Competitive Analysis for Multi-Commodity Ski-Rental Problem

Binghan Wu, **Wei Bao**, **Dong Yuan** and **Bing Zhou**
Faculty of Engineering, The University of Sydney
biwu6051@uni.sydney.edu.au, {wei.bao, dong.yuan, bing.zhou}@sydney.edu.au

Abstract

We investigate an extended version of the classical ski-rental problem with multiple commodities. A customer uses a set of commodities altogether, and he/she needs to choose payment options to cover the usage of each commodity without the knowledge of the future. The payment options of each commodity include (1) renting: to pay for an on-demand usage and (2) buying: to pay for the lifetime usage. It is a novel extension of the classical ski-rental problem which deals with only one commodity. To address this problem, we propose a new online algorithm called the Multi-Object Break-Even (MOBE) algorithm and conduct competitive analysis. We show that the *tight lower and upper bounds* of MOBE algorithm's competitive ratio are $\frac{e}{e-1}$ and 2 respectively against adaptive adversary under arbitrary renting and buying prices. We further prove that MOBE algorithm is an *optimal* online algorithm if commodities have the same rent-to-buy ratio. Numerical results verify our theoretical conclusion and demonstrate the advantages of MOBE in a real-world scenario.

1 Introduction

The classical ski-rental problem [Karlin *et al.*, 2003] describes the dilemma where a customer trades off between two payment options for one commodity without the knowledge of the future: 1) renting, the customer only pays for an on-demand usage; 2) buying: the customer pays for lifetime usage at once, and no further payment will be incurred after that. Existing works have obtained the optimal online algorithm by minimizing the competitive ratio and extending the problem to adopt more complex payment options [Fujiwara *et al.*, 2020; Patt-Shamir and Yadai, 2020; Wang *et al.*, 2020; Wu *et al.*, 2021].

However, literature performs sub-optimally when the customer uses a set of commodities altogether and each commodity is rented/brought individually. In this paper, we focus on this *multi-commodity ski-rental problem*. In this case, simply applying the classical ski-rental algorithm causes degraded performance in terms of competitive ratio. For example, we consider a ski-helmet-suit rental problem as follows: When a customer goes skiing, he/she needs a pair of skis, a helmet, and a suit (three commodities) every time. Each commodity is rented or bought individually. Suppose the prices of buying each commodity are all \$10, and the rental prices are all \$0.1. If we employ the classical ski-rental algorithm on each commodity, the customer will buy all commodities when he/she goes skiing for the hundredth time. Then, the competitive ratio is $\frac{(9.9+10)\times 3}{10\times 3} = 1.99$. (Note that by employing the classical ski-rental algorithm, 2 bounds the competitive ratio for all possible situations [Karlin *et al.*, 2003], and 1.99 is the optimal competitive ratio in this example.) However, what out of the expectation is that 1.99 is no longer the optimal competitive ratio here. The customer can buy the three commodities at the 45^{th}, 77^{th}, and 100^{th} time of use respectively. It surprisingly reduces the competitive ratio to $\frac{4.4+7.6+9.9+30}{10+10+10} = 1.73$. The ratio of 1.73 is reached when three commodities are used 100 times.

An online algorithm jointly considering multiple commodities can improve the competitive ratio, however, the problem then becomes more challenging due to the following reasons. First, arbitrary commodities can be involved in the system. We need to design an algorithm to systematically consider all involved commodities and their payment options, which produce a large solution space. Second, different commodities may have *different rent-to-buy ratios*. Such unbalanced commodities further complicate choices between renting and buying. Finally, although we may reduce the competitive ratio through the joint considerations of commodities, this reduction should have a limit. We also aim to answer how much the competitive ratio can be reduced.

Please note that besides the aforementioned ski-helmet-suit rental problem, there are many real-world examples of the multi-commodity ski-rental problem. Here are some more examples.

