Improved Algorithms for Online Rent Minimization Problem Under Unit-Size Jobs

Enze Sun
The University of Hong Kong, Hong Kong, China

Zonghan Yang
Shanghai Jiao Tong University, China

Yuhao Zhang
Shanghai Jiao Tong University, China

Abstract

We consider the Online Rent Minimization problem, where online jobs with release times, deadlines, and processing times must be scheduled on machines that can be rented for a fixed length period of T. The objective is to minimize the number of machine rents. This problem generalizes the Online Machine Minimization problem where machines can be rented for an infinite period, and both problems have an asymptotically optimal competitive ratio of $O(\log(p_{\max}/p_{\min}))$ for general processing times, where $p_{\max}$ and $p_{\min}$ are the maximum and minimum processing times respectively. However, for small values of $p_{\max}/p_{\min}$, a better competitive ratio can be achieved by assuming unit-size jobs. Under this assumption, Devanur et al. (2014) gave an optimal e-competitive algorithm for Online Machine Minimization, and Chen and Zhang (2022) gave a $(3e+7) \approx 15.16$-competitive algorithm for Online Rent Minimization. In this paper, we significantly improve the competitive ratio of the Online Rent Minimization problem under unit size to 6, by using a clean oracle-based online algorithm framework.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online Algorithm, Scheduling, Machine Minimization, Rent Minimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.97

Related Version *Full Version*: https://arxiv.org/abs/2306.17241

Funding *Yuhao Zhang*: supported by National Natural Science Foundation of China, Grant No. 62102251.

Acknowledgements The authors would like to thank Minming Li, Pinyan Lu, and Biaoshuai Tao for many insightful discussions on the research topic of this paper.

1 Introduction

Machine Minimization is a classical scheduling problem in combinatorial optimization. We are given n jobs with release time and deadline to schedule. Each job j has a length p_j and must be assigned to a machine for p_j units of time between its release time r_j and its deadline d_j. However, in many practical scenarios, such as cloud computing, we may not need to buy the machines but only rent them for a fixed period of time. This motivates the *Rent Minimization* problem, introduced by Saha [12]. In this problem, we are given a constant T, which represents the duration of a machine rent. The goal is to minimize the number of rents we make to process all jobs within their deadlines.

Another related formulation, inspired by nuclear weapon testing, is the *Calibration* problem, proposed by Bender et al. [3]. In this problem, we are given m machines and a set of jobs that must be completed feasibly. However, before using a machine, we need to *calibrate* it. Each calibration, similar to a rent, activates the machine for a time period of T. The goal is to minimize the number of calibrations to process all jobs on time. The main difference between the Calibration and the Rent Minimization problems is that the former restricts us to have at most m machines working in parallel at any given time, while the latter does not have such a constraint. Therefore, the Rent Minimization problem can be regarded as a special case of the Calibration problem when $m = \infty$.

On the other hand, in the cloud rental scenario and many other practical applications, the computing requests usually increase over time and can be modeled as online released jobs. Therefore, we investigate the problem in an online setting. We do not have any prior knowledge about the jobs before their release time, and need to schedule jobs and rent machines online and irrevocably over time. The goal is to minimize the total number of *rents* for scheduling all jobs. Note that the online generalization is also studied in the Calibration problem by Chen and Zhang [5]. To ensure that online algorithms can schedule all jobs, they also assume $m = \infty$ in their model, which coincides with the Online Rent Minimization model.

Why consider unit-size jobs? Saha [12] proposes an $O(\log(p_{\max}/p_{\min}))$-competitive algorithm for the Online Machine Minimization problem. By paying a constant factor, it can be extended to an $O(\log(p_{\max}/p_{\min}))$-competitive algorithm for the Online Rent Minimization problem. ($p_{\max}$ and $p_{\min}$ are the longest and shortest processing time among all jobs.), which was proved to be the best competitive ratio asymptotically. However, in many real-world applications, one company usually receives similar length requests, so the ratio between $p_{\max}$ and $p_{\min}$ may not be too large; and it is worthwhile to reduce the constant factor of the ratio when $p_{\max}/p_{\min}$ is small. To this end, we focus on the special case of unit-size jobs (i.e., all $p_j = 1$). Note that by partitioning jobs by their length into $\log(p_{\max}/p_{\min})$ groups, the α-competitive unit-size algorithm can be extended to a roughly $(3\alpha \log(p_{\max}/p_{\min}))$-competitive algorithm in the general case.

The unit-size special case has also been considered in the Online Machine Minimization problem [5, 8, 10]. Devanur et al. [8] present an e-competitive algorithm for the Online Machine Minimization problem under unit-size jobs (though earlier work by Bansal et al. [2] implies the same result), and it is the optimal ratio among all deterministic algorithms. Current best lower bound of the online renting problem under unit-size jobs is also e since it is a generalized model. On the positive side, Chen and Zhang [5] study the online renting problem under unit-size jobs. They improve the implicit constant ratio by Saha [12] to $3e + 7 \approx 15.16$. In their algorithm, jobs are distinguished by whether they are long or short based on the length of their time window (i.e., $d_j - r_j$) and are handled separately. Our paper significantly improves the competitive ratio to 6 with a cleaner oracle-based algorithm without identifying whether jobs are short or long.

▶ **Theorem 1.** *There exists an efficient* 6*-competitive algorithm for the online renting problems under unit-size jobs.*

Our techniques. In the work of Chen and Zhang [5], they rent machines for long and short jobs separately; as a result, their final competitive ratio is the sum of two cases, which makes the ratio large. The technical reason behind this result is that they use the e-competitive Online Machine Minimization algorithm by Devanur et al. [8] as a black box, which is only suitable for *short* jobs. (Roughly speaking, it is because we can view $T = \infty$ when jobs are short.) Therefore, they must use another approach to handle long jobs.

In our paper, we formalize and extend the Online Machine Minimization algorithm to an oracle-based framework, instead of using the algorithm as a black box. The oracle-based framework uses an offline algorithm to guide our online decision. Note that the Online

Machine Minimization algorithm also uses an efficient offline optimal algorithm as an oracle. However, we do not know a polynomial offline Rent Minimization algorithm for unit-size jobs. The main algorithmic novelty is that we find an efficient substitute for the optimal algorithm to act as a bridge between online decisions and the optimal offline solution. The oracle is a kind of optimal augmentation algorithm. It is allowed to use a rent length of $3T$, and the rent number is at most OPT with rent length T. It also satisfies some online monotone properties so that we can control the cost of the online algorithm. Finally, we prove that by following the offline oracle and paying a factor of 6 online, we can recover the same ability for scheduling jobs as the offline oracle. This concludes the competitive ratio of 6.

