, even for OMSS$(m, 1)$, $m \geq 4$, a tight bound is not known. Restricting to the unit processing time case ($p_j \equiv 1$), we again obtain a tight bound, which is the smallest possible ratio (due to Theorem 5):

▶ **Theorem 17.** *For all $m \in \mathbb{N}$, there is a 2-competitive algorithm for the special case of OMSS$(m, 3)$ where the jobs have unit processing times.*

The algorithm in question employs an adapted round-robin method which assigns jobs to machines according to the scenarios they appear in. Essentially, we try to assign jobs $j \in S_1 \setminus (S_2 \cup S_3)$ and $j' \in (S_2 \cup S_3) \setminus S_1$ in a similar manner, and analogously for cyclic permutations of the indices $i \in \{1, 2, 3\}$. This gives a partition of jobs $j \in [n] \setminus (S_1 \cap S_2 \cap S_3)$ in three categories. The round robin assignments for each category are translated by roughly $m/3$, and jobs $j \in S_1 \cap S_2 \cap S_3$ are handled separately.

The algorithm is best visualized by drawing an analogue to a bingo card, for which we refer the reader to the full version.

6 Conclusion and Outlook

Throughout, we have established several competitiveness results for Online Makespan Scheduling under Scenarios. The first main takeaway from our work is a competitiveness gap between two and at least three scenarios when we consider $m = 2$ machines. This result draws a parallel to the known tractability results of several other problems, in which the line between easy and hard was drawn between two and three scenarios. Surely, the fact that the subsets of $\{1, 2, 3\}$ are not laminar, has been a deciding factor in our simple non-competitiveness result, as has been in several of the well-known results. Interestingly, restricting to the proxy competitive ratio as well as fixing the method of assigning double-scenario jobs essentially reduced the existence problem to the description of a polyhedron. We believe that similar techniques might prove useful for other cases and related problems as well.

We have further compared the behavior of competitiveness for increasing number K of scenarios and increasing number m of machines. In the special setting of unit processing times and $m = 3$ machines, the contrast is evident: On one hand, there is a 2-competitive algorithm for three scenarios and m machines for all $m \in \mathbb{N}$, which matches the lower bound obtained readily for $m = 2$. On the other hand, for K sufficiently large, the trivial upper bound of 3 cannot be beaten. That being said, $K + 1$ is a strict upper bound as well for a problem on K scenarios. To summarize, for large values of m and small values of K, we would expect the possibilities for competitive ratios to be dominated by K.

In light of our results, we may raise several open questions: In addition to the obvious task of finding tight bounds for all possibilities (m, K) of the numbers of machines and scenarios, exploring the competitiveness as m approaches to infinity in the weighted processing time case remains interesting. However, these problems have not been resolved even for $K = 1$, whence it is fair to assume challenges in further progress. Another interesting research direction is to introduce *migration*, as seen in e.g. [20]. In this model, we are allowed to revoke a bounded size of decisions in hindsight. Similarly, instead of minimizing the worst-case scenario, one could look into minimizing the average scenario or maximum regret.

References

1 Susanne Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999. doi:10.1137/S0097539797324874.

2 Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51(3):359–366, 1995. doi:10.1006/jcss.1995.1074.

3 Dimitris Bertsimas, Patrick Jaillet, and Amedeo R. Odoni. A priori optimization. *Operations Research*, 38(6):1019–1033, 1990. doi:10.1287/OPRE.38.6.1019.

4 Danny Blom, Dylan Hyatt-Denesik, Afrouz Jabal Amelia, and Bart Smeulders. Approximation algorithms for k-scenario matching. In Marcin Bieńkowski and Matthias Englert, editors, *Approximation and Online Algorithms*, pages 89–103, Cham, 2025. Springer Nature Switzerland.

5 Yulle Borges, Vinicius Loti de Lima, Flávio Miyazawa, Lehilton Pedrosa, Thiago Queiroz, and Rafael Schouery. Algorithms for the bin packing problem with scenarios. *CoRR*, May 2023. doi:10.48550/arXiv.2305.15351.

6 Thomas Bosman, Martijn van Ee, Ekin Ergen, Csanád Imreh, Alberto Marchetti-Spaccamela, Martin Skutella, and Leen Stougie. Total completion time scheduling under scenarios. In *Proceedings of the 21st International Workshop on Approximation and Online Algorithms*, pages 104–118. Springer, 2023. doi:10.1007/978-3-031-49815-2_8.

7 Attila Bódis and János Balogh. Bin packing problem with scenarios. *Central European Journal of Operations Research*, 27(2):377–395, June 2019. doi:10.1007/s10100-018-0574-3.

8 Moses Charikar, Alantha Newman, and Aleksandar Nikolov. Tight hardness results for minimizing discrepancy. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1607–1614, January 2011. doi:10.1137/1.9781611973082.124.

9 Ulrich Faigle, Walter Kern, and Gyorgy Turan. On the performance of on-line algorithms for partition problems. *Acta Cybern.*, 9:107–119, January 1989. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3359.

10 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.

11 Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. In Mike S. Paterson, editor, *Algorithms - ESA 2000*, pages 202–210, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. doi:10.1007/3-540-45253-2_19.

12 Gábor Galambos and Gerhard J. Woeginger. An on-line scheduling heuristic with better worst-case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22(2):349–355, 1993. doi:10.1137/0222026.

13 R. L. Graham. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, 45(9):1563–1581, 1966. doi:10.1002/j.1538-7305.1966.tb01709.x.

14 Yaqiao Li and Denis Pankratov. Online vector bin packing and hypergraph coloring illuminated: Simpler proofs and new connections. *CoRR*, June 2023. doi:10.48550/arXiv.2306.11241.

15 J. Nagy-György and Cs. Imreh. Online hypergraph coloring. *Information Processing Letters*, 109(1):23–26, 2008. doi:10.1016/j.ipl.2008.08.009.

16 Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *J. ACM*, 29(2):285–309, April 1982. doi:10.1145/322307.322309.

17 Karger D. R., Phillips S. J., and Torng E. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 1996.

18 John F. Rudin. *Improved bounds for the on-line scheduling problem.* PhD thesis, The University of Texas at Dallas, 2001.

19 John F. Rudin and R. Chandrasekaran. Improved bounds for the online scheduling problem. *SIAM Journal on Computing*, 32(3):717–735, 2003. doi:10.1137/S0097539702403438.

20 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming*, pages 1111–1122, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-27836-8_92.

21 Dvir Shabtay and Miri Gilenson. A state-of-the-art survey on multi-scenario scheduling. *European Journal of Operational Research*, 2022.

22 Joel Spencer. Balancing games. *Journal of Combinatorial Theory, Series B*, 23(1):68–74, August 1977. doi:10.1016/0095-8956(77)90057-0.

23 Joel Spencer. Six standard deviations suffice. *Transactions of the American Mathematical Society*, 289(2):679–706, June 1985. Copyright: Copyright 2016 Elsevier B.V., All rights reserved. doi:10.1090/S0002-9947-1985-0784009-0.

24 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.