1. Bahncard problem [Fleischer, 2001] with bus and train. A customer takes the bus and train (two commodities) to go to work. For each type of transportation (commodity), he/she has two payment options. The first option is paying for a one-time ride (renting), and the other is purchasing a Bahncard to cover the monthly ride (buying). In the past, one could reliably estimate the times of commutes per month to decide whether to buy a Bahncard. However, due to the COVID-19 pandemic, people may work from home for a number of days in a month, and the number of commutes in one month is

uncertain.

2. Cloud service acquisition problem with multiple service components [Wang *et al.*, 2015]. The customer utilizes multiple service components (commodities) including virtual machines, storage, and serverless messaging to implement a cloud application. Every time the application is used, the customer must choose among payment options of each service component (commodity) to cover the usages. The payment options include 1) pay-as-you-go (PAYG): to pay for the on-demand usage of one service component (renting), and 2) reservation: to pay an up-front fee without further cost for one service component (buying). By convention, the literature assumes no prior statistics of the usages due to the high fluctuations [Guo *et al.*, 2019] of the workloads and statistically non-stationary demands [Stewart *et al.*, 2007]. Please note that we use this scenario in our trace-driven experiment in Section 6.2.

Our Contributions. In this paper, we investigate the multi-commodity ski-rental problem. Given a commodity set with multiple commodities and the corresponding renting/buying costs, our objective is to minimize the overall competitive ratio. To address this issue, we propose Multi-Object Break-Even (MOBE) algorithm with N thresholds of cost corresponding to the N commodities. When the accumulated rental cost of a commodity reaches its threshold, the algorithm buys this commodity. Otherwise, the algorithm rents it to cover the usage. We prove an $\frac{e}{e-1}$ tight lower bound for the competitive ratio of MOBE, and further show that this bound is optimal when all the commodities share the same rent-to-buy ratio. In addition, we propose an $O(N^2)$ algorithm to obtain the N thresholds for the MOBE algorithm. We prove that the calculated thresholds lead to the optimal competitive ratio with the same rent-to-buy ratio commodities. Finally, we perform a trace-driven experiment to verify our theoretical analysis and compare the performance in a real-world scenario with benchmarks.

2 Related Work

The ski-rental problem [Karlin *et al.*, 2003] deals with the trade-off between "renting" or "buying" without the knowledge of the future. It is a typical online scheme addressed by minimizing the competitive ratio. Recent works have investigated many variants of the classical ski-rental problem. [Fujiwara *et al.*, 2020; Hu and Xu, 2017] studied the multi-slope ski-rental problem where the customer was provided with multiple extra payment options that charged both a one-shot up-front fee and per-use fees. [Ai *et al.*, 2014; Wang *et al.*, 2020] studied the multi-shop ski-rental problem where different shops had different rent/buy prices. [Patt-Shamir and Yadai, 2020] analyzed the non-linear ski-rental problem where the pay-per-use cost is a continuous monotonic function of time. [Zhang and Conitzer, 2020] considered the system with multiple resources and the buying cost was a submodular function of the complementary set of what had been brought. [Wu *et al.*, 2021] proposed the two-level ski-rental problem with multiple commodities. In addition to renting or buying a single commodity, all commodities could be brought together as a "combo purchase". However, when the "combo purchase" is disabled, their work is reduced to N independent classical ski-rentals for N commodities, which is essentially different from ours. [Fleischer, 2001] studied the Bahncard problem where the customer can buy a Bahncard to receive a discount on the rental cost.

The ski-rental problem is widely applied in real-world ICT systems where many problems can be modeled as the "rent or buy" predicament. [Wang *et al.*, 2015] investigate the problem of reserving homogeneous infrastructure as a service (IaaS) instances. At each time, the user may request multiple instances, and the decision-maker must decide to rent or buy enough instances to satisfy the request. [Khanafer *et al.*, 2013] studied the trade-off in web applications. For each query, a web application can choose whether to save (buy) the response in the cache, so as to avoid repeat computation (rent) in the future. [Saha *et al.*, 2018] applied the ski-rental problem to the wireless channels leasing problem. The operator has to choose between better but paid channel quality and degraded channel quality with no charges without knowing the customer's demands and the channel availability. Their work proves that the competitive ratio's upper bound is strictly less than 4. Different from all the above works, this paper utilizes the characteristic that the customer uses multiple commodities altogether as a novel extension of the classical ski-rental problem.