Extension to the model with delay. Chen and Zhang [5] also raise a perspective that the operation *rent* (a.k.a. *calibration* in their paper) needs a non-negative time λ to finish. We call it *Online Rent Minimization with Delay*. They propose an $(3(e+1)\lambda + 3e + 7) \approx (11.15\lambda + 15.16)$-competitive algorithm. We use a black box reduction to extend the algorithm in Theorem 1 and improve the ratio to $6(\lambda + 1)$.

▶ **Theorem 2.** *As a corollary of Theorem 1, there exists an efficient* $6(\lambda + 1)$*-competitive algorithm when we need* λ *time to finish each rent.*

Other related works. Offline Machine Minimization is a well-studied and classic model. Garey and Johnson [9] shows that it is NP-hard. On the algorithm side, Raghavan and Thompson [11] propose an $O(\frac{\log n}{\log \log n})$-approximation algorithm. Later, the ratio has been improved to $O(\sqrt{\frac{\log n}{\log \log n}})$ by Chuzhoy et al. [6]. Whether there exists a constant approximation ratio is still open. Moreover, several special cases are also discussed. Cieliebak et al. [7] focus on the case that each job's active time $(d_j - r_j)$ is small. Yu and Zhang [13] achieve a ratio of 2 in the equal release time case and a ratio of 6 in the equal processing time case.

Scheduling to minimize the number of calibrations is proposed by Bender et al. [3]. The general case is NP-hard even for checking feasibility. Under unit-size jobs, Bender et al. give a 2-approximation algorithm; later, Chen et al. [4] give the first PTAS algorithm. However, it is worth noting that whether the unit-size special case is polynomially solvable is still open. Moreover, Angel et al. [1] introduce the concept of *delay*, which means that each calibration requires λ time to finish. They study the delay setting on the one-machine special case of the offline calibration problem and show that it is polynomially solvable.

2 Preliminaries

We first define the models and introduce the basic notations.

Rent Minimization. We have a set of jobs $\mathcal{J} = \{1, \cdots, n\}$ and a fixed rent length T. For job $j \in \mathcal{J}$, it has a release time r_j, a deadline d_j, and a unit processing time $p_j = 1$. Each job should be assigned to one active machine at an integer time unit $[t, t+1)$, where t is an integer such that $r_j \le t \le d_j - 1$. We can rent a machine at any integer time point t. Then we will have an active machine during $[t, t+T)$. The objective is to minimize the number of machine rents to process all jobs in $\mathcal{J}$.

Online Rent Minimization. In the online version, all jobs are released online, and they become *visible* at their release time. On the other hand, we need to make rent decisions and assign jobs online irrevocably. In particular, at an integer time point t, we have:

- Jobs with release time equal to t become visible.
- We can decide to rent a machine at t or any time after that.
- We can schedule jobs on active machines during the time unit $[t, t+1)$ irrevocably.

Notations on Rent Set. We use a multiset $I = \{[s_1, c_1), [s_2, c_2), \cdots [s_i, c_i), \cdots\}$ to denote a set of rents, where the i-th rent interval starts at s_i and is active in $[s_i, c_i)$. Note that c_i always equals $s_i + T$ when the rent length is T; however, we use the general notation for future reference.

Focusing on the time unit $[t, t+1)$, the number of *active units* $A_I(t)$ is defined as the number of active machines at time t, which means that we can schedule at most $A_I(t)$ jobs at time t. For a given rent set I, we have $A_I(t) = |\{[s_i, c_i) \in I \mid t \in [s_i, c_i)\}|$. We also extend the notation for intervals, such that $A_I(r^*, d^*) = \sum_{t=r^*}^{d^*-1} A_I(t)$ is the number of active units during the time interval $[r^*, d^*)$.

Feasibility of Rent Set. We call a rent set I *feasible* for J, if we can schedule all jobs in J on I. We introduce a lemma based on Hall's Theorem to check whether I is feasible.

For a given jobs set J, we define $J(r^*, d^*) = \{j \in J | r^* \leq r_j < d_j \leq d^*\}$ to represent the jobs that must be assigned inside the interval $[r^*, d^*)$.

▶ **Lemma 3** (Feasibility). *I is feasible for J iff. $\forall r^* \leq d^* \in \mathbb{N},\ A_I(r^*, d^*) \geq |J(r^*, d^*)|$.*

Proof. For any fixed r^* and d^*, the sum of active units provided by I is $A_I(r^*, d^*)$. Each job released and due between this period must be scheduled on these time slots. If there exists a pair of r^* and d^* such that $A_I(r^*, d^*) < |J(r^*, d^*)|$, there is no feasible assignment because of the pigeonhole principle. On the other hand, if the inequality holds for all r^* and d^*, we can view it as a bipartite matching between jobs and active units. There is a feasible assignment by Hall's Theorem. ◀

An Efficient Checker and Scheduler: Earliest Deadline First (EDF). **E**arliest **D**eadline **F**irst is a greedy algorithm that can find a feasible assignment for J on I if and only if I is feasible for J. When we call EDF(J, I), we scan time units from early to late, and assign the released job with the earliest deadline to a free active machine at the current time unit. If a job with deadline d cannot find a free active machine at the time unit $[d-1, d)$, EDF(J, I) fails, and we call d the *fail time* of EDF(J, I). Otherwise, EDF(J, I) succeeds. Bender et al. [3] has already proved that EDF can check the feasibility. It is also worth noting that EDF can be efficiently implemented in $O(n \log n)$ by using a heap, instead of going through all integer time points directly.

▶ **Lemma 4** ([3]). *EDF(J, I) succeeds, i.e., it can find a feasible schedule for J on I, if and only if I is feasible for J.*

Using EDF online. Note that we only make comparisons between released jobs. Therefore, the EDF algorithm can be simulated online : we only need to find a feasible rent set I, and then EDF can automatically find a feasible assignment online.

3 Oracle-based Online Algorithm Framework

Moving towards online algorithms, one natural way is to use an offline algorithm as an *oracle* to suggest the actions of online algorithms. We keep track of this offline algorithm and make corresponding online decisions when the offline algorithm changes along with the online jobs release. Whenever the offline algorithm increases by one at the moment t because of the change of the job set, the online algorithm performs one Batch-Rent at this time t, which is a fixed rent scheme that contains Γ machines. Intuitively, we use these Γ machines to catch up with the one increment of the offline oracle. It is worth noting that the e-competitive algorithm for Online Machine Minimization follows this approach [8].

In our case, compared to Machine Minimization, we have two main differences in our oracle-based framework. The first difference is the job set we input to the oracle. Because of the online fashion, the most natural way is to input the set of all released jobs to the oracle. In Machine Minimization, it works because $T = \infty$ and earlier rents are always more powerful; however, this is not true in Rent Minimization. Indeed, too early rents may cause trouble in Rent Minimization. Intuitively, we are only allowed to make online rent when the offline oracle reports an increment if we want to bound the competitive ratio in the framework. Consider the case where $T = 10$ and two jobs will be due at 100 with release time 0 and 90. If we report the first job to the offline oracle at 0, the offline oracle will return one new rent interval. Following the oracle-based framework, we will make Γ online rent intervals at 0. However, we still need to make more rent intervals at 90, while the offline oracle may not increase. To this end, we use the job set J_t as the job set we input to the oracle at time t. A job j is in J_t if it satisfies the following two properties:

1) It is *visible* at t, i.e., $r_j \leq t$.
2) It is *emergent* at (or before) t, i.e., $d_j \leq t + T$.

Another difference is an augmentation factor α. The oracle is allowed to have αT active time for each rent. Since the existence of a polynomial time optimal offline algorithm for Rent Minimization is still unknown, this factor allows us to find an efficient substitute. We use $\mathsf{Oracle}_\alpha(J, T)$ (instead of $\mathsf{Oracle}(J, \alpha T)$, since the target rent length is still T) to denote an oracle with augmentation factor α. The framework is formalized in Algorithm 1.

Algorithm 1 Oracle-based Online Algorithm Framework.

```
procedure OracleBasedOnline($t$: time, $J$: known jobs, $T$: length of rent)
    $\Delta_t = |\mathsf{Oracle}_\alpha(J_t, T)| - |\mathsf{Oracle}_\alpha(J_{t-1}, T)|$        ▷ $\alpha$ is a positive integer
    Perform $\Delta_t$(if $\Delta_t > 0$) Batch-Rent operations at $t$, consisting of $\Gamma$ machines that start
at or after $t$.
    schedule jobs at $t$ following EDF($J, I$), where $I$ is the current online rent set.
end procedure
```

Then, we discuss how this framework helps us control the number of rents made by the online algorithm. First, as a substitute for the optimal offline algorithm, Oracle_α needs to maintain some properties similar to those of the optimal offline algorithm. Second, Batch-Rent should support the online algorithm to be as powerful as the offline oracle in scheduling all released jobs. We integrate and formalize these messages in the following lemma.

▶ **Lemma 5.** *Let* $\mathsf{OPT}(J, T)$ *be the number of rents used by the optimal offline algorithm to schedule the job set* J*, Algorithm 1 is* Γ*-competitive if these three properties are guaranteed:*

1) *For any job set* J*,* $|\mathsf{Oracle}_\alpha(J, T)| \leq \mathsf{OPT}(J, T)$*;*
2) *The offline oracle is online monotone:* $|\mathsf{Oracle}_\alpha(J_{t_1}, T)| \leq |\mathsf{Oracle}_\alpha(J_{t_2}, T)|$ *if* $t_1 \leq t_2$*;*
3) *Algorithm 1 is feasible for scheduling all online released jobs.*

Proof. The online algorithm makes $\sum_{\Delta_t > 0} \Delta_t$ rent batches, which is exactly $|\mathsf{Oracle}_\alpha(\mathcal{J}, T)|$ by property 2) and is not greater than $|\mathsf{OPT}(\mathcal{J}, T)|$ by property 1). Also, the output satisfies the feasibility requirement by property 3). Therefore, Algorithm 1 is Γ-competitive. ◀

The e-competitive algorithm for Online Machine Minimization. We can use the framework to understand the e-competitive Online Machine Minimization algorithm.

- Oracle is the optimal offline algorithm, and we set $\alpha = 1$. The monotonicity directly comes from optimality.
- J_t is the set of visible jobs at t because all jobs are emergent.
- Each Batch-Rent contains e new machines in average; for simplicity, we omit any rounding issues related to e.

Choice of the oracle. Recall that we do not have an efficient optimal algorithm for Rent Minimization currently. It remains to find a suitable substitute that also uses a small number of rents (property 1). One candidate algorithm may be the Lazy-Binning algorithm by Bender et al. [3], which requires an augmentation factor of 2 to satisfy property 1). However, Lazy-Binning algorithm, as well as other relatively simple 2 approximation algorithms we come up with, cannot guarantee monotonicity. This will make us fail to bound the competitive ratio of the online algorithm. In the next section, we introduce our oracle with an augmentation factor of 3, called the semi-online algorithm, which provides all the properties we need in Lemma 5.

4 The Semi-Online Algorithm

In this section, we introduce the semi-online algorithm shown as Algorithm 2, which uses an augmentation factor of 3 and acts as the Oracle_3 in our framework.

Algorithm 2 The Semi-Online Algorithm.

```
procedure SemiOnline($J$: input job set, $T$: length of rent)
    $J', I \leftarrow \varnothing$                              ▷ $I$ is a multiset for rents.
    $\tau_j = \max\{r_j, d_j - T\}$ for all $j$.
    for $j \in J$ in non-decreasing order of $\tau_j$ do
        $J' \leftarrow J' \cup \{j\}$
        if EDF$(J', I)$ fails then
            $I \leftarrow I \cup \{[\tau_j - T, \tau_j + 2T)\}$      ▷ A rent that starts at $\tau_j - T$ with length $3T$.
        end if
    end for
    return $I$
end procedure
```

We call Algorithm 2 semi-online, because the enumerating order is exactly the same as how J_t increases in the oracle-based framework. Thus, if we have some new jobs with $r_j = t$ or $d_j - T = t$ when the online time moves from $t-1$ to t, the only possible difference between $\mathsf{SemiOnline}(J_{t-1}, T)$ and $\mathsf{SemiOnline}(J_t, T)$ is some rent intervals of $[t-T, t+2T)$. This observation could be formalized into the following properties of the semi-online algorithm.

▶ **Lemma 6** (Strong Monotonicity). *We have the following two properties for Algorithm 2.*

1. *For any J_{t_1} and J_{t_2} where $t_1 \le t_2$, we have* $\mathsf{SemiOnline}(J_{t_1}, T) \subseteq \mathsf{SemiOnline}(J_{t_2}, T)$.
2. $\mathsf{SemiOnline}(J_t, T) \setminus \mathsf{SemiOnline}(J_{t-1}, T)$ *is a multiset of a fixed rent interval* $[t-T, t+2T)$.

Proof. Intuitively, the reason behind the lemma is that the order of τ_j is the same as the order in which we insert jobs into J_t as t increases. Formally speaking, compare J_{t_1} and J_{t_2} and consider a job $j \in J_{t_2} \setminus J_{t_1}$. By definition, we have that $\tau_j \ge \max_{j' \in J_{t_1}} \tau_{j'}$. Therefore, Algorithm 2 first enumerates the jobs in J_{t_1} and then the jobs in $J_{t_2} \setminus J_{t_1}$, which concludes the first property immediately. For the second property, the reason is that $J_t \setminus J_{t-1}$ is a set of jobs with $r_j = t$ or $d_j - T = t$. In other words, we enumerate them after jobs in J_{t-1}. Thus, if I continues to grow when we enumerate them, the new interval must be $[t - T, t + 2T)$. ◀

The strong monotonicity in Lemma 6 suffices to show the weak monotonicity in property 2) of Lemma 5. On the other hand, these two properties are also used in the proof of property 3) later. It remains to prove property 1) by bounding the cardinality of the semi-online algorithm's solution.

▶ **Lemma 7.** $|\mathsf{SemiOnline}(J, T)| \le \mathsf{OPT}(J, T)$*, and* $\mathsf{SemiOnline}(J, T)$ *is feasible for* J*.*

Before proving Lemma 7, we introduce coOPT so that we can better understand the solution structure.

▶ **Definition 8** (Optimal complement solution)**.** *For a job set* J*, rent length* T*, and a given rent set* I *(which may not be length* T*), the optimal complement solution of* I*, denoted as* $\mathsf{coOPT}(I)$*, is defined as a rent set of length* T *with minimum cardinality such that* $I \cup \mathsf{coOPT}(I)$ *is feasible for* J*.*

▶ **Fact 9.** $\mathsf{coOPT}(\varnothing) = \mathsf{OPT}, \mathsf{coOPT}(\mathsf{OPT}) = \varnothing$.

Consider a rent set I that is infeasible for J. Below, we state the main property of coOPT.

▶ **Lemma 10.** *Let* d *be the fail time of* $\mathsf{EDF}(J, I)$*. There exists a* $\mathsf{coOPT}(I)$ *such that there is a rent interval* $[s, s + T) \in \mathsf{coOPT}(I)$ *that satisfies:* $d - T \le s < d$*.*

Proof. We use coOPT as a shorthand for $\mathsf{coOPT}(I)$ in this proof. Let $[s, s + T)$ be the earliest interval in coOPT.

First, we show that $s < d$. Suppose, for a contradiction, that $s \ge d$. Let j be the job that fails in $\mathsf{EDF}(J, I)$, where J is a fixed given job set. Then, j has no more active units in coOPT, since all rent intervals start at or after d. But this contradicts the definition of coOPT, which is a feasible rent set for J.

Second, we show that $s \ge d - T$ can be true. If this is not true, we construct a new rent set $\mathsf{coOPT}' = \mathsf{coOPT} \setminus \{[s, s + T)\} \cup \{[d - T, d)\}$. That is, we replace the rent interval at s with another one at $d - T$. We claim that $\mathsf{coOPT}' \cup I$ is also feasible for J. This means that coOPT' is also a feasible coOPT, and $[s' = d - T, T)$ is a feasible rent interval that satisfies our desired condition.

To prove our claim, we fix an arbitrary choice of $r^* \le d^* \in \mathbb{N}$, and we verify the condition in Lemma 3, i.e., $A_{I \cup \mathsf{coOPT}'}(r^*, d^*) \ge |J(r^*, d^*)|$.

We consider two cases:

- Case 1: $d^* < d$. If the condition does not hold, we have $A_I(r^*, d^*) \le A_{I \cup \mathsf{coOPT}'}(r^*, d^*) < J'(r^*, d^*)$, which means that $\mathsf{EDF}(I, J')$ must fail no later than d^* since the active units are not enough beforehand. This contradicts the definition of d.
- Case 2: $d^* \ge d$. The only difference between coOPT and coOPT' is the contribution of active units by $[s, s + T)$ and $[d - T, d)$. We prove that $[d - T, d)$ must provide at least as many active units as $[s, s + T)$ does. Referring to Figure 1, we see that $[d - T, d)$ has more active units than $[s, s + T)$ in $[s + T, d)$, and vice versa in $[s, d - T)$. Since $d^* \ge d$, we need $r^* \le d - T$ to reach the advantage area of coOPT; however, $[r^*, d^*)$ then covers the whole part of $[d - T, d)$. This implies that the total contribution of coOPT never exceeds that of coOPT'.

The discussion concludes the claim. ◀

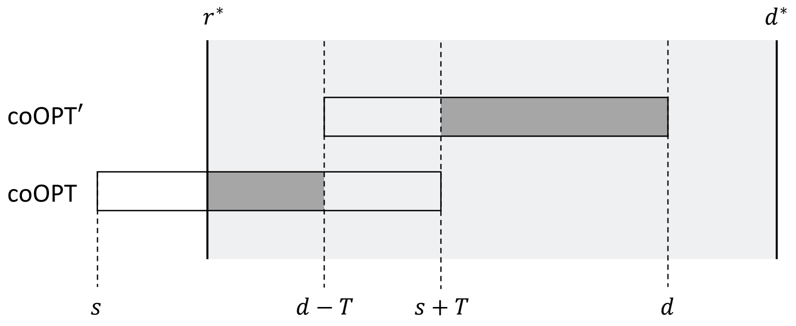

Figure 1 coOPT' contributes at least as many active units as coOPT on $[r^*, d^*)$ when $d^* \geq d$.

▶ **Lemma 11.** *At the end of each iteration of j, I is feasible for J'.*

Proof. We prove it by induction. In the base case, I is feasible for J' when they are $\varnothing$. Then, assume the lemma is true after the $(j-1)$-th iteration. At the j-th iteration, we add a job j to J'. This means that $\forall r^* \leq r_j,\ d^* \geq d_j$, $|J'(r^*, d^*)|$ will increase by one. If $\mathsf{EDF}(J', I)$ is already feasible, we are done. Otherwise, the algorithm will employ a new $3T$ length rent interval $[\tau_j - T, \tau_j + 2T)$. Notice that $d^* \geq d_j \geq \tau_j$. Every $A_I(r^*, d^*)$ also increases by at least one. Thus, we still have $A_I(r^*, d^*) \geq |J'(r^*, d^*)|$ after we employ $[\tau_j - T, \tau_j + 2T)$ (at the end of the j-th iteration). ◀

▶ **Corollary 12.** $\mathsf{SemiOnline}(J, T)$ *is feasible for J.*

▶ **Lemma 13.** *In the j-th iteration, if* $\mathsf{EDF}(J', I)$ *is infeasible in Algorithm 2, before we rent $[t - T, t + 2T)$, we have*

$$|\mathsf{coOPT}(I \cup \{[t - T, t + 2T)\})| \leq |\mathsf{coOPT}(I)| - 1.$$

Proof. By the condition of the lemma and Lemma 11, we have $\mathsf{EDF}(J', I)$ is infeasible while $\mathsf{EDF}(J' \setminus \{j\}, I)$ is feasible. By the enumerating order, all the jobs j' in J' must satisfy $\max\{r_j, d_{j'} - T\} \leq \tau_j$. Therefore, $\mathsf{EDF}(J', I)$ must fail at a deadline $d \leq \tau_j + T$. On the other hand, for all $j' \in J$ such that $d_{j'} < \tau_j$, we must have $j' \in J' \setminus \{j\}$, also because of the enumerating order. Thus, we can show that $d \geq \tau_j$. Otherwise, $J' \setminus \{j\}$ would be infeasible for I, which is a contradiction. In conclusion, we show that the failure time d of $\mathsf{EDF}(J', I)$ satisfies $\tau_j \leq d < \tau_j + T$.

Finally, by Lemma 10, there exists a coOPT with rent interval $[s, s + T]$ such that $d - T \leq s < d$. It implies that $\tau_j - T < s < \tau_j + T$. Therefore $[s, s + T)$ is always a subset of $[\tau_j - T, \tau_j + 2T)$. We have

$$|\mathsf{coOPT}(I \cup \{[t - T, t + 2T)\})| \leq |\mathsf{coOPT}(I \cup \{s\})| = |\mathsf{coOPT}(I)| - 1.$$ ◀

Proof of Lemma 7. Recall Fact 9 that $\mathsf{coOPT}(\varnothing) = \mathsf{OPT}$. It follows that $\mathsf{SemiOnline}$ rents at most OPT times as a corollary of Lemma 13. ◀

5 The 6-competitive Online Algorithm

It remains to define the rent scheme for each $\mathsf{Batch\text{-}Rent}$ in the framework. For completeness, we formally describe the algorithm in Algorithm 3.

Algorithm 3 The Online algorithm.

```
procedure OnlineRent($t$: time, $J$: known jobs, $T$: length of rent)
    $\Delta = |\mathsf{SemiOnline}(J_t, T)| - |\mathsf{SemiOnline}(J_{t-1}, T)|$
    Perform $\Delta$ Batch-Rent at time $t$, each consists of 6 machines: 4 at $t$ and 2 at $t+T$.
    schedule jobs at $t$ following $\mathsf{EDF}(J, I)$, where $I$ is the current online rent set.
end procedure
```

Next, we prove the property 3) of Lemma 5 by our design of Batch-Rent, i.e., to show Algorithm 3 is feasible for the total job set $\mathcal{J}$. Combining with the property 1) and 2) by the semi-online algorithm, we can conclude our online algorithm is 6-compeititve as claimed in Theorem 1.

First, we introduce an obvious relationship between online and semi-online algorithms.

▶ **Fact 14.** *At every moment t, there always exists a bijection from one semi-online rent batch to one online rent batch, such that both batches are at the same time: $4 \times [t, t+T) + 2 \times [t+T, t+2T)$ in* Online $\mapsto$ *$[t-T, t+2T)$ in* SemiOnline.

Proof. This fact is directly implied by the second property of Lemma 6. Whenever $\mathsf{SemiOnline}(J_t, T)$ increases from $\mathsf{SemiOnline}(J_{t-1}, T)$ by some rent intervals of $[t-T, t+2T)$, the new batches must be $4 \times [t, t+T) + 2 \times [t+T, t+2T)$. Each of them can correspond to one $[t-T, t+2T)$. ◀

We prove the feasibility by showing that the active units provided by the online algorithm are always enough for the possible jobs inside any possible interval $[r^*, d^*)$.

▶ **Lemma 15.** *Let I be the rent sets made by our algorithm. We have $\forall r^* \le d^* \in \mathbb{N}$, $A(r^*, d^*) \ge |\mathcal{J}(r^*, d^*)|$, where we use A to denote A_I for simplicity.*

We remark that Lemma 15 provides the necessary information to prove the correctness via the feasibility lemma (Lemma 3). It only remains to complete the proof of Lemma 15.

5.1 Proof of Lemma 15

Let us fix an arbitrary range $r^* \le d^*$, and discuss $A(r^*, d^*)$ and $\mathcal{J}(r^*, d^*)$ separately. First, we discuss $A(r^*, d^*)$. The behavior of the online algorithm can be represented by a set of online rent batches. Note that only those rent batches that start in $(r^* - 2T, d^*)$ can provide active units inside $[r^*, d^*)$. Therefore, we only discuss a subset B of all rent batches with a start time in $(r^* - 2T, d^*)$. Each b means a Batch-Rent made by Algorithm 3. We use $t(b)$ to mean its decision time. That is, a batch b contains 4 rent intervals of $[t(b), t(b)+T)$, and 2 of $[t(b)+T, t(b)+2T)$.

We partition the time interval $[r^*, d^*)$ by several critical time points. The first time point is $\theta_1 = \max\left\{\left(\frac{r^*+d^*}{2}\right), d^* - T\right\}$. It means that $\theta_1 = d^* - T$ when $d^* - r^* \ge 2T$ and $\theta_1 = \left(\frac{r^*+d^*}{2}\right)$ when $d^* - r^* < 2T$. Remark that in both cases, $\theta_1 \ge d^* - T$. Then we recursively define $\theta_{i+1} = \left(\frac{\theta_i + d^*}{2}\right)$ for every $i \ge 1$ until $\lfloor \theta_i \rfloor = d^* - 1$. Besides, we let $\theta_0 = r^* - 1$. For $i \ge 1$, we call $[\lfloor \theta_{i-1} \rfloor + 1, \lfloor \theta_i \rfloor]$ the i-th sub-interval, which is the minimal sub-interval of $(\theta_{i-1}, \theta_i]$ that contains all integers in it. We define $B_i \subseteq B$ as the set of rent batches starting in the i-th sub-interval. Moreover, we let B_0 be the set of rent batches starting in $(r^* - 2T, r^*]$.

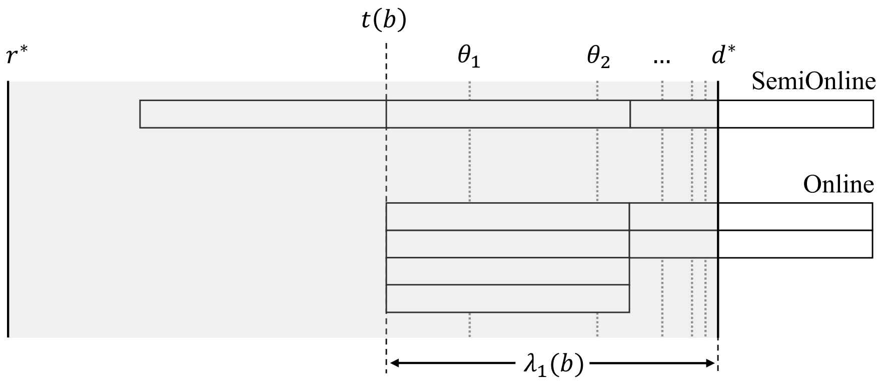

Figure 2 An example for $b \in B_1$ when $d^* - r^* > 2T$. The shaded area is the considered $[r^*, d^*)$, and the online rent batch provides $2\lambda_1(b) + 2\min\{\lambda_1(b), T\}$ active units to it.

For each $b \in B$, recall that $t(b)$ is the time it was allocated, and we let $\lambda_i(b)$ be the length of the intersecting interval of $[t(b), t(b) + 2T)$ and $[\lfloor\theta_{i-1}\rfloor + 1, d^*)$. Note that $\lambda_1(b)$ represents the intersecting interval with the whole $[r^*, d^*)$. Let $L = \min\{2T, d^* - r^*\} = 2(d^* - \theta_1)$ denote the maximum possible length in λ_1, and by our partition method. It follows the property of the length of sub-intervals by our partition.

▶ **Lemma 16** (Partition length property). *For any batch $b \in B_i$ where $i \geq 1$, the intersection of $[t(b), t(b) + 2T)$ and $[\lfloor\theta_{i-1}\rfloor + 1, d^*)$ satisfies: $2^{-i} \cdot L \leq \lambda_i(b) \leq 2^{1-i} \cdot L$.*

Proof. For $i = 1$, $\lambda_1(b) = \min\{2T, d^* - t(b)\}$. By definition, $r^* \leq t(b) \leq \lfloor\theta_1\rfloor$, hence

$$L/2 = \min\{T, d^* - \theta_1\} \leq \min\{2T, d^* - t(b)\} \leq \min\{2T, d^* - r^*\} = L.$$

For $i \geq 2$, by definition $d^* - \theta_i = 2^{-i} \cdot L$. Because $t(b) \in [\lfloor\theta_{i-1}\rfloor + 1, \lfloor\theta_i\rfloor] \subseteq (\theta_{i-1}, \theta_i]$, we have $2^{-i} \cdot L \leq \lambda_i(b) \leq \min\{2T,\ 2^{1-i} \cdot L\}$. Since $2 \cdot L \leq 2T$, we conclude the lemma. ◀

Then, we present the lemmas for a lower bound of active units and an upper bound of job numbers.

▶ **Lemma 17** (Lower bound of active units).

$$\begin{aligned} A(r^*, d^*) \geq & \sum_{b \in B_0} (2\lambda_1(b) + 2\max\{\lambda_1(b) - T, 0\}) \\ & + \sum_{b \in B_1} (2\lambda_1(b) + 2\min\{\lambda_1(b), T\}) \\ & + \sum_{i \geq 2} 4 \cdot (2^{-i} \cdot L) \cdot |B_i|. \end{aligned}$$

Proof. Three terms on the RHS of the inequality are counting of B_0, B_1 and $B_{\geq 2}$, where the first two are straightforward counting as shown in Figure 2, and the last term was scaled down a bit by Lemma 16:

$$4 \sum_{i \geq 2} \sum_{b \in B_i} \lambda_1(b) = 4 \sum_{i \geq 2} \sum_{b \in B_i} \lambda_i(b) \geq \sum_{i \geq 2} 4 \cdot (2^{-i} \cdot L) \cdot |B_i|.$$ ◀

Let $\mathcal{J}_i$ be the job set with deadline at most d^* and released in the i-th subinterval:

$$\mathcal{J}_i = \{j \in \mathcal{J} \mid \lfloor \theta_{i-1} \rfloor + 1 \le r_j \le \lfloor \theta_i \rfloor,\ d_j \le d^*\}\,.$$

We provide an upper bound of $\mathcal{J}_i$ by the performance of our algorithm.

▶ **Lemma 18.** *Let I_t be the semi-online batches allocated at or before time t. We have that $\mathcal{J}_i$ is no more than the active units after $\lfloor \theta_{i-1} \rfloor + 1$ provided by* SemiOnline *at $\lfloor \theta_i \rfloor$. i.e.,*

$$\mathcal{J}_i \le A_{I_{\lfloor \theta_i \rfloor}}(\lfloor \theta_{i-1} \rfloor + 1, d^*).$$

Proof. Let us observe the time point $t = \lfloor \theta_i \rfloor$. SemiOnline$(J_t, T)$ reports I_t at this time. By Lemma 11, I_t is feasible for J_t. By the definition of θ_i, we prove $t \ge d^* - T$ because $\theta_1 \ge d^* - T$. Thus, all jobs with deadlines at most d^* are already in J_t, and combining with the feasibility of I_t, we have:

$$\mathcal{J}_i = J_t(\lfloor \theta_{i-1} \rfloor + 1, d^*) \le A_t(\lfloor \theta_{i-1} \rfloor + 1, d^*).$$

The lemma then concludes because we define $t = \lfloor \theta_i \rfloor$. ◀

▶ **Lemma 19** (Upper bound of job numbers)**.** *We have two different upper bounds for $\mathcal{J}_i$:*

- $i = 1$: $$|\mathcal{J}_1| \le \sum_{b \in B_0} \lambda_1(b) + \sum_{b \in B_1} (\lambda_1(b) + L/2)\,.$$
- $i \ge 2$: $$|\mathcal{J}_i| \le \sum_{b \in B_0} \lambda_i(b) + 2 \cdot (2^{-i} \cdot L) \cdot \sum_{j=1}^{i} |B_j|.$$

Proof. In this proof, we apply Lemma 18 and count the number of active units after θ_{i-1} by $I_{\lfloor \theta_i \rfloor}$. Note that by Fact 14, each $3T$ length rent interval in $I_{\lfloor \theta_i \rfloor}$ corresponds to an online Batch-Rent.

First, let us consider the case $i = 1$. $I_{\lfloor \theta_i \rfloor}$ corresponds to the online batches with $t(b) \le I_{\lfloor \theta_i \rfloor}$. Note that the ending time of b and its corresponding semi-online rent interval are both $t(b) + 2T$. Thus, $A_{I_{\lfloor \theta_i \rfloor}}(\lfloor \theta_{i-1} \rfloor + 1, d^*)$ corresponds to B_0 and B_1.

For $i = 1$, we keep the B_0 straightforward and calculate the upper bound of active units provided by the corresponding semi-online batch for each batch b in B_1. Note that the semi-online rent set spans $[t(b) - T, t(b) + 2T)$. We split the $3T$ interval into first T and last $2T$: the latter could be upper bounded by $\lambda_1(b)$ using Lemma 16, and we could find that the former is at most $L/2$:

- When $d^* - r^* < 2T$, $t(b) - r^* < (d^* - r^*)/2$, then $\min\{T, t(b) - r^*\} < L/2$.
- When $d^* - r^* \ge 2T$, $L = 2T$, and then $\min\{T, t(b) - r^*\} \le T = L/2$.

So we can conclude that

$$|\mathcal{J}_1| \le \sum_{b \in B_0} \lambda_1(b) + \sum_{b \in B_1} (\lambda_1(b) + L/2)\,.$$

For the case $i \ge 2$, all jobs released in sub-interval i can be allocated at most $[\theta_{i-1} + 1, d^*)$, and by Lemma 16 each semi-online batch covers at most $2 \cdot (2^{-i} \cdot L)$. Like $i = 1$, the inequality follows direct counting on $\cup_{j=0}^{i} B_j$. ◀

Thus far, we are ready to prove Lemma 15 by a charging argument.

Proof of Lemma 15. Recall that in Lemma 17 we have

$$\begin{aligned} A(r^*, d^*) \geq & \sum_{b \in B_0} (2\max\{\lambda_1(b) - T, 0\} + 2\lambda_1(b)) + \sum_{b \in B_1} (2\lambda_1(b) + 2\min\{T, \lambda_1(b)\}) \\ & + \sum_{i \geq 2} 4 \cdot (2^{-i} \cdot L) \cdot |B_i|. \end{aligned} \tag{1}$$

Also, by Lemma 19,

$$\mathcal{J}(r^*, d^*) \leq \sum_{b \in B_0} \lambda_1(b) + \sum_{b \in B_1} (\lambda_1(b) + L/2) + \sum_{i \geq 2} \left(\sum_{b \in B_0} \lambda_i(b) + 2 \cdot (2^{-i} \cdot L) \cdot \sum_{j=1}^{i} |B_j| \right). \tag{2}$$

Using these two inequalities, we charge the upper bound of $\mathcal{J}(r^*, d^*)$ and the lower bound of $A(r^*, d^*)$ to each $b \in B$. We prove that for each b, the charged amount of $\mathcal{J}(r^*, d^*)$'s upper bound is at most $A(r^*, d^*)$'s lower bound.

First, for $b \in B_0$, the contribution of b to the lower bound of $A(r^*, d^*)$ (i.e., RHS of Equation (1)) is: $2\max\{\lambda_1(b) - T, 0\} + 2\lambda_1(b)$. The contribution to the upper bound of $\mathcal{J}(r^*, d^*)$ (i.e., RHS of Equation (2)) is: $\lambda_1(b) + \sum_{i \geq 2} \lambda_i(b)$. Note that $b \in B_0$ only contributes to a prefix of $[r, d)$, it is easy to see that $\lambda_i(b) \leq 2^{1-i}\lambda_1(b)$, and thus

$$\lambda_1(b) + \sum_{i \geq 2} \lambda_i(b) \leq \lambda_1(b) + \sum_{i \geq 2} 2^{1-i}\lambda_1(b) < 2\lambda_1(b).$$

We are done for $b \in B_0$.

Second, for $b \in B_1$, the contribution of b to the lower bound of $A(r^*, d^*)$ (i.e., RHS of Equation (1)) is: $2\lambda_1(b) + 2\min\{T, \lambda_1(b)\}$. The contribution to the upper bound of $\mathcal{J}(r^*, d^*)$ (i.e., RHS of Equation (2)) is: $\lambda_1(b) + L/2 + \sum_{i \geq 2} 2 \cdot (2^{-i} \cdot L)$. Then, we have

$$\begin{aligned} \lambda_1(b) + L/2 + \sum_{i \geq 2} 2 \cdot (2^{-i} \cdot L) & < \lambda_1(b) + L/2 + 2 \cdot \left(2^{-1} \cdot L\right) \\ & \leq \lambda_1(b) + L/2 + 2 \cdot \min\{T, \lambda_1(b)\} \\ & \leq 2\lambda_1(b) + 2\min\{T, \lambda_1(b)\}. \end{aligned}$$

The last inequality holds by Lemma 16. Therefore, we are done for $b \in B_1$.

Finally, for $b \in B_{i \geq 2}$, the contribution of b to the lower bound of $A(r^*, d^*)$ (i.e., RHS of Equation (1)) is: $4 \cdot (2^{-i} \cdot L)$. The contribution to the upper bound of $\mathcal{J}(r^*, d^*)$ (i.e., RHS of Equation (2)) is: $\sum_{i' \geq i} 2 \cdot (2^{-i'} \cdot L)$. We are done because $\sum_{i' \geq i} 2^{-i'} < 2^{1-i}$. Summing up three parts, we have proved that $A(r^*, d^*) \geq \mathcal{J}(r^*, d^*)$. ◀

5.2 A Remark on Running Time

We only need to recalculate SemiOnline if the job set gets updated, and the procedure of reconstructing the rent set in SemiOnline can be maintained incrementally. Thus, it is possible to implement the algorithm calling EDF at most $n + \mathsf{OPT} \leq 2n$ times, and hence achieve a worst case guarantee of $O(n^2 \log n)$. Also, note that a job can influence the calculation of SemiOnline for at most $O(T)$ time units, so the SemiOnline can also update in $O(nw \log w)$ if there are at most $O(w)$ jobs within any interval of length $O(T)$. Therefore, the online algorithm is efficient in terms of worst case guarantee and also average online updating.

6 Online Rent Minimization with Delay

In the version with delay, we are also given an online released job set $\mathcal{J}$ and a rent length T. We aim to minimize the number of rents needed to process all jobs. The notations are the same as in the Online Rent Minimization problem. Moreover, we are given a nonnegative integer λ as the delay parameter. It means that if we rent a machine at time t, we will have an active machine at $[t+\lambda, t+\lambda+T)$. Note that it is impossible to serve an unknown emergency job with $d_j - r_j \leq \lambda$ online; following Chen and Zhang [5], we require that the active time $d_j - r_j$ is at least $\lambda + 1$.

We use the following reduction lemma and our 6-competitive no-delay algorithm as a black box to prove Theorem 2. Chen and Zhang [5] also mention this approach.

▶ **Lemma 20** (Reduction). *If* $\mathsf{ALG}(\mathcal{J}) \leq \Gamma \cdot \mathsf{OPT}(\mathcal{J})$ *for every job set* $\mathcal{J}$*, we have an algorithm* ALG_λ *that guarantees*

$$\mathsf{ALG}_\lambda(\mathcal{J}, \lambda) \leq \Gamma \cdot (\lambda+1) \cdot \mathsf{OPT}(\mathcal{J}),$$

if $\forall j \in \mathcal{J}$, $d_j - r_j \geq \lambda + 1$.

After proving Lemma 20, Theorem 2 follows directly. See the full version for a complete proof.

▶ **Theorem 2.** *As a corollary of Theorem 1, there exists an efficient* $6(\lambda+1)$*-competitive algorithm when we need* λ *time to finish each rent.*

7 Conclusion and Future Work

In conclusion, our main contribution is a 6-competitive algorithm for the Online Rent Minimization problem under unit-size jobs, which follows the oracle-based framework.

Since the Online Rent Minimization problem is a generalization of the Online Machine Minimization problem, where we have an optimal e-competitive algorithm, one major question is: Is the Rent Minimization problem strictly harder than the Machine Minimization problem?

On the other hand, we are also interested in the power of oracle-based algorithms. Note that the optimal e-competitive algorithm for Machine Minimization follows the oracle-based framework. It is interesting to ask: What is the best competitive ratio we can achieve for the Online Rent Minimization problem by using the oracle-based framework? The Semi-Online captures our current understanding of the possible range of optimal solutions, so replacing it with an optimal oracle cannot improve ratio directly by the same argument in the paper. Is it possible to obtain a better ratio with access to an optimal oracle?

References

1 Eric Angel, Evripidis Bampis, Vincent Chau, and Vassilis Zissimopoulos. On the Complexity of Minimizing the Total Calibration Cost. In Mingyu Xiao and Frances Rosamond, editors, *Frontiers in Algorithmics*, Lecture Notes in Computer Science, pages 1–12, Cham, 2017. Springer International Publishing. `doi:10.1007/978-3-319-59605-1_1`.

2 Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), March 2007. `doi:10.1145/1206035.1206038`.

3 Michael A. Bender, David P. Bunde, Vitus J. Leung, Samuel McCauley, and Cynthia A. Phillips. Efficient scheduling to minimize calibrations. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 280–287, Montréal Québec Canada, July 2013. ACM. `doi:10.1145/2486159.2486193`.

4 Lin Chen, Minming Li, Guohui Lin, and Kai Wang. Approximation of Scheduling with Calibrations on Multiple Machines (Brief Announcement). In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 237–239, Phoenix AZ USA, June 2019. ACM. doi:10.1145/3323165.3323173.

5 Zuzhi Chen and Jialin Zhang. Online scheduling of time-critical tasks to minimize the number of calibrations. *Theoretical Computer Science*, page S0304397522000548, January 2022. doi:10.1016/j.tcs.2022.01.040.

6 J. Chuzhoy, S. Guha, S. Khanna, and J. Naor. Machine Minimization for Scheduling Jobs with Interval Constraints. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 81–90, Rome, Italy, 2004. IEEE. doi:10.1109/FOCS.2004.38.

7 Mark Cieliebak, Thomas Erlebach, Fabian Hennecke, Birgitta Weber, and Peter Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In Jean-Jacques Lévy, Ernst W. Mayr, and John C. Mitchell, editors, *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004), 22-27 August 2004, Toulouse, France*, volume 155 of *IFIP*, pages 209–222. Kluwer/Springer, 2004. doi:10.1007/1-4020-8141-3_18.

8 Nikhil Devanur, Konstantin Makarychev, Debmalya Panigrahi, and Grigory Yaroslavtsev. Online Algorithms for Machine Minimization. *arXiv:1403.0486 [cs]*, March 2014. arXiv:1403.0486.

9 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

10 Mong-Jen Kao, Jian-Jia Chen, Ignaz Rutter, and Dorothea Wagner. Competitive design and analysis for machine-minimizing job scheduling problem. In Kun-Mao Chao, Tsan-sheng Hsu, and Der-Tsai Lee, editors, *Algorithms and Computation – 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012. Proceedings*, volume 7676 of *Lecture Notes in Computer Science*, pages 75–84. Springer, 2012. doi:10.1007/978-3-642-35261-4_11.

11 Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Comb.*, 7(4):365–374, 1987. doi:10.1007/BF02579324.

12 Barna Saha. Renting a Cloud. In Anil Seth and Nisheeth K. Vishnoi, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013)*, volume 24 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 437–448, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.FSTTCS.2013.437.

13 Guosong Yu and Guochuan Zhang. Scheduling with a minimum number of machines. *Oper. Res. Lett.*, 37(2):97–101, 2009. doi:10.1016/j.orl.2009.01.008.