Online Scheduling on Identical Machines with a Metric State Space

Hiromichi Goko
Toyota Motor Corporation, Aichi, Japan

Akitoshi Kawamura
Kyoto University, Japan

Yasushi Kawase
University of Tokyo, Japan

Kazuhisa Makino
Kyoto University, Japan

Hanna Sumita
Tokyo Institute of Technology, Japan

Abstract

This paper introduces an online scheduling problem on m identical machines with a metric state space, which generalizes the classical online scheduling problem on identical machines, the online traveling salesman problem, and the online dial-a-ride problem. Each job is associated with a source state, a destination state, a processing time, and a release time. Each machine can process a job on and after its release time. Before processing a job, a machine needs to change its state to the source state (in a time corresponding to the distance), and after the process of the job, the machine's state becomes the destination state. While related research deals with a model in which only release times are unknown to the algorithm, this paper focuses on a general model in which destination states and processing times are also unknown. The main result of this paper is to propose a $O(\log m/\log\log m)$-competitive online algorithm for the problem, which is best possible. A key approach is to divide the difficulty of the problem. To cope with unknown release times, we provide frameworks to produce a $\min\{2\rho+1/2,\ \rho+2\}$-competitive algorithm using a ρ-competitive algorithm for a basic case where all jobs are released at time 0. Then, focusing on unknown destination states and processing times, we construct an $O(\log m/\log\log m)$-competitive algorithm for the basic case. We also provide improved algorithms for some special cases.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online scheduling, Competitive analysis, Online dial-a-ride

Digital Object Identifier 10.4230/LIPIcs.STACS.2022.32

Funding This work was partially supported by the joint project of Kyoto University and Toyota Motor Corporation, titled "Advanced Mathematical Science for Mobility Society", JST PRESTO Grant Number JPMJPR2122, and JSPS KAKENHI Grant Numbers JP17K12646, JP19K22841, JP20H05967, JP20K19739, JP21K17708, and JP21H03397.

1 Introduction

For a metric space $\boldsymbol{M} = (X, d)$ and a positive integer m, we consider the following $(\boldsymbol{M}, m)$-*scheduling problem*. We have m identical machines, which work in parallel. Each machine i has a *state* $s_i(t)$ in X at time $t \in \mathbb{R}_+$, and moves along a path P in $\boldsymbol{M}$ to change a state x to a state y, where each machine is assumed to move in $\boldsymbol{M}$ with at most unit speed. All machines are initially (i.e., at time 0) located at the origin o in X and need to return to o at the end. Machines process jobs given in an online fashion. Each job j has two states called the *source* state $a_j \in X$ and *destination* state $b_j \in X$. It also has the processing time

$p_j \in \mathbb{R}_+$ and the release time $r_j \in \mathbb{R}_+$. Job j appears on r_j, can be processed on and after r_j, and requires p_j time to process it. To processes job j, machine i first needs to change its state to the source state a_j and reaches the destination state b_j afterward. We assume that $p_j \geq d(a_j, b_j)$, which means that $p_j - d(a_j, b_j)\,(\geq 0)$ time is additionally required to process the job j. The jobs are *non-preemptive*, i.e., once a machine starts to process a job, it must finish the processing of the job without doing anything else. In addition, any job has to be processed by exactly one machine. The $(\boldsymbol{M}, m)$-scheduling problem is to find a schedule to minimize the *makespan*, that is, the time when all machines return to the origin o after completing all jobs.

We study the *real-time online* version of this scheduling problem, called the *online* $(\boldsymbol{M}, m)$*-scheduling problem*. No information on jobs is available before their release time. Namely, we first see the job j at time r_j, and we do not know if it comes or not before time r_j. There exist two information models, called the *complete* and *incomplete* models, for the online $(\boldsymbol{M}, m)$-scheduling problem, where we mainly study the incomplete model. In the complete model, all the information of the job j are revealed at the release time r_j. On the other hand, in the incomplete model, the processing time p_j and destination state b_j are still unknown at time r_j and revealed when the job j is completed by some machine. Several fundamental cases of the online $(\boldsymbol{M}, m)$-scheduling problem in the complete information model have been studied in the literature of scheduling and combinatorial optimization [1–10, 17, 18], as described in the next subsection. On the other hand, little is known for the incomplete information model [19, 21]. However, there are a number of practical situations for which the incomplete model is suitable. For example, in the scheduling problems such as repairing companies, we do not know in advance the *exact* processing time for jobs as well as their situations (or states) when finishing them. Classical taxi-hailing services only collect the pick-up locations when customers phone the companies, and elevator systems are usually equipped with only landing call buttons, although the systems can get information on the directions (i.e., up and down) of the requests. Our goal is to design online algorithms for the online $(\boldsymbol{M}, m)$-scheduling problem under several natural settings of $\boldsymbol{M}$ and m. We also consider designing online algorithms for the *basic online* $(\boldsymbol{M}, m)$*-scheduling problem*, where all jobs are released at time 0, since it turns out that any algorithm for the basic problem can be extended to the one for the general problem. We analyze the performance of online algorithms by the *competitive ratio*, that is, the ratio between the optimal makespan and the one of the schedule obtained by the online algorithm.

1.1 Previous work

One of the simplest cases of the online $(\boldsymbol{M}, m)$-scheduling problem is the case when the metric space consists of a single point, i.e., the states of the machines are fixed. We denote the trivial metric as $\mathbb{R}^0$. For the online $(\mathbb{R}^0, m)$-scheduling problem, the following greedy algorithm is $(2 - 1/m)$-competitive, and this is best possible [13–15, 21]: assign an unprocessed job to any available machine anytime if possible. This result is regardless of information models. Shmoys et al. [21] focus on the incomplete information model and introduced a technique to convert a ρ-competitive algorithm for the basic online $(\mathbb{R}^0, m)$-scheduling problem into a 2ρ-competitive one for the general online $(\mathbb{R}^0, m)$-scheduling problem.

An important special case of the problem is when the processing of each job does not change the state, i.e., $a_j = b_j$ for all jobs j. Such a situation appears in a production system with sequence-dependent setup or changeover, e.g., mold setup, die setup, or color setup [16, 20]. To process a job in a plastic production system, we must attach the corresponding injection mold to an injection machine. Thus, we need setup time before and after processing the job.

Gambosi and Nicosia [12] studied an online version (in the one-by-one model) of scheduling with setup costs. Furthermore, a particular case where the processing time of every job equals 0 is studied as the online traveling salesman problem (TSP) [2, 3, 7, 8, 17]. The competitive ratio of the online TSP is 2 for any metric and any number of machines [2, 17]. In addition, the competitive ratio of the single-server ($m = 1$) online TSP with the line metric is $(9 + \sqrt{17})/8 \approx 1.64$ [2, 3, 7]. We remark that the online TSP is included in the complete information model.

When the processing time of each job is equal to its travel distance, i.e., $p_j = d(a_j, b_j)$ for all jobs j, our problem is studied under the name of the *online dial-a-ride problem*. There is a large body of studies on the online dial-a-ride problem [1, 4–6, 9, 10, 18, 19], but only Lipmann et al. [19] addresses the incomplete information model. Lipmann et al. [19] showed that upper and lower bounds of the single machine online dial-a-ride problem are 4 and $1 + \frac{3}{2}\sqrt{2}$ (≈ 3.121), respectively. For the upper bound, they designed an algorithm, called BOUNCER, which decides a process order based on a solution to the online TSP over the source states. In the algorithm, every time the machine completes a job, the machine changes the state to the source state of the job. Thus, the makespan is the length of the TSP tour plus twice the sum of processing times. Because a 2-competitive algorithm is known for the online TSP, the BOUNCER algorithm is 4-competitive. Note that the BOUNCER algorithm is easily extended to the online $(\boldsymbol{M}, m)$-scheduling problem, but its competitive ratio is $2 + 2m$. We also note that in their paper, the processing time and destination state of a job are revealed at the moment when some machine starts processing the job. When $m = 1$ and preemption is not allowed, this is the same as our model. However, our model is more general when $m > 1$. For the online dial-a-ride problem under the complete information model, Ascheuer et al. [1] analyzed two natural algorithms, which they call IGNORE and REPLAN, for the single machine setting. They also provide an algorithm called SMARTSTART, which is 2-competitive for any metric and any number of machines. This is best possible since a lower bound of the competitive ratio is 2 even for the online TSP [2].

1.2 Our results

Our main result is to provide an $O(\log m / \log \log m)$-competitive algorithm (Theorem 8) and prove that this is best possible up to a constant factor (Theorem 7) among algorithms for the general $(\boldsymbol{M}, m)$-scheduling problem. We summarize our results and existing ones in Table 1.

We describe the techniques to obtain our results. Our approach is to divide the difficulty of our problem into two types of unknown information; one is a release time (online arrival), and the other consists of a destination state and a processing time.

First, to cope with the unknown release times, we show a framework to design an algorithm using one for the basic online $(\boldsymbol{M}, m)$-scheduling problem. We describe this in Section 3. Roughly speaking, our framework computes a schedule by repeatedly applying the basic case algorithm. As rules for applying the algorithm, we adopt three natural strategies, called IGNORE, REPLAN, and SMARTSTART, proposed for the online dial-a-ride problem in the complete information model [1]. We analyze these strategies in detail using three factors that contribute to the makespan. The analysis implies that a ρ-competitive algorithm for the basic online $(\boldsymbol{M}, m)$-scheduling problem can be converted into a $\min\{2\rho + 1/2, \rho + 2\}$-competitive algorithm for the general case (Corollaries 2 and 4). We remark that this extends the result by Shmoys et al. [21]. They deal with only the fixed state case (i.e., $\boldsymbol{M} = \mathbb{R}^0$), and their method results in a 2ρ-competitive algorithm. If ρ is greater than 2, our method leads to a better competitive ratio. Consequently, we only need to focus on the basic case.

Then, focusing only on unknown destination states and processing times, we present an $O(\log m/\log\log m)$-competitive algorithm for the basic online $(\boldsymbol{M}, m)$-scheduling problem (Algorithm 1) in Section 4.1. Our algorithm first partition the jobs into m groups and assigns one group to each machine based only on the information of source states. To shorten the makespan, we set machines that complete their assigned groups to process a group that has not yet complete. It is not a good idea to assign additional machines to an uncompleted group immediately and greedily: this method may cause a machine making a vain effort to assist one completing soon. Thus, we control reassignments. Namely, we appropriately wait and reassign jobs so that the number of assisting machines increases exponentially with base q where $q^q > m \geq (q-1)^{q-1}$. Our analysis is not only elementary but also it shows a tight competitive ratio for the basic problem. This result together with the above conversion implies our main algorithmic result. We can see the ratio is the best possible because we show that any online algorithm is $\Omega(\log m/\log\log m)$-competitive for the basic online $(\boldsymbol{M}, m)$-scheduling problem. Note that this lower bound also holds for the cases of (i) $p_j = d(a_j, b_j)$ for all job j, (ii) $a_j = b_j$ for all job j, and (iii) the destination state and the processing time of each job are revealed at the moment when some machine starts processing the job.

We also discuss the competitive ratio of special cases in Section 5. One is a single machine case. We show that Algorithm 1 is 3-competitive for the basic online $(\boldsymbol{M}, 1)$-scheduling problem, and prove that no online algorithm has the competitive ratio better than 2.255 and 3.181 for the basic and the general problems, respectively. We note that our bound of 3.181 improves the best known lower bound of $1 + \frac{3\sqrt{2}}{2} \approx 3.12$ for the online dial-a-ride problem under the incomplete information model [19]. Another is a two-machine case. We improve Algorithm 1 to obtain a 3.5-competitive and a $(4 + \frac{\sqrt{3}}{2})$-competitive algorithm for the basic and the general online $(\boldsymbol{M}, 2)$-scheduling problem, respectively. In addition, when the optimal values of the TSP with a single machine and m machines are close, we provide a 13/3-competitive algorithm for the basic online $(\mathbb{R}, m)$-scheduling problem.

Finally, we discuss several other variants of our problem in Section 6: minimizing the total completion time is hopeless, the open setting (the machines do not need to return to the origin) can easily be reduced to the closed setting (the machines do not need to return to the origin) with loss of factor 2, and preemption does not help if the destination state and the processing time of each job are revealed upon completion.

Due to space limitations, some proofs are deferred to Appendix A.

Table 1 Summary of our results and previous work.

release time	# machines	incomplete info. model: upper bound	incomplete info. model: lower bound	complete info. model: upper bound	complete info. model: lower bound
general	m	$\boldsymbol{\Theta}(\frac{\log m}{\log\log m})$ (Thms. 8, 7)		2 [1]	
	1	4 (Thm. 13)	**3.181** (Thm. 15)		
0 (basic)	m	$\boldsymbol{\Theta}(\frac{\log m}{\log\log m})$ (Thms. 9, 7)		1	
	1	**3** (Thm. 12)	**2.255** (Thm. 14)		

2 Preliminaries

An instance of the online $(\boldsymbol{M}, m)$-scheduling problem is specified by a metric space $\boldsymbol{M} = (X, d)$ with a distinguished origin $o \in X$, the number of machines m, and a set of jobs J. Let $[m] = \{1, 2, \ldots, m\}$ be the set of machines. Note that $d\colon X \times X \to \mathbb{R}_+$ is a function such

that for any $x, y, z \in X$, the following holds: (i) $d(x, y) = 0 \iff x = y$, (ii) $d(x, y) = d(y, x)$, and (iii) $d(x, z) \leq d(x, y) + d(y, z)$. We assume that $\boldsymbol{M}$ is path-connected, i.e., for any pair of points $x, y \in X$, there is a continuous path $\gamma\colon [0, 1] \to X$ with $\gamma(0) = x$ and $\gamma(1) = y$ of length $d(x, y)$. Examples of such metric spaces are the real line $\mathbb{R}$, the Euclidean plane $\mathbb{R}^2$, and a circle $\mathbb{R}/\mathbb{Z}$, where we assume that each set has the Euclidean distance. Each job $j \in J$ is associated with a tuple $(a_j, b_j, p_j, r_j) \in X \times X \times \mathbb{R}_+ \times \mathbb{R}_+$, where a_j and b_j are respectively the source and the destination states, p_j is the processing time, and r_j is the release time of job j. When processing a job j by a machine in state s, it is first necessary to change the state of machine to a_j in time $d(s, a_j)$. Then after the machine starts to process job j, the machine's state is changed continuously from a_j, and finally reaches b_j. Possibly the process is done without state change (e.g., injection molds remain unchanged during production of one item). A job j is called *empty* if $a_j = b_j$ and $p_j = 0$. Assume that the state of each machine i is the origin o at time 0 (i.e., $s_i(0) = o$), the state can be changed in at most unit speed (i.e., $d(s_i(t), s_i(t')) \leq |t - t'|$ for all $i \in [m]$ and $t, t' \in \mathbb{R}_+$), and the processing time p_j is at least $d(a_j, b_j)$ for all job j. Each machine can process at most one job at a time. We do not allow preemption; once a machine starts processing a job, it is not permitted to stop until the process is completed. The objective of the problem is to minimize the completion time (makespan), which is the time when the machines have completed all jobs and returned to o.

A (feasible) *schedule* is a sequence of processing and states change of the machines satisfying the following: (1) the state of each machine is the origin o at time 0 and at the end, (2) the state of each machine is changed in at most unit speed, and (3) every job is processed on or after its release time. An online algorithm decides a partial schedule in real-time. We focus on the *incomplete* information model in which an online algorithm has no information about the destination state and the processing time of each job until the job is completed. The source state of each job is revealed at its release time.

We evaluate the performance of an online algorithm by the *competitive ratio*. Throughout the paper, we only care about deterministic online algorithms. For a problem instance I, we denote the completion time by an algorithm ALG and an optimal offline algorithm OPT by $\mathrm{ALG}(I)$ and $\mathrm{OPT}(I)$, respectively. We assume that the optimal offline algorithm knows the information of all the jobs in advance. An algorithm ALG is said to be c-competitive if $\mathrm{ALG}(I) \leq c \cdot \mathrm{OPT}(I)$ for any instance I of the online (basic) $(\boldsymbol{M}, m)$-scheduling problem. We refer to the schedule of the optimal offline algorithm as the optimal offline schedule.

3 Reduction to the Basic Problem

In this section, we consider three natural strategies, called IGNORE, REPLAN, and SMARTSTART, to convert an algorithm for the basic online $(\boldsymbol{M}, m)$-scheduling problem into an algorithm for the online $(\boldsymbol{M}, m)$-scheduling problem. These strategies are applied to the (complete information) online dial-a-ride problem in [1]. IGNORE is a strategy that runs an optimal subsequent schedule (which can be computed in the complete information setting) for unprocessed jobs if they exist, ignoring all new jobs until all machines finish the assigned jobs and come back to the origin o. REPLAN is a greedy strategy that stops the current plan whenever a new job is released, makes all the machines return to the origin o right after the current process, and starts a replanned optimal subsequent schedule for the remaining jobs. We remark that these two strategies decide when to start independently of processing times, while SMARTSTART uses the information. SMARTSTART is similar to IGNORE, but it may keep the machines idle for a time depending on an estimated processing time of

remaining jobs (which is, for example, calculated by assuming that the processing time is all zero). These strategies do not move machines until some jobs are released, and hence we do not need to consider the instance with no jobs, called the empty instance, in the competitive analysis.

Let ALG_0 be an algorithm for the basic online $(\boldsymbol{M}, m)$-scheduling problem, and let I_0 be an instance. We analyze our conversions by the following three factors that contribute to the makespan: (i) the optimal completion time $\mathrm{OPT}(I_0)$, (ii) the mean sum $p(I_0)/m$ of processing times per machine, where $p(I_0) = \sum_{j \in J} p_j$ for the jobs in I_0, and (iii) the lower bound $\mathrm{LB}(I_0)$ on the minimum completion time for I_0, which is given by the optimal value of the *m-traveling salesman problem* (m-TSP) over the source states. We refer to the m-TSP as the problem of finding m tours that visit every location so as to minimize the length of the longest tour [11]. The 1-TSP coincides with TSP. In what follows, we assume that

$$\mathrm{ALG}_0(I_0) \leq \alpha \mathrm{OPT}(I_0) + \beta \cdot p(I_0)/m + \gamma \mathrm{LB}(I_0), \tag{1}$$

where α, β, and γ are nonnegative reals independent from I_0. Note that ALG_0 is $(\alpha+\beta+\gamma)$-competitive because $\mathrm{OPT}(I_0) \geq p(I_0)/m$ and $\mathrm{OPT}(I_0) \geq \mathrm{LB}(I_0)$. In addition, $\alpha + \gamma \geq 1$ because $\mathrm{ALG}_0(I_0) \leq (\alpha+\gamma) \cdot \mathrm{OPT}(I_0)$ for any instance I_0 with $p(I_0) = 0$. For a set S' of jobs, we denote by S'_0 the instance of the basic $(\boldsymbol{M}, m)$-scheduling problem with job set S'.

We first consider the following IGNORE algorithm. The machines remain idle (i.e., the machines are in the origin and not working) until a non-empty set S of unprocessed jobs appear. The algorithm then immediately processes S following the schedule obtained by ALG_0 for S_0. We refer to this schedule as a subschedule for S. All jobs that arrive during the execution of the subschedule are temporarily ignored until the subschedule is completed and all machines become idle again. Then, the algorithm continues the same process.

▶ **Theorem 1.** *IGNORE is $(2\alpha+\beta+2\gamma+1/2)$-competitive for the online $(\boldsymbol{M}, m)$-scheduling problem.*

Proof. We fix a non-empty instance I. Let t^* be the last released time of jobs in I.

Suppose that the machines are idle at time t^*. Let R be the set of jobs processed in the last subschedule of the IGNORE algorithm. By construction, the length of the last subschedule is $\mathrm{ALG}_0(R_0) \leq \alpha \mathrm{OPT}(R_0) + \beta p(R_0)/m + \gamma \mathrm{LB}(R_0) \leq (\alpha+\beta+\gamma)\mathrm{OPT}(I)$. Since $t^* \leq \mathrm{OPT}(I)$ and $\alpha + \beta \geq 1$, it holds that

$$\begin{aligned} \mathrm{IGNORE}(I) &= t^* + \mathrm{ALG}_0(R_0) \leq t^* + (\alpha+\beta+\gamma) \cdot \mathrm{OPT}(I) \\ &\leq (1+\alpha+\beta+\gamma)\mathrm{OPT}(I) \leq (2\alpha+\beta+2\gamma+1/2)\mathrm{OPT}(I). \end{aligned}$$

Now suppose that some machines are not idle at time t^*. Then t^* is in the second last subschedule of the algorithm, and the last subschedule starts right after the second last subschedule ends. Let R and S be the sets of jobs processed in the last and the second last subschedule, respectively. We denote by $r' = \min_{(a,b,p,r) \in R} r$.

Let q_i be the state of machine i at time r' in the optimal offline schedule for I. Thus, $\max_{i \in [m]} d(o, q_i) \leq r' \leq \mathrm{OPT}(I)$ and $\mathrm{OPT}(I) \geq 2 \max_{i \in [m]} d(o, q_i)$. For the instance R_0, the minimum makespan is $\mathrm{OPT}(R_0)$. On the other hand, when we first move each machine i to state q_i, and then imitate an optimal offline schedule for I after time r', ignoring jobs not in R, the makespan will be $\max_{i \in [m]} d(o, q_i) + \mathrm{OPT}(I) - r'$. Thus, we see that $\mathrm{OPT}(R_0) \leq \max_{i \in [m]} d(o, q_i) + \mathrm{OPT}(I) - r'$. Hence, we obtain that

$$
\begin{aligned}
&\mathrm{IGNORE}(I)\\
&\le r' + \mathrm{ALG}_0(S_0) + \mathrm{ALG}_0(R_0)\\
&\le r' + (\alpha+\gamma)\mathrm{OPT}(S_0) + (\alpha+\gamma)\mathrm{OPT}(R_0) + \beta p(S_0)/m + \beta p(R_0)/m\\
&\le r' + (\alpha+\gamma)\mathrm{OPT}(I) + (\alpha+\gamma)\mathrm{OPT}(R_0) + \beta p(I_0)/m + (\alpha+\gamma-1)(r' - \max_i d(o,q_i))\\
&\le (\alpha+\gamma)\left(r' + \mathrm{OPT}(R_0) - \max_{i\in[m]} d(o,q_i)\right) + (\alpha+\beta+\gamma)\mathrm{OPT}(I) + \max_{i\in[m]} d(o,q_i)\\
&\le (2\alpha+\beta+2\gamma+1/2)\mathrm{OPT}(I).
\end{aligned}
$$

◀

▶ **Corollary 2.** *If there exists a ρ-competitive algorithm for the basic online $(\boldsymbol{M}, m)$-scheduling problem, then there exists a $(2\rho+1/2)$-competitive algorithm for the online $(\boldsymbol{M}, m)$-scheduling problem.*

Next, we consider the following REPLAN algorithm. When a new job is released, the REPLAN algorithm calls all the machines, and each machine comes back to the origin o immediately after completing the currently processing job. After all machines are returned, it processes the unprocessed jobs S following the schedule obtained by ALG_0 for S_0.

▶ **Theorem 3.** *REPLAN is $(\alpha+\beta+\gamma+2)$-competitive for the online $(\boldsymbol{M}, m)$-scheduling problem.*

Proof. We fix a non-empty instance I. Let t^* be the last release time of jobs appearing in I, and let R be the set of unprocessed jobs at time t^*.

At the time t^*, all machines are made to directly return to the origin o as soon as possible. A machine that is not processing any job (including on the way to some job) at time t^* can return in t^* ($\le \mathrm{OPT}(I)$) time, and a machine that is processing job j can return in $p_j + d(b_j, o)$ ($\le \mathrm{OPT}(I)$) time. Thus, all machines return in at most $\mathrm{OPT}(I)$ time. After all the machines have returned to the origin o, it takes $\mathrm{ALG}_0(R_0) \le \alpha\mathrm{OPT}(R_0) + \beta p(R_0)/m + \gamma\mathrm{LB}(R_0) \le (\alpha+\beta+\gamma)\mathrm{OPT}(I)$ time to complete all the jobs in R. Hence, the completion time by REPLAN is at most

$$
\mathrm{REPLAN}(I) \le t^* + \mathrm{OPT}(I) + (\alpha+\beta+\gamma)\mathrm{OPT}(I) \le (\alpha+\beta+\gamma+2)\mathrm{OPT}(I).
$$

◀

▶ **Corollary 4.** *If there exists a ρ-competitive algorithm for the basic online $(\boldsymbol{M}, m)$-scheduling problem, then there exists a $(\rho+2)$-competitive algorithm for the online $(\boldsymbol{M}, m)$-scheduling problem.*

Finally, we discuss the SMARTSTART strategy, which is originally proposed for the complete information model [1]. SMARTSTART calculates the minimum completion time T for the unprocessed jobs and waits until the time T. However, this is impossible in the incomplete information model. Therefore, we use $\mathrm{LB}(I_0)$ as an alternative parameter to decide when to start.

To be precise, the algorithm SMARTSTART has a fixed parameter $\theta \ge 0$, which will be optimized later. When the machines are idle, and there is a non-empty set S of unprocessed jobs, the algorithm computes $\mathrm{LB}(S_0)$. The algorithm keeps machines idle until the time $\theta \cdot \mathrm{LB}(S_0)$ if it is before the time. Then, it immediately processes S following the schedule obtained by ALG_0 for S_0. The algorithm ignores all jobs that arrived during the execution of the subschedule until the subschedule is completed and all machines become idle again.

▶ **Theorem 5.** *SMARTSTART is* $\frac{6\alpha+4\beta+4\gamma+1+\sqrt{(2\alpha+1)^2+8\gamma}}{4}$*-competitive for the online* $(\boldsymbol{M}, m)$*-scheduling problem by setting* $\theta = \frac{2\alpha+1+\sqrt{(2\alpha+1)^2+8\gamma}}{4}$.

We remark that similar statements to Theorems 1, 3, and 5 hold for other settings such as complete information and preemptive setting. Corollaries 2 and 4 yield that we only need to construct an $O(\log m/\log\log m)$-competitive algorithm for the basic case, and we do this in the next section. We mention that SMARTSTART derives better competitive algorithms than others for the single and two machine cases as shown Section 5.

4 Algorithm and Hardness of the Basic Problem

In this section, we show our main results for the basic online $(\boldsymbol{M}, m)$-scheduling problem. We first present an $O(\log m/\log\log m)$-competitive algorithm in Section 4.1, and then prove that this is best possible among deterministic online algorithms in Section 4.2.

Specifically, we show the following theorems.

▶ **Theorem 6.** *For any metric* $\boldsymbol{M}$*, there exists an* $O(\log m/\log\log m)$*-competitive algorithm for the basic online* $(\boldsymbol{M}, m)$*-scheduling problem.*

▶ **Theorem 7.** *There exists a metric* $\boldsymbol{M}$ *such that every online algorithm is* $\Omega(\log m/\log\log m)$*-competitive for the basic online* $(\boldsymbol{M}, m)$*-scheduling problem.*

We remark that Theorem 6 together with Theorem 1 or 3 in the previous section implies our main result for the general case.

▶ **Theorem 8.** *For any metric* $\boldsymbol{M}$*, there exists an* $O(\log m/\log\log m)$*-competitive algorithm for the online* $(\boldsymbol{M}, m)$*-scheduling problem.*

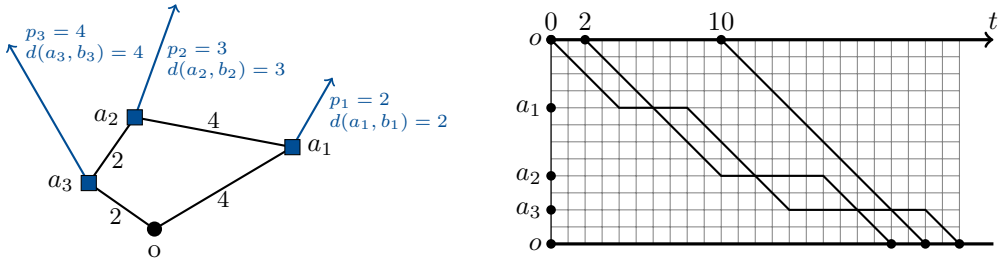
The lower bound on the basic case also applies to the general case, and hence we see that the competitive ratio of $O(\log m/\log\log m)$ is best possible.

4.1 Upper bound

Let I be an instance of the basic online $(\boldsymbol{M}, m)$-scheduling problem. Recall that we know the source states of all the jobs. A simple way to construct a reasonable schedule for the problem would be to use a shortest tour for the source states. More precisely, we solve the m-TSP over the source states. Let $C_1, \ldots, C_m$ be the optimal solution of the m-TSP over the source states. Then each machine i processes jobs appearing in (directed) tour C_i in the order of C_i. After processing a job, the machine returns to its source state and moves on to the next job. Unfortunately, the competitive ratio of this simple method is $\Omega(m)$ even when the metric is $\mathbb{R}^2$ (see Example 21 in Appendix).

The reason why the above simple method did not work well is that some machines spend time in idleness. We resolve this issue by carefully reassigning idle machines to an *uncompleted* tour. A tour is said to be completed if all the jobs appearing in the tour have been completed, and all the machines assigned to the tour have returned to the origin o. We describe the idea of our algorithm. We first assign machine i to each tour C_i, and each machine processes jobs in the assigned tour in the same way as the simple method. If a tour takes a long time to be completed, our algorithm additionally assigns idle machines to the tour. Each machine travels along the assigned tour. When a machine arrives at the source of an unprocessed job, then it processes the job, returns to its source, and continues the travel

(see Figure 1). The point of our algorithm is that it does not reassign machines immediately and greedily. Our algorithm reassigns machines so that the number of machines assigned to each uncompleted tour increases exponentially by the following rule. To avoid reassigning to a tour that will be completed soon, the algorithm waits for a certain number of idle machines before reassigning. This exponential increase in the number of assigned machines will help us analyze the competitive ratio of the algorithm.



(a) A tour C_i.

(b) Diagram of machines' locations.

Figure 1 A process of a tour when three machines assigned at time 0, 2, and 10, respectively.

Let q be a positive integer, which will be set later. The execution of our algorithm consists of phases. In phase k, our algorithm adds idle machines to each uncompleted tour so that the number of assigned machines for the tour is q^{k-1}. The phase k continues until the number of uncompleted tours decreases to $\lfloor m/q^k \rfloor$. The algorithm terminates in phase k^* with $m/q^{k^*-1} \geq 1 > m/q^{k^*}$, that is, $k^* = \lfloor \log_q m \rfloor + 1$ $(\leq q)$. Then, we set q so as to bound the makespan. Intuitively, the number of phases increases as q becomes smaller, and the time length of each phase increases as q becomes larger. We set q to the integer such that $q^q > m \geq (q-1)^{q-1}$, taking the tradeoff between them into account. Note that $q = \Theta(\log m / \log\log m)$.

Our algorithm is summarized as Algorithm 1. Note that, at line 6, the number of uncompleted tours may become less than $\lfloor m/q^k \rfloor$ because of multiple tours being completed at the same time. In such a case, we break ties arbitrarily to choose $\lfloor m/q^k \rfloor$ tours for phase k and carry the other (already completed) tours over into the next phase.

Algorithm 1 The proposed algorithm for the basic online$(\boldsymbol{M}, m)$-scheduling problem.

1 Let q be the integer such that $q^q > m \geq (q-1)^{q-1}$;
2 Set $k^* \leftarrow \lfloor \log_q m \rfloor + 1$;
3 Compute optimal m-TSP tours $C_1, \ldots, C_m$ over the sources;
4 **for** $k \leftarrow 1, 2, \ldots, k^*$ **do**
5 Assign available machines to the uncompleted tours so that the number of assigned machines becomes q^{k-1} for each tour;
6 Continue the process for each uncompleted tour until the number of uncompleted tours becomes $\lfloor m/q^k \rfloor$;

We prove that Algorithm 1 is $O(\log m / \log\log m)$-competitive for any metric $\boldsymbol{M}$, which immediately yields Theorem 6.

▶ **Theorem 9.** *For any metric $\boldsymbol{M}$, Algorithm 1 is $O(\log m / \log\log m)$-competitive for the basic online $(\boldsymbol{M}, m)$-scheduling problem.*

We prove this theorem in the following. We define ℓ_i to be the length of tour C_i (over the sources) and J_i to be the set of all jobs corresponding to C_i for each $i \in [m]$. Let $h_i^{\mathrm{sum}} = \sum_{j \in J_i}(p_j + d(b_j, a_j))$ and $h_i^{\max} = \max_{j \in J_i}(p_j + d(b_j, a_j))$. Note that if one machine processes jobs in J_i according to C_i, then it takes at most $\ell_i + h_i^{\mathrm{sum}}$ time.

We first provide lower bounds on the optimal offline makespan $\mathrm{OPT}(I)$.

▶ **Lemma 10.** $\mathrm{OPT}(I) \geq \max\{\max_{i\in[m]} \ell_i,\ \max_{i\in[m]} h_i^{\max},\ \sum_{i\in[m]} h_i^{\mathrm{sum}}/(2m)\}$.

Proof. As the optimal offline algorithm must reach all source states and $\max_{i\in[m]} \ell_i$ is the optimal value of m-TSP over the sources, we have $\mathrm{OPT}(I) \geq \max_{i\in[m]} \ell_i$.

Let j^* be a job such that $p_{j^*} + d(b_{j^*}, a_{j^*}) = \max_{i\in[m]} h_i^{\max}$ $(= \max_{j\in J}(p_j + d(b_j, a_j)))$. Since j^* must be processed at some time, we have

$$\mathrm{OPT}(I) \geq d(o, a_{j^*}) + p^* + d(b_{j^*}, o) \geq p^* + d(b_{j^*}, a_{j^*}) = \max_{i\in[m]} h_i^{\max},$$

where the second inequality holds by the triangle inequality.

Finally, the makespan is not less than the mean sum of processing times spent by each machine. By this and the assumption that $p \geq d(a, b)$ for each job, we obtain

$$\mathrm{OPT}(I) \geq \frac{1}{m}\sum_{j\in J} p_j \geq \sum_{j\in J} \frac{p_j + d(a_j, b_j)}{2m} = \sum_{i\in[m]} \frac{h_i^{\mathrm{sum}}}{2m}.$$ ◀

We analyze the completion time for each set of jobs J_i in terms of ℓ_i, $h_i^{\max}$, and h_i^{sum}.

▶ **Lemma 11.** *For a set of jobs J_i, let κ be the number of machines assigned for J_i throughout the algorithm. Then, the process for J_i is completed within $\ell_i + h_i^{\max} + \frac{h_i^{\mathrm{sum}}}{\kappa}$ from the time when the last machine was added.*

Proof. Throughout this proof, we only focus on the elapsed time after the last investment of machines for J_i. Let x and y be the machines that return to the origin o first and last, respectively. Suppose to the contrary that y returns to the origin o after time $\ell_i + h_i^{\max} + h_i^{\mathrm{sum}}/\kappa$. As the sum of the completion times of κ machines for J_i is at most $\kappa \cdot \ell_i + h_i^{\mathrm{sum}}$, x will complete the process on and before time $\ell_i + h_i^{\mathrm{sum}}/\kappa$. Thus, x returns to o at least $h_i^{\max}$ time earlier than y.

Consider the time when y starts processing the last job j^* in J_i. Here, y processes at least one job since otherwise y returns to the origin o by time ℓ_i, which is a contradiction. When working on j^*, the machine y must precede (or be at the same point as) the machine x, since otherwise y does not process j^*. Therefore, the time at which x return to the origin o is at most $p_{j^*} + d(b_{j^*}, a_{j^*})$ time earlier than y. However, this contradicts the assumption because $p_{j^*} + d(b_{j^*}, a_{j^*}) \leq h_i^{\max}$. ◀

Combining the above two lemmas, we can prove Theorem 9. For each phase $k \in [k^*]$, let $\tau_k \in \mathbb{R}_+$ be the time length of phase k, and let $S_k \subseteq [m]$ be the set of tours completed in phase k. Note that $|S_k| = \lfloor m/q^{k-1} \rfloor - \lfloor m/q^k \rfloor$ $(\forall k \in [k^*])$ and the makespan of the schedule obtained by the algorithm is $\sum_{k=1}^{k^*} \tau_k$. We will bound the value τ_k by using Lemma 11. Using the lemma in an intuitive way, we can obtain that $\tau_k \leq \max_{i\in S_k} \left(\ell_i + h_i^{\max} + \frac{h_i^{\mathrm{sum}}}{q^{k-1}}\right)$. However, this does not work well because it is far from $\sum_{i\in[m]} h_i^{\mathrm{sum}}/(2m)$, which is the only tool we have now to bound h_i^{sum} by $\mathrm{OPT}(I)$. Our main idea to overcome this issue is to use S_{k+1} instead of S_k, which allows us to evaluate τ_k as an average rather than a maximum. By combining this with the exponential increase in the number of machines in each phase, we can obtain the desired upper bound.

Proof of Theorem 9. It is sufficient to prove $\sum_{k=1}^{k^*} \tau_k = O(q) \cdot \mathrm{OPT}(I)$ because $O(q) = O(\log m / \log\log m)$.

We first bound τ_k for each phase $k \in [k^*-1]$. By Lemma 11, we have $\tau_k \le \ell_i + h_i^{\max} + \frac{h_i^{\mathrm{sum}}}{q^{k-1}}$ for any $i \in S_{k+1}$. Hence, by Lemma 10, we obtain

$$\begin{aligned}\tau_k &\le \min_{i \in S_{k+1}} \left(\ell_i + h_i^{\max} + \frac{h_i^{\mathrm{sum}}}{q^{k-1}}\right) \le \frac{1}{|S_{k+1}|} \sum_{i \in S_{k+1}} \left(\ell_i + h_i^{\max} + \frac{h_i^{\mathrm{sum}}}{q^{k-1}}\right) \\ &\le 2\mathrm{OPT}(I) + \sum_{i \in S_{k+1}} \frac{h_i^{\mathrm{sum}}}{q^{k-1} \cdot |S_{k+1}|} = 2\mathrm{OPT}(I) + \frac{2m}{q^{k-1} \cdot |S_{k+1}|} \cdot \sum_{i \in S_{k+1}} \frac{h_i^{\mathrm{sum}}}{2m}.\end{aligned}$$

As $|S_{k+1}| = \lfloor m/q^k \rfloor - \lfloor m/q^{k+1} \rfloor \ge \lfloor m/q^k \rfloor - \frac{1}{q}\lfloor m/q^k \rfloor = (1-1/q)\lfloor m/q^k \rfloor$, we get

$$\frac{2m}{q^{k-1} \cdot |S_{k+1}|} \le 2q \cdot \frac{m/q^k}{\lfloor m/q^k \rfloor - \frac{1}{q}\lfloor m/q^k \rfloor} = 2q \cdot \frac{1}{1-1/q} \cdot \frac{m/q^k}{\lfloor m/q^k \rfloor} \le 8q,$$

where the second inequality holds by $q \ge 2$ and $m/q^k \ge 1$. Thus, we obtain

$$\tau_k \le 2\mathrm{OPT}(I) + 8q \sum_{i \in S_{k+1}} \frac{h_i^{\mathrm{sum}}}{2m}. \tag{2}$$

Next, we bound τ_{k^*}. By Lemmas 10 and 11, we have

$$\begin{aligned}\tau_{k^*} &\le \max_{i \in S_{k^*}} \left(\ell_i + h_i^{\max} + \frac{h_i^{\mathrm{sum}}}{q^{k^*-1}}\right) \le 2\mathrm{OPT}(I) + \max_{i \in S_{k^*}} \frac{h_i^{\mathrm{sum}}}{q^{k^*-1}} \\ &< 2\mathrm{OPT}(I) + 2q \cdot \max_{i \in S_{k^*}} \frac{h_i^{\mathrm{sum}}}{2m} \le (2+2q)\mathrm{OPT}(I),\end{aligned} \tag{3}$$

where the third inequality holds by $m < q^{k^*}$.

Hence, by (2) and (3), we obtain

$$\begin{aligned}\sum_{k=1}^{k^*} \tau_k &\le 2(k^*-1) \cdot \mathrm{OPT}(I) + 8q \sum_{k=1}^{k^*-1} \sum_{i \in S_{k+1}} \frac{h_i^{\mathrm{sum}}}{2m} + (2+2q) \cdot \mathrm{OPT}(I) \\ &\le 4q \cdot \mathrm{OPT}(I) + 8q \cdot \sum_{i \in [m]} \frac{h_i^{\mathrm{sum}}}{2m} \le 4q \cdot \mathrm{OPT}(I) + 8q \cdot \mathrm{OPT}(I) = 12q \cdot \mathrm{OPT}(I).\end{aligned}$$

Therefore, Algorithm 1 is $O(\log m / \log\log m)$-competitive. ◀

Before concluding this subsection, we would like to mention that we can also construct a polynomial-time $O(\log m / \log\log m)$-competitive algorithm for the online $(\boldsymbol{M}, m)$-scheduling problem by using a constant approximation solution of m-TSP (which can be computed in polynomial-time [11]) instead of the optimal one in Algorithm 1.

4.2 Lower bound

We prove Theorem 7 that the competitive ratio $O(\log m / \log\log m)$ is best possible.

Proof of Theorem 7. We define a star-shaped metric $\boldsymbol{M} = (X, d)$ with $o = (0,0)$ as follows:

$$X = \big(\mathbb{N} \times (0,1]\big) \cup \{o\} \text{ and } d\big((i,x),(j,y)\big) = \begin{cases} |x-y| & \text{if } i = j, \\ x+y & \text{if } i \ne j. \end{cases}$$

Let us denote the end point $(i,1) \in X$ by σ_i.

Let q be the positive integer such that $q^q \le m < (q+1)^{q+1}$. For each $i \in [q^q]$, we prepare sufficiently many jobs (say m^2) that are either $(\sigma_i, \sigma_i, 0)$ or $(\sigma_i, \sigma_i, 1)$, depending on the actions taken by the online algorithm. We will refer to jobs $(\sigma_i, \sigma_i, 0)$ and $(\sigma_i, \sigma_i, 1)$ as empty and non-empty, respectively.

We fix an online algorithm. Then the adversary partitions the index set $[q^q]$ of end points into $q+1$ sets $S_1, S_2, \ldots, S_{q+1}$ based on the behavior of the algorithm. For each $k \in [q+1]$, every job with source σ_i ($i \in S_k$) will be set to be non-empty if the algorithm picks it (strictly) before time k, and empty otherwise. For each time t and $i \in [q^q]$, we denote by $\kappa_i(t)$ the number of machines such that the distance to σ_i is less than 1 at time t (i.e., machines at positions in $\{(i,x) \mid x \in (0,1]\}$). Define S_1 to be the set of indices $i \in [q^q]$ of end points with the $q^q - q^{q-1}$ largest values of $\kappa_i(1)$. Hence, $|S_1| = q^q - q^{q-1}$ and $\kappa_i(1) \ge \kappa_{i'}(1)$ for any $i \in S_1$ and $i' \notin S_1$. Similarly, for each $k = 2, 3, \ldots, q$, define S_k sequentially to be the set of $i \in [q^q] \setminus \bigcup_{k'=1}^{k-1} S_{k'}$ such that $\kappa_i(k)$ ($i \in S_k$) are the $q^{q+1-k} - p^{q-k}$ largest values among $\kappa_{i'}(k)$ ($i' \in [q^q] \setminus \bigcup_{k'=1}^{k} S_{k'}$). Finally, define S_{q+1} to be the set of remaining indices, i.e., $S_{q+1} = [q^q] \setminus \bigcup_{k'=1}^{q} S_{k'}$. We remark that S_k's are disjoint, and S_{q+1} is a singleton because $\sum_{k'=1}^{q} |S_{k'}| = \sum_{k'=1}^{q} (q^{q+1-k'} - q^{q-k'}) = q^q - 1$.

As some jobs with source σ_i ($i \in S_k$) must be processed at time $q+1$ or later, the makespan of the schedule obtained by the online algorithm is at least $q+2$.

For each $k \in [q]$, let J_k be the set of non-empty jobs that were started to be processed in the interval $[k, k+1)$. We observe the cardinality of J_k. Recall that it takes one unit time to process a non-empty job. Hence, each machine can process at most one job in J_k. In addition, in order for a machine to start processing a job with source σ_i in the interval, the distance from σ_i from its state at time k must be less than 1. Since the sources of jobs in J_k are located at σ_i for some $i \in \bigcup_{k'=k+1}^{q+1} S_{k'}$, we obtain that

$$|J_k| \le \sum_{i \in \bigcup_{k'=k+1}^{q+1} S_{k'}} \kappa_i(k) \le m \cdot \frac{\left|\bigcup_{k'=k+1}^{q+1} S_{k'}\right|}{\left|\bigcup_{k'=k}^{q+1} S_{k'}\right|} = m \cdot \frac{q^{q-k}}{q^{q+1-k}} \cdot m = \frac{m}{q}.$$

As the algorithm starts to process any non-empty job at a time in $[1, q+1) = \bigcup_{k=1}^{q} [k, k+1)$, the total number of non-empty jobs is $\sum_{k \in [q]} |J_k| \le (m/q) \cdot q = m$. Hence, the optimal offline makespan is at most the sum of 2 time units to process the empty jobs and 3 time units to process the non-empty jobs, which equals 5. Therefore, the competitive ratio is at least $(q+2)/5 = \Omega(\log m / \log \log m)$. ◀

5 Special Cases

In this section, we focus on the following three special cases: (i) single machine case ($m = 1$), (ii) two machines case ($m = 2$), and (iii) the optimal values of the 1-TSP and the m-TSP are close. In particular, for cases (ii) and (iii), we provide algorithms that improves Algorithm 1.

5.1 Single machine case

In this subsection, we consider the case where there is only one machine, i.e., $m = 1$. We first show that Algorithm 1 with $m = 1$ is 3-competitive.

▶ **Theorem 12.** *For any metric $\boldsymbol{M}$, Algorithm 1 is 3-competitive for the basic online $(\boldsymbol{M}, 1)$-scheduling problem.*

We remark that the competitive ratio 3 is tight for Algorithm 1. To see this, let us consider an instance with $\boldsymbol{M} = \mathbb{R}$ and three jobs, where job 1 is $(1, 0, 1)$, job 2 is $(1, 1, 0)$, and job 3 is $(0, 1, 1)$. In Algorithm 1, the machine processes the jobs in the order $1, 2, 3$, and the makespan is 6. On the other hand, the optimal offline makespan is 2 by processing jobs in the order $3, 2, 2$. Hence, the competitive ratio of Algorithm 1 is at least 3 when $m = 1$.

The proof of Theorem 12 implies that the makespan of the schedule obtained by Algorithm 1 is at most $2p(I) + \mathrm{LB}(I)$ for any instance I, i.e., $(\alpha, \beta, \gamma) = (0, 2, 1)$ for the values in (1). By Theorems 1, 3, and 5, the competitive ratios of IGNORE, REPLAN, and SMARTSTART incorporating with Algorithm 1 are 4.5, 5, and 4 for the online $(\boldsymbol{M}, 1)$-scheduling problem, respectively. We remark that our SMARTSTART is the same algorithm as the BOUNCER algorithm proposed by Lipmann et al. [19] (if it uses SMARTSTART as a 2-competitive algorithm for the online TSP over the sources).

▶ **Theorem 13.** *For any metric $\boldsymbol{M}$, there exists a 4-competitive algorithm for the online $(\boldsymbol{M}, 1)$-scheduling problem.*

Next, we provide a lower bound of the competitive ratio for the basic online $(\boldsymbol{M}, 1)$-scheduling problem.

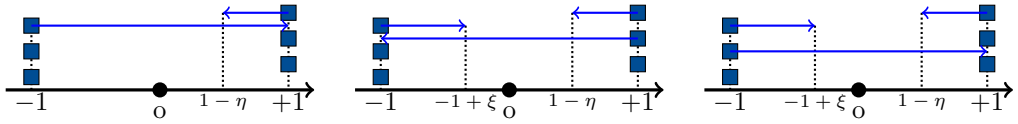
▶ **Theorem 14.** *There exists no 2.255-competitive algorithm even for the basic online $([-1, +1], 1)$-scheduling problem.*

Proof. Fixing an algorithm, we construct an adversary that gives the lower bound. Let η and ξ be the solutions of the following equations:

$$(8 + 2\eta)/4 = (10 + 2\xi)/(4 + 2\eta) = 12/(4 + 2\xi).$$

Note that $0.5101 < \eta < 0.5102$ and $0.6606 < \xi < 0.6607$.

Suppose that there are 6 jobs in total, three of which have their source state in $+1$, and the other three have their source state in -1. Without loss of generality, the algorithm first serves a job in $+1$. Then, the adversary sets the first job to be $(1, 1 - \eta, \eta)$. We prove the theorem by cases according to the behavior of the algorithm (see Figure 2).



(a) Case 1. **(b)** Case 2.1. **(c)** Case 2.2.

Figure 2 Adversarial instance for the basic online $([-1, +1], 1)$-scheduling problem.

Case 1. Suppose that the algorithm next serves another job in $+1$. Then the adversary sets the remaining jobs in $+1$ to be $(1, 1, 0), (1, 1, 0)$. After jobs in $+1$, the machine moves to -1, and the adversary sets the first job in -1 chosen by the algorithm to be $(-1, 1, 2)$, and the remaining jobs to be $(-1, -1, 0), (-1, -1, 0)$. Then the algorithm takes time at least $1 + \eta \times 2 + 2 + 2 + 2 + 1 = 8 + 2\eta$, while the optimal offline makespan is 4. Hence, the competitive ratio is at least $(8 + 2\eta)/4 > 2.255$.

Case 2. Suppose that the algorithm next serves a job in -1. Then, the adversary sets the job to be $(-1, -1 + \xi)$. We divide cases according to the third job which the machine processes.

Case 2.1. If the third job is in -1, then the remaining jobs are set to be $(-1,-1,0)$, $(-1,-1,0)$, $(1,-1,2)$, $(1,1,0)$, and the algorithm is made to choose the jobs in this order. The algorithm takes time at least $10+2\xi$ while the optimal offline makespan is $4+2\eta$. Hence, the competitive ratio is at least $(10+2\xi)/(4+2\eta) > 2.255$.

Case 2.2. If the third job is in $+1$, then the remaining jobs are set to be $(1,1,0)$, $(1,1,0)$, $(-1,1,2)$, $(-1,-1,0)$, and the algorithm is made to choose the jobs in this order. The algorithm takes time at least 12 while the optimal offline makespan is $4+2\xi$. Hence, the competitive ratio is at least $12/(4+2\xi) > 2.255$.

Therefore, the competitive ratio is at least 2.255 in any cases. ◀

Combining the instance in the proof of Theorem 14 with the idea of setting jobs used in [19, Theorem 4], we can obtain a lower bound of 3.181 for the general case. This result also improves the best known lower bound of $1+\frac{3\sqrt{2}}{2} \approx 3.12$ for the online dial-a-ride problem under the incomplete information model [19].

▶ **Theorem 15.** *For some metric $\boldsymbol{M}$, there exists no* 3.181*-competitive algorithm for the online $(\boldsymbol{M},1)$-scheduling problem.*

Proof. Let $\eta = 0.362\ldots$, $\xi = 0.514\ldots$, and $\rho = 3.181\ldots$ be the unique numbers satisfying

$$\rho = \frac{12+2\cdot\eta}{4} = \frac{14+2\cdot\xi}{4+2\cdot\eta} = \frac{16}{4+2\cdot\xi}. \tag{4}$$

Let $N \in \mathbb{N}$ be an integer bigger than $1.25/(\rho-3.181)$, and let $\boldsymbol{M}$ be the star graph with $2\cdot N$ leaves, each of which is at unit distance from the origin o. Consider an online algorithm ALG. We construct an instance for which ALG has makespan 3.181 times longer than the optimal (offline) schedule. Initially, there are three jobs (whose sources are) in each leaf. Each time ALG processes a job before time $4\cdot N-4$, the adversary sets the job to be empty, and adds a job with the same source and the release time being one unit amount of time later. Thus, at most $4\cdot N-4$ jobs are added by time $4\cdot N-4$, and there are exactly three unprocessed jobs (including unreleased jobs) in each leaf at time $4\cdot N-4$. We call them the *decisive* jobs (as in [19]), and specify their processing information below.

Because it takes two units of time to travel between leaves, at least two of the $2\cdot N$ leaves, say 0^- and 0^+, are left unvisited by ALG during the time interval $[0,\, 4\cdot N-4)$. Likewise, among the $2\cdot N-2$ remaining leaves, there are two, say 1^- and 1^+, that are unvisited during $[4,\, 4\cdot N-4)$. Continuing in this way, we can name the leaves $0^\pm$, $1^\pm$, …, $(N-1)^\pm$ so that for each $j \in \{0,\ldots,N-1\}$, neither j^- nor j^+ is visited by ALG during $[4\cdot j,\, 4\cdot N-4)$. We identify the path $M_j \subseteq \boldsymbol{M}$ between j^- and j^+ (through o) with the interval $[-1,\,+1]$, and determine the processing information of the six decisive jobs in $j^\pm$ by the order in which ALG processes them, in the way described in the cases 1, 2.1, and 2.2 in the proof of Theorem 14 (but now with the new η and ξ defined in (4)).

Note that some jobs may be unreleased at time $4N-3$. This affects Case 1 in the proof of Theorem 14, in which the machine processes three jobs in the same leaf consecutively. However, since the machine skips unreleased jobs and has to come back later, the completion time gets longer. Thus, we may assume that three jobs are released on each leaf.

By the analysis there, the amount of time after $4\cdot N-5$ spent by ALG in M_j is at least $8+2\cdot\eta$, $10+2\cdot\xi$, and 12 in the three cases, respectively, whereas the best schedule for the basic $(M_j,1)$-scheduling instance given by the six decisive jobs in $j^\pm$ (forgetting their release times) has makespan 4, $4+2\cdot\eta$, and $4+2\cdot\xi$, respectively. Note that simply concatenating these best schedules for $j=0,\ldots,N-1$ gives an offline schedule for our whole instance,

since all jobs in $j^{\pm}$ appear by time $4 \cdot j + 1$. Thus, writing N_1, $N_{2.1}$, and $N_{2.2}$ for the number of $j \in \{0, \dots, N-1\}$ for which the three cases happen ($N_1 + N_{2.1} + N_{2.2} = N$), we can bound from below the ratio of ALG's makespan to the optimal by

$$\begin{aligned}
&\frac{(4 \cdot N - 5) + N_1 \cdot (8 + 2 \cdot \eta) + N_{2.1} \cdot (10 + 2 \cdot \xi) + N_{2.2} \cdot \quad 12}{N_1 \cdot \quad 4 \quad + N_{2.1} \cdot (4 + 2 \cdot \eta) + N_{2.2} \cdot (4 + 2 \cdot \xi)} \\
=&\frac{N_1 \cdot (12 + 2 \cdot \eta) + N_{2.1} \cdot (14 + 2 \cdot \xi) + N_{2.2} \cdot \quad 16 \quad - 5}{N_1 \cdot \quad 4 \quad + N_{2.1} \cdot (4 + 2 \cdot \eta) + N_{2.2} \cdot (4 + 2 \cdot \xi)} \\
\geq &\min\left\{ \frac{12 + 2 \cdot \eta}{4}, \frac{14 + 2 \cdot \xi}{4 + 2 \cdot \eta}, \frac{16}{4 + 2 \cdot \xi} \right\} - \frac{5}{N \cdot 4} \geq \rho - \frac{1.25}{N} > 3.181.
\end{aligned}$$

◀

5.2 Two machines case

In this subsection, we focus on the two-machine case, i.e., $m = 2$. As shown in Example 22 in Appendix, the competitive ratio of Algorithm 1 is at least 4 even for the basic online $(\mathbb{R}, 2)$-scheduling problem. In fact, we can improve Algorithm 1 as follows. The main idea is to move the machines in the opposite directions in phase 2. Formally, our algorithm can be stated as follows. The algorithm first computes optimal 2-TSP tours C_1 and C_2 over the sources. Then, machine i travels along C_i in a direction $(i = 1, 2)$. When a machine arrives at the source of an unprocessed job, then it processes the job, returns to its source, and continues the travel. If C_1 is completed first, then machine 1 now travels along C_2 in the reverse direction. Similarly for the case when C_2 is completed. When all jobs have been processed, each machine directly returns to the origin as soon as possible.

▶ **Theorem 16.** *For any metric $\boldsymbol{M}$, there exists a 3.5-competitive algorithm for the basic online $(\boldsymbol{M}, 2)$-scheduling problem.*

We can also prove that the makespan of the above schedule is at most $\frac{1}{2}\mathrm{OPT}(I) + 2 \cdot p(I)/2 + \mathrm{LB}(I)$, i.e., $(\alpha, \beta, \gamma) = (1/2, 2, 1)$ for the values in (1). Thus, the competitive ratio of SMARTSTART incorporating with the above algorithm is $4 + \frac{\sqrt{3}}{2} \approx 4.866$ by Theorem 5.

▶ **Theorem 17.** *For any metric $\boldsymbol{M}$, there exists a $(4 + \frac{\sqrt{3}}{2})$-competitive algorithm for the online $(\boldsymbol{M}, 2)$-scheduling problem.*

5.3 Special metrics

Finally, we improve Algorithm 1 for the case where the optimal values of 1-TSP and m-TSP are close. A typical example of such a situation is instances with the half-line metric: the optimal value of 1-TSP coincides with that of m-TSP for any m because the values are twice the distance between origin and the rightmost source. The main idea of the improvement is to use the optimal 1-TSP tour instead of the m-TSP. This reduces the completion time because the number of phases is reduced to one. We further reduce the completion time by moving the machines in both directions of the optimal 1-TSP tour.

To be more precise, our algorithm first computes an optimal 1-TSP tour C over the sources. After that, half the machines (i.e., $\lceil m/2 \rceil$ machines) travel along the tour in a direction, and the other half (i.e., $\lfloor m/2 \rfloor$ machines) travel along the tour in the opposite direction. When a machine arrives at the source of an unprocessed job, then it processes the job, returns to its source, and continues the travel. When all jobs have been processed (including processing), each machine directly returns to the origin as soon as possible.

▶ **Theorem 18.** *Fix the number of machines $m \geq 2$ and the metric $\boldsymbol{M}$. Suppose that the ratio between the optimal values of 1-TSP and m-TSP is at most μ for any instance on $\boldsymbol{M}$. There exists a $(\frac{\lceil m/2 \rceil}{m} \cdot \mu + 3)$-competitive algorithm for the online $(\boldsymbol{M}, m)$-scheduling problem.*

Note that $\frac{\lceil m/2 \rceil}{m} \leq 2/3$ for any $m \geq 2$. Additionally, recall that, when $m = 1$, there is a 3-competitive algorithm for any metric (Theorem 13). As the value μ can be taken as 1 and 2 for $\mathbb{R}_+$ and $\mathbb{R}$, respectively, we can obtain the following corollaries.

▶ **Corollary 19.** *For any m, there exists a $11/3$-competitive algorithm for the basic online $(\mathbb{R}_+, m)$-scheduling problem.*

▶ **Corollary 20.** *For any m, there exists a $13/3$-competitive algorithm for the basic online $(\mathbb{R}, m)$-scheduling problem.*

6 Other Settings

In this section, we discuss other variants of online $(\boldsymbol{M}, m)$-scheduling problems.

We first consider minimizing the total completion time, i.e., the sum of completion times of all jobs. We observe that if the objective is to minimize the total completion time, any online algorithm is not competitive even for the basic online $(\mathbb{R}^0, 1)$-scheduling problem. Fix an online algorithm. Let n be a positive integer and consider an instance with n jobs with source and destination being the origin o. Suppose that the algorithm first processes job j^*. We set $(a_j, b_j, p_j) = (o, o, 0)$ for all $j \neq j^*$ and $(a_{j^*}, b_{j^*}, p_{j^*}) = (o, o, 1)$. Then, the total completion time of the algorithm is at least n as the completion time of every job is at least 1, but the optimal total completion time is 1 by processing job j^* last. As n can be taken as an arbitrarily large number, any algorithm is not competitive.

For the case where the processing time is known (but the destination state is not), we can design a constant competitive algorithm. In fact, we can obtain a 3-competitive algorithm for the basic case by using a solution of m-TSP in which the processing time is taken into account. On the other hand, if the destination state is known (but the processing time is not), the competitive ratio is $\Theta(\log m / \log \log m)$ since the lower bound shown in Theorem 7 also holds in this setting.

Next, we observe the case where the machines do not need to return to the origin, i.e., the objective is to minimize the time until all jobs have been completed. Such a setting is called *open* or *nomadic*, whereas the setting of our problem is said to be *closed* or *homing*. It is not difficult to see that the optimal makespan for the open setting is not less than half of the optimal makespan for the closed setting. Hence, if there exists a ρ-competitive algorithm for the closed setting, then there exists a 2ρ-competitive algorithm for the open setting. By combining this with 8, we obtain an $O(\log m / \log \log m)$-competitive algorithm for the open version of the online $(\boldsymbol{M}, m)$-scheduling problem. For the open version of the basic online $(\boldsymbol{M}, 1)$-scheduling problem, we can obtain a 3-competitive algorithm by the same way as Algorithm 1 (but use the path TSP). In addition, for the open version of the online $(\boldsymbol{M}, 1)$-scheduling problem, we can obtain a 6-competitive algorithm by applying REPLAN.

Finally, we discuss the preemptive version, i.e., the machines are allowed to preempt jobs in any point and resume the job later. We can observe that the competitive ratio of the basic online $(\boldsymbol{M}, m)$-scheduling problem is lower bounded by $\Omega(\log m / \log \log m)$ even when the source and the destination states are the same for all jobs. To see this, we consider an adversary similar to the one shown in Theorem 7. Consider the same metric, sources and the destinations of the jobs, and partition of the end points $S_1, S_2, \ldots, S_{q+1}$. However, we set the processing time of each job j with source in S_k to be $\min\{t + \epsilon, 1\}$ for each $k \in [q + 1]$,

where t is the total processing length of that job j has been processed by time k and $\epsilon > 0$. Then, the makespan of the schedule obtained by the online algorithm is at least $q + 2$ while the optimal offline makespan is at most 5. Therefore, by setting $\epsilon \to 0$, the competitive ratio is at least $(q + 2)/5 = \Omega(\log m / \log \log m)$.

References

1 Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. Online Dial-a-Ride Problems: Minimizing the Completion Time. In *Proceedings of STACS*, volume 1770, pages 639–650, 2000.

2 G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the On-Line Travelling Salesman. *Algorithmica*, 29(4):560–581, 2001.

3 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Competitive algorithms for the on-line traveling salesman. In *Proceedings of the 4th Workshop on Algorithms and Data Structures*, pages 206–217, 1995.

4 Alexander Birx. *Competitive analysis of the online dial-a-ride problem*. PhD thesis, Technische Universität Darmstadt, 2020.

5 Alexander Birx and Yann Disser. Tight analysis of the smartstart algorithm for online dial-a-ride on the line. *SIAM Journal on Discrete Mathematics*, 34(2):1409–1443, 2020.

6 Alexander Birx, Yann Disser, and Kevin Schewior. Improved bounds for open online dial-a-ride on the line. In *Proceedings of Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 145, 2019.

7 Antje Bjelde, Jan Hackfeld, Yann Disser, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Miriam Schlöter, Kevin Schewior, and Leen Stougie. Tight Bounds for Online TSP on the Line. *ACM Transactions on Algorithms*, 17(1):1–58, 2021.

8 Michiel Blom, Sven O Krumke, Willem E de Paepe, and Leen Stougie. The online tsp against fair adversaries. *INFORMS Journal on Computing*, 13(2):138–148, 2001.

9 Vincenzo Bonifaci, Maarten Lipmann, and Leen Stougie. *Online multi-server dial-a-ride problems*. SPOR-Report : reports in statistics, probability and operations research. Technische Universiteit Eindhoven, 2006.

10 Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.

11 Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation Algorithms for Some Routing Problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

12 Giorgio Gambosi and Gaia Nicosia. On-line scheduling with setup costs. *Information Processing Letters*, 73(1–2):61–68, 2000.

13 R. L. Graham. Bounds for Certain Multiprocessing Anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

14 Dan Gusfield. Bounds for naive multiple machine scheduling with release times and deadlines. *Journal of Algorithms*, 5(1):1–6, 1984.

15 Leslie A. Hall and David B. Shmoys. Approximation schemes for constrained scheduling problems. In *Proceedings of Annual Symposium on Foundations of Computer Science*, pages 134–139, 1989.

16 Hongtao Hu, K. K.H. Ng, and Yichen Qin. Robust Parallel Machine Scheduling Problem with Uncertainties and Sequence-Dependent Setup Time. *Scientific Programming*, 2016, 2016.

17 Patrick Jaillet and Michael R. Wagner. Generalized online routing: New competitive ratios, resource augmentation, and asymptotic analyses. *Operations Research*, 56(3):745–757, 2008.

18 Sven O. Krumke. *Online optimization: Competitive analysis and beyond*. PhD thesis, Technische Universität Berlin, 2001.

19 Maarten Lipmann, Xiwen Lu, Willem E. de Paepe, Rene A. Sitters, and Leen Stougie. On-Line Dial-a-Ride Problems Under a Restricted Information Model. *Algorithmica*, 40(4):319–329, 2004.

20 David M Miller, Hui-Chuan Chen, Jessica Matson, and Qiang Liu. A hybrid genetic algorithm for the single machine scheduling problem. *Journal of Heuristics*, 5(4):437–454, 1999.
21 David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313–1331, 1995.

A Omitted proofs and examples

▶ **Theorem 5.** *SMARTSTART is* $\frac{6\alpha+4\beta+4\gamma+1+\sqrt{(2\alpha+1)^2+8\gamma}}{4}$*-competitive for the online* $(\boldsymbol{M}, m)$*-scheduling problem by setting* $\theta = \frac{2\alpha+1+\sqrt{(2\alpha+1)^2+8\gamma}}{4}$.

Proof. We note that θ satisfies the equality $(2\alpha+1)\theta+\gamma = 2\theta^2$. We consider a non-empty instance I. Let S be the set of jobs processed in the last subschedule of the SMARTSTART algorithm, and let t_S be the time when the execution of the subschedule started.

Suppose that the machines are idle just before time t_S. In this case, $t_S = \theta \cdot \mathrm{LB}(S_0)$ and we have

$$\begin{aligned}\mathrm{SMARTSTART}(I) &= t_S + \mathrm{ALG}_0(S_0)\\ &\le \theta \cdot \mathrm{LB}(S_0) + \alpha\mathrm{OPT}(S_0) + \beta \cdot p(S_0)/m + \gamma\mathrm{LB}(S_0)\\ &\le (\theta+\alpha+\beta+\gamma)\mathrm{OPT}(I)\\ &= \frac{6\alpha+4\beta+4\gamma+1+\sqrt{(2\alpha+1)^2+8\gamma}}{4} \cdot \mathrm{OPT}(I).\end{aligned}$$

Next, suppose that the last subschedule is started immediately after the second last one. Let R be the set of jobs processed in the second last subschedule and let t_R be the time when the second last subschedule is started. Define $\lambda = \frac{1}{2} + \frac{\gamma}{2\theta}$ $(= \theta - \alpha)$. Then, we have

$$\begin{aligned}&\mathrm{SMARTSTART}(I)\\ &= t_R + \mathrm{ALG}_0(R_0) + \mathrm{ALG}_0(S_0)\\ &\le t_R + (\alpha\mathrm{OPT}(R_0) + \beta \cdot p(R_0)/m + \gamma\mathrm{LB}(R_0)) + (\alpha+\gamma)\mathrm{OPT}(S_0) + \beta p(S_0)/m\\ &\le (1+\gamma/\theta)t_R + (\alpha+\beta)\mathrm{OPT}(I) + (\alpha+\gamma-\lambda)\mathrm{OPT}(S_0) + \lambda\mathrm{OPT}(S_0)\\ &\le (1+\gamma/\theta)t_R + (2\alpha+\beta+\gamma-\lambda)\mathrm{OPT}(I) + \lambda\mathrm{OPT}(S_0)\\ &= 2\lambda \cdot t_R + (2\alpha+\beta+\gamma-\lambda)\mathrm{OPT}(I) + \lambda\mathrm{OPT}(S_0),\end{aligned}$$

where the second inequality holds by $t_R \ge \theta \cdot \mathrm{LB}(R_0)$ and $p(R_0)/m + p(S_0)/m \le \mathrm{OPT}(I)$, and the third inequality holds by $\alpha+\gamma \ge \lambda$. Let $f \in \arg\max_{j\in S} d(o, a_j)$. As $\mathrm{OPT}(S_0) \le \mathrm{OPT}(I) - t_R + d(o, a_f)$, we have,

$$\begin{aligned}\mathrm{SMARTSTART}(I) &\le 2\lambda \cdot t_R + (2\alpha+\beta+\gamma-\lambda)\mathrm{OPT}(I) + \lambda(\mathrm{OPT}(I) - t_R + d(o, a_f))\\ &= (2\alpha+\beta+\gamma)\mathrm{OPT}(I) + \lambda(t_R + d(o, a_f))\\ &\le (2\alpha+\beta+\gamma+\lambda)\mathrm{OPT}(I) = (\alpha+\beta+\gamma+\theta)\mathrm{OPT}(I)\\ &= \frac{6\alpha+4\beta+4\gamma+1+\sqrt{(2\alpha+1)^2+8\gamma}}{4} \cdot \mathrm{OPT}(I).\end{aligned}$$

◀

▶ **Example 21.** To observe this, consider an instance with m^2 jobs, where the source, destination, and processing time of job j are respectively

$$a_j = b_j = \left(\cos\tfrac{2\pi(j-1)}{m}, \sin\tfrac{2\pi(j-1)}{m}\right) \quad \text{and} \quad p_j = \begin{cases} 100 & \text{if } j \equiv 1 \pmod m,\\ 0 & \text{otherwise.}\end{cases}$$

Note that only jobs $1, m+1, \ldots, m(m-1)+1$ are non-empty. Then, the makespan of the schedule obtained by the simple method is $100m+2$. Indeed, each machine processes m jobs $i, m+i, \ldots, m(m-1)+i$ for some i. The completion time of the machine processing jobs $1, \ldots, m(m-1)+1$ is $100m+2$, while that of others is 2. On the other hand, the optimal schedule is that each machine processes each of the m non-empty jobs, and then machine i processes jobs $i, m+i, \ldots, m(m-1)+i$ for $i = 2, \ldots, m-1$ (see also Figure 3). The makespan of this schedule is at most 104, and hence the competitive ratio is $(100m+2)/104 = \Omega(m)$.

(a) Simple method. **(b)** Optimal.

Figure 3 An instance that the simple method does not work well ($m = 5$).

▶ **Theorem 12.** *For any metric $\boldsymbol{M}$, Algorithm 1 is 3-competitive for the basic online $(\boldsymbol{M}, 1)$-scheduling problem.*

Proof. When $m = 1$, Algorithm 1 just processes the jobs along with an optimal 1-TSP tour over the sources (there is only one phase). Let $h^{\mathrm{sum}} = \sum_{j \in J}(p_j + d(b_j, a_j))$ and let ℓ_1 be the length of the optimal 1-TSP tour. By Lemma 10, we observe that $h^{\mathrm{sum}} \leq 2p(I) \leq 2\mathrm{OPT}(I)$ and $\ell_1 = \mathrm{LB}(I) \leq \mathrm{OPT}(I)$. Thus, the makespan of the schedule obtained by the algorithm is at most $\ell_1 + h^{\mathrm{sum}} \leq \mathrm{LB}(I) + 2p(I) \leq 3\mathrm{OPT}(I)$. Therefore, Algorithm 1 is 3-competitive when $m = 1$. ◀

▶ **Example 22.** To observe this, let us consider an instance with 6 jobs, where job 1 is $(0, -1, 1)$, job 2 is $(1, 0, 1)$, job 3 is $(1, 1, 0)$, job 4 is $(0, 1, 1)$, job 5 is $(-1, 0, 1)$, and job 6 is $(-1, -1, 0)$. It is not difficult to see that an optimal 2-TSP tours over the sources are $C_1 = (1, 2, 3, 4)$ and $C_2 = (5, 6)$. In Algorithm 1, machines 1 and 2 processes jobs $\{1, 2, 3, 4\}$ and $\{5, 6\}$ in these orders (see Figure 4). Hence, the makespan of the schedule obtained by the algorithm is 8. On the other hand, the makespan is 2 if machines 1 and 2 processes jobs $\{4, 3, 2\}$ and $\{1, 6, 5\}$, respectively, in these orders. Hence, the competitive ratio of Algorithm 1 is at least 4 when $m = 2$.

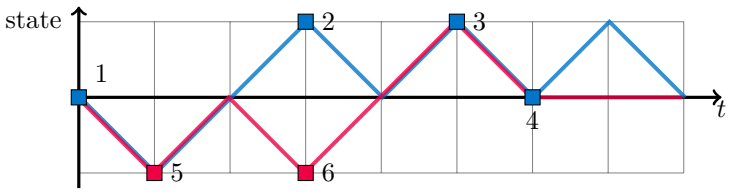


Figure 4 The schedule of Algorithm 1.

▶ **Theorem 16.** *For any metric $\boldsymbol{M}$, there exists a 3.5-competitive algorithm for the basic online $(\boldsymbol{M}, 2)$-scheduling problem.*

Proof. We analyze the above algorithm. Define ℓ_1 and ℓ_2 to be the lengths of the tours C_1 and C_2, respectively, and let $h^{\mathrm{sum}} = \sum_{j\in J}\big(p_j + d(b_j, a_j)\big)$. By Lemma 10, $(\ell_1+\ell_2)/2 \le \max\{\ell_1, \ell_2\} \le \mathrm{OPT}(I)$ and $h^{\mathrm{sum}} \le 2\sum_{j\in J} p_j \le 4\mathrm{OPT}(I)$. Let τ^* be the latest time to start processing a job, and let i be the machine that returns to the origin later (breaking ties arbitrarily). Let job j be the last job processed by machine i. Note that job j may be started to be processed earlier than τ^*. We may assume without loss of generality that such a job exists, because otherwise i processes no jobs and the makespan is at most $\max\{\ell_1, \ell_2\} \le \mathrm{OPT}(I)$. Note that $\tau^* \le \frac{1}{2}\big(\ell_1 + \ell_2 + h^{\mathrm{sum}}\big) \le 3\mathrm{OPT}(I)$.

Suppose that i is processing (or starts processing) job j at time τ^*. Then, the time when job j is started to process is at the latest $\frac{1}{2}\left(\ell_1 + \ell_2 + \sum_{j'\in J\setminus\{j\}}\big(p_{j'} + d(b_{j'}, a_{j'})\big)\right)$. Hence, the makespan is at most

$$\begin{aligned}
&\frac{1}{2}\left(\ell_1 + \ell_2 + \sum_{j'\in J\setminus\{j\}}\big(p_{j'} + d(b_{j'}, a_{j'})\big)\right) + p_j + d(b_j, o)\\
&\quad\le \frac{1}{2}\big(\ell_1 + \ell_2 + h^{\mathrm{sum}} - p_j - d(b_j, a_j)\big) + p_j + \frac{d(o, a_j) + d(a_j, b_j) + d(b_j, o)}{2}\\
&\quad\le \frac{1}{2}\big(\ell_1 + \ell_2 + h^{\mathrm{sum}} + d(o, a_j) + p_j + d(b_j, o)\big) \le \frac{7}{2}\mathrm{OPT}(I),
\end{aligned}$$

where the last inequality holds by $d(o, a_j) + p_j + d(b_j, o) \le \mathrm{OPT}(I)$.

Next, suppose that i is returning to a tour after processing job j at time τ^*. Let u be the state of i at time τ^*. As u is on the way from b_j to a_j, we have

$$\begin{aligned}
d(u, o) &\le \min\{d(u, a_j) + d(a_j, o),\, d(u, b_j) + d(b_j, o)\}\\
&\le \frac{1}{2}\Big(\big(d(u, a_j) + d(a_j, o)\big) + \big(d(u, b_j) + d(b_j, o)\big)\Big)\\
&= \frac{1}{2}\big(d(o, a_j) + d(a_j, b_j) + d(b_j, o)\big) \le \frac{1}{2}\mathrm{OPT}(I).
\end{aligned}$$

Hence, the makespan is at most $\tau^* + d(u, o) \le \frac{7}{2}\mathrm{OPT}(I)$.

Finally, suppose that i is traveling a tour without processing a job at time τ^*. Let u be the state of i at time τ^*. Then, $d(u, o) \le \frac{1}{2}\max\{\ell_1, \ell_2\} \le \frac{1}{2}\mathrm{OPT}(I)$. Hence, the makespan is at most $\tau^* + d(u, o) \le \frac{7}{2}\mathrm{OPT}(I)$.

Therefore, the competitive ratio of the above algorithm is at most $7/2 = 3.5$. ◀

▶ **Theorem 18.** *Fix the number of machines $m \ge 2$ and the metric $\boldsymbol{M}$. Suppose that the ratio between the optimal values of 1-TSP and m-TSP is at most μ for any instance on $\boldsymbol{M}$. There exists a $(\frac{\lceil m/2\rceil}{m}\cdot\mu + 3)$-competitive algorithm for the online $(\boldsymbol{M}, m)$-scheduling problem.*

Proof. We analyze the above algorithm. Let ℓ_1^* and ℓ_m^* be the optimal length of the 1-TSP and m-TSP over the sources, respectively. In addition, let $h^{\mathrm{sum}} = \sum_{j\in J}(p_j + d(b_j, a_j))$, and let τ^* be the time when the last job started to be processed. Then, τ^* is at most

$$\frac{1}{m}(\lceil m/2\rceil\cdot\ell_1^* + h^{\mathrm{sum}}) \le \frac{\lceil m/2\rceil}{m}\frac{\ell_1^*}{\ell_m^*}\mathrm{OPT}(I) + \mathrm{OPT}(I) \le \left(\frac{\lceil m/2\rceil}{m}\cdot\mu + 2\right)\mathrm{OPT}(I)$$

because $\ell_m^* \le \mathrm{OPT}(I)$ and $h^{\mathrm{sum}}/(2m) \le \mathrm{OPT}(I)$ by Lemma 10. Next, consider the machine i^* that is the last to return to the origin. If i^* is either in the middle of processing job j or returning to its source after processing job j at time (just after) τ^*, the makespan is at most

$\tau^* + p_j + d(b_j, o) \leq \left(\frac{\lceil m/2 \rceil}{m} \cdot \mu + 3\right) \mathrm{OPT}(I)$ because $p_j + d(b_j, o) \leq d(o, a_j) + p_j + d(b_j, o) \leq \mathrm{OPT}(I)$. Otherwise, suppose that the state of i^* at time τ^* is q, which is located on the way from a_j to $a_{j'}$. Then, we have

$$\begin{aligned} d(p, o) &\leq \min\{d(q, a_j) + d(a_j, o),\ d(q, a_{j'}) + d(a_{j'}, o)\} \\ &\leq \frac{1}{2}(d(a_j, o) + d(a_j, a_{j'}) + d(a_{j'}, o)) \leq d(a_j, o) + d(a_{j'}, o) \leq \mathrm{OPT}(I). \end{aligned}$$

Thus, the makespan is at most $\tau^* + \mathrm{OPT}(I) \leq \left(\frac{\lceil m/2 \rceil}{m} \cdot \mu + 3\right) \mathrm{OPT}(I)$. ◀