3 Problem Formulation

3.1 System Overview

The system is formulated by a commodity set $\mathbf{S}$ consisting of multiple commodities, which a customer will use unpredictably. Whenever the commodity set is used, all commodities $s_i \in \mathbf{S},\ i = 1, 2, \ldots, |\mathbf{S}|$ are used once, and the decision-maker (customer) needs to pay for the use of them all. There are two payment options for each usage of commodity s_i: (1) Renting: the user pays d_i for the on-demand usage. We assume that every time of use incurs the same cost d_i for s_i. (2) Buying: the customer immediately pays c_i, and the commodity will not incur any further payment. Let the set of all possible commodity sets as $\mathbb{S}$. In general, the commodities $s_1, s_2, \ldots, s_{|\mathbf{S}|}$ in $\mathbf{S} \in \mathbb{S}$ can be either homogeneous ($d_i = d_{i'}$ and $c_i = c_{i'}$, $\forall i, i' \in \{1, 2, \ldots, |\mathbf{S}|\}$) or heterogeneous (not homogeneous).

Whenever the commodity set is used, the customer must choose between the two payment options to cover the usage cost of each commodity in $\mathbf{S}$. Let $\delta_i(j)$ and $\gamma_i(j)$ be the 0-1 decision on commodity s_i made by the customer at the j^{th} use of the commodity set, where

$$\delta_i(j) = \begin{cases} 1, & \text{if the customer pays by renting,} \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

$$\gamma_i(j) = \begin{cases} 1, & \text{if the customer buys } s_i, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

For any given system, the input can be defined by the total times of use of the commodity set, so the input space is $\mathbb{Z}^+ = \{1, 2, 3, \ldots\}$. For any feasible input $x \in \mathbb{Z}^+$, the cost of any

given algorithm ALG is defined as the following:

$$ALG(x, \mathbf{S}) = \sum_{j=1}^{x} \sum_{i=1}^{|\mathbf{S}|} (\delta_i(j) d_i + \gamma_i(j) c_i). \tag{3}$$

Please note that for the j^{th} use, it must be covered either by renting of the j^{th} use or buying on or before the j^{th} use.

3.2 Online Environment

In this paper, we assume no prior usage statistics. We apply an adaptive adversary who knows the algorithm taken by the customer, and it can decide whether there will be another use of the commodity set according to history decisions. It is the most general type of adversaries [Boyar *et al.*, 2020].

Conventionally, in this environment, the performance of the customer's commodities acquisition algorithm is measured by the competitive ratio [Karlin *et al.*, 2003]. To introduce this concept, we need to first introduce the offline optimal algorithm which serves as the performance benchmark. For each commodity, the offline optimal algorithm either buys it at the very beginning or rents it forever. Suppose x is the total number of uses chosen by the adversary and is also known by the offline optimal algorithm, the optimal algorithm can simply make decisions by taking the minimum between the renting cost and the buying cost. Formally, let $OPT(x, \mathbf{S})$ be the cost of the offline optimal algorithm that knows x in advance, then we have

$$OPT(x, \mathbf{S}) = \sum_{i=1}^{|\mathbf{S}|} \min(x \cdot d_i, c_i). \tag{4}$$

The notion of competition ratio is how close the performance of the online algorithm is to the performance of the offline optimal algorithm. As defined by (5), the competitive ratio is the maximum ratio between the online algorithm's cost and the offline optimal algorithm's cost among all possible inputs, which serves as the "worst" case performance guarantee. Since the online algorithm could not perform better than the offline optimal algorithm, $R^*(\mathbf{S}) \geq 1$ for all $\mathbf{S}$. The smaller the competitive ratio, the better the online algorithm.

$$R^*(\mathbf{S}) = \arg\max_{x \in \mathbb{Z}^+} \frac{ALG(x, \mathbf{S})}{OPT(x, \mathbf{S})}. \tag{5}$$

Please note that if we consider all commodities in $\mathbf{S}$ as a whole with renting cost $d = \sum_{s_i \in \mathbf{S}} d_i$ and buying cost $c = \sum_{s_i \in \mathbf{S}} c_i$, the problem can be modeled by the classical ski-rental problem [Karlin *et al.*, 2003]. It has been proved that the best possible competitive ratio for the classical ski-rental problem against the adaptive adversary approaches to 2. However, this method shrinks the solution space and ignores that the customer utilizes commodities altogether. On the contrary, we allow buying some commodities first while buying others later. In this case, the solution space is larger, and thus we have a chance to obtain a better competitive ratio.

4 MOBE Algorithm

4.1 Preliminaries

Suppose it is the J^{th} time to use the commodity set. We define the typical cost for the commodity s_i as the following:

$$\sigma_i = \sum_{j \in [1, J]} \delta_i(j) d_i. \tag{6}$$

σ_i stands for the rental cost on commodity s_i until now (the J^{th} usage). For the current usage, we let $\delta_i(j) = 1$ to calculate the typical cost even though the algorithm has not made any decision.

Let $\Theta = \{\theta_1, \theta_2, \ldots, \theta_{|\mathbf{S}|}\}$ be the thresholds for commodities in $\mathbf{S}$ where

$$\theta_i \geq 0, \ \forall \theta_i \in \Theta. \tag{7}$$

Whenever the typical cost of commodity s_i exceeds θ_i, we buy commodity s_i. At that time, the algorithm invests c_i without incurring any further rental cost on s_i.

4.2 MOBE Algorithm

Algorithm 1 defines MOBE algorithm formally. The algorithm is awakened when the commodity set is used (Line 1). For each commodity, if it has been brought, then there is no further decision needed (Line 3). Otherwise, the algorithm computes the typical cost for the commodity (Line 4). If the typical cost σ_i of commodity s_i reaches θ_i, the algorithm buys s_i (Lines 5). Otherwise, when the typical cost is smaller than the threshold, the algorithm covers the payment by renting (Line 6).

MOBE algorithm buys a commodity if it has incurred enough costs by renting, which is defined by the corresponding threshold and measured by the typical cost. The performance of MOBE algorithm relies heavily on the thresholds. Our designed thresholds for MOBE algorithm are calculated by Algorithm 2. Please note that Algorithm 2 further calls Algorithm 3 to search for subsets of thresholds by binary search.

4.3 Discussion on Threshold Solver and Sub-Threshold Solver

To solve the thresholds for commodities, we need to consider two characterizing dimensions: (1) rent-to-buy ratio; (2) buying cost. The rent-to-buy ratio measures the relative price

Algorithm 1: Multi-Object Break-Even (MOBE)

input: $\mathbf{S} = \{s_1, s_2, \ldots, s_N\}$, $\Theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$ (To be computed by `Threshold_Solver(`$\mathbf{S}$`)`)

```
1 if the commodity set S is used then
2     for i ← 1 to N do
3         if s_i has not been brought then
4             compute σ_i;
5             if σ_i ≥ θ_i then buy commodity s_i;
6             else rent commodity s_i;
```

between renting and buying. First, the rent-to-buy ratio influences the decision. For example, we are not urgent to buy a \$100 commodity if the rental price is only \$1. However, we may have to be decisive if the rental price is raised up to \$80 because the cost of renting twice is more expensive than buying. Second, the buying cost influences the decision in another way. Suppose we have two commodities with the same rent-to-buy ratio, and the buying costs are \$1 and \$100. We tend to buy the cheaper one before the more expensive one because once we buy a commodity, the buying cost becomes our sunk cost, and we are more conservative to a larger sunk cost.

Algorithm 2 solves the thresholds for all $\mathbf{S} \in \mathbb{S}$. Following the above observation, Algorithm 2 first separates $\mathbf{S}$ into mutually exclusive subsets $\mathbf{S}^j\ \forall j = 1, 2, \ldots, T$ by descending order of rent-to-buy ratio in Lines 1–2. Then, within each $\mathbf{S}^j$, Algorithm 2 sorts commodities by the ascending order of the buying cost in Line 4.

In Line 7, we call Algorithm 3 to solve thresholds for commodities in each $\mathbf{S}^j$. Please note that all the thresholds of commodities are computed exactly once since $\bigcup_{j=1,2,\ldots,T} \mathbf{S}^j = \mathbf{S}$ and $\mathbf{S}^i \cap \mathbf{S}^j = \phi\ \forall i \neq j$.

Define $\mathbb{S}_r \subset \mathbb{S}$ by

$$\mathbb{S}_r = \left\{ \mathbf{S} \middle| \mathbf{S} \in \mathbb{S} \text{ and } \frac{d_i}{c_i} = \frac{d_j}{c_j},\ \forall s_i, s_j \in \mathbf{S} \right\}. \tag{8}$$

$\mathbb{S}_r$ is the set of commodity sets where all commodities have the same rent-to-buy ratio. Please note that $\mathbf{S}^j \in \mathbb{S}_r,\ \forall j = 1, 2, \ldots, T$.

Algorithm 3 is designed to solve sub-problems produced by Algorithm 2. The input is $\forall \mathbf{S} \in \mathbb{S}_r$, and it computes thresholds by performing a binary search on θ_1. First, it initializes the maximum and minimum bounds of θ_1 (Lines 1–2). In the loop, it iteratively computes $\theta_2, \theta_3, \ldots, \theta_N$ according to the following formula in Line 6:

$$\theta_i = \frac{d_i}{d_{i-1}} \left(\frac{c_i}{\sum_{q=1}^{i-1} \theta_q + \sum_{p=1}^{i-1} c_p} + 1 \right) \theta_{i-1}. \tag{9}$$

Algorithm 2: Threshold_Solver

input : $\mathbf{S} = \{s_1, s_2, \ldots, s_N\}$
output: $\Theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$

1 $r_1, r_2, \ldots, r_T$ = *all T different values of $\frac{d_i}{c_i}$ in descending order* $\forall s_i \in \mathbf{S}$;
2 $\mathbf{S}^j = \left\{ s_i \middle| \frac{d_i}{c_i} = r_j,\ \forall s_i \in \mathbf{S} \right\}\ \forall j = 1, 2, \ldots, T$;
3 **for** $j \leftarrow 1$ **to** T **do**
4 *sort $\mathbf{S}^j$ in ascending order of buying cost*;
5 *denote $\mathbf{S}^j$ as $\left\{ s_1^j, s_2^j, \ldots, s_M^j \right\}$*;
6 *denote* `id(j,i)` *as the index of s_i^j in set $\mathbf{S}$, i.e.* $s_i^j = s_{\texttt{id}(j,i)}$;
7 $\theta_{\texttt{id}(j,1)}, \theta_{\texttt{id}(j,2)}, \ldots, \theta_{\texttt{id}(j,M)}$ = `Sub_Threshold_Solver(`$\mathbf{S}^j$`)`;
8 **return** $\Theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$

Algorithm 3: Sub_Threshold_Solver

input : $\mathbf{S} = \{s_1, s_2, \ldots, s_N\}$
output: $\Theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$

1 $\theta_{1,\max} = c_1$;
2 $\theta_{1,\min} = 0$;
3 **while** *true* **do**
4 $\theta_1 = (\theta_{1,\max} + \theta_{1,\min})/2$;
5 **for** $i \leftarrow 2$ **to** N **do**
6 *compute θ_i according to (9)*;
7 **if** $|\theta_N - c_N| < \epsilon$ **then** break;
8 **else if** $\theta_N > c_N$ **then** $\theta_{1,\max} = \theta_1$;
9 **else** $\theta_{1,\min} = \theta_1$;
10 **return** $\Theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$

If θ_N is close enough to c_N, we stop the loop and return the results (Line 7). Otherwise, we update the upper/lower bound (Lines 8–9), and then update θ_1 (Line 4) in the next iteration. Please note that the termination condition in Line 7 implies that thresholds are optimal when $\theta_N = c_N$.

5 Performance Analysis

As a specification of (3), we define $ALG(x, \mathbf{S}, \Theta)$ as the cost of MOBE given input $x \in \mathbb{Z}^+$ on the commodity set $\mathbf{S} \in \mathbb{S}$ with threshold Θ, and

$$R^*(\mathbf{S}, \Theta) = \arg\max_{x \in \mathbb{Z}^+} \frac{ALG(x, \mathbf{S}, \Theta)}{OPT(x, \mathbf{S})} \tag{10}$$

as the competitive ratio of MOBE on the commodity set $\mathbf{S}$ with threshold Θ.

For any given input $\mathbf{S} \in \mathbb{S}$ and the corresponding threshold Θ, we are interested in the competitive ratio $R^*(\mathbf{S}, \Theta)$. We have the following theorem to compute it numerically.

Theorem 1. *Algorithm 2 solves the numerical competitive ratio for all $\mathbf{S} \in \mathbb{S}$ for MOBE algorithm.*

Given Theorem 1, we can compute the competitive ratio before the actual operation, so that we can have a clear expectation of how MOBE performs.

Beyond the numerical competitive ratio of a specific $\mathbf{S}$, we can bound the competitive ratio for all possible commodity sets to develop an explicit frame of what performance we can expect from MOBE algorithm. To show this bound, we have the following two theorems:

Theorem 2. *The competitive ratio of MOBE algorithm is bounded by $\frac{e}{e-1}$ and 2, i.e. for all commodity sets $\mathbf{S} \in \mathbb{S}$ we have* $\arg\min_\Theta R^*(\mathbf{S}, \Theta) \in \left[\frac{e}{e-1}, 2 \right]$.

One step further, we also have Theorem 3 to show that this bound is tight.

Theorem 3. $\left[\frac{e}{e-1}, 2 \right]$ *is the tight bound of MOBE algorithm.*

It is neutral to see 2 serves as the upper bound, because $|\mathbf{S}| = 1$ is the special case when it is reduced to the classical ski-rental problem and 2 is reached. $\frac{e}{e-1} \approx 1.582$ is a neat lower bound of the competitive ratio. This bound is

reached when $\mathbf{S}$ contains a large number of homogeneous commodities (same d_i and same c_i for all s_i). When $|\mathbf{S}| = 10, 20, 50$, and 100 (homogeneous commodities), numerical results show the competitive ratios are $1.631, 1.606, 1.592$, and 1.587, respectively, approaching $\frac{e}{e-1}$.

In many real-world cases, the commodities in the commodity set have the same rent-to-buy ratio. For example, rent-to-buy ratios of AWS saving plans for nano, small, medium, and large EC2 are the same [AWS, 2021c]. In that case, there is a need to further investigate the performance of MOBE given the set of commodities with the same rent-to-buy ratio. First, we prove that for all deterministic online algorithms, $\frac{e}{e-1}$ is the lowest possible competitive ratio by Theorem 4.

Theorem 4. *$\forall \mathbf{S} \in \mathbb{S}_r$ (same rent-to-buy ratio), $\frac{e}{e-1}$ is the optimal competitive ratio lower bound for any deterministic online algorithm.*

Further, we have Theorem 5 to show that the thresholds calculated by Algorithm 2 are optimal for all $\mathbf{S} \in \mathbb{S}_r$.

Theorem 5. *$\forall \mathbf{S} \in \mathbb{S}_r$ Algorithm 2 returns optimal thresholds.*

Theorem 5 means that with the special case of $\mathbf{S} \in \mathbb{S}_r$, our algorithm is the best possible solution, and MOBE algorithm is the optimal algorithm.

In addition to the above performance bounds, an ideal algorithm should solve the problem efficiently. However, solving the competitive ratio can be very costly because the solution space grows exponentially with the number of commodities. We have the following theorem which ensures that Algorithm 2 works in polynomial time.

Theorem 6. *Algorithm 2 operates in $O(|\mathbf{S}|^2)$.*

6 Evaluation

In this section, we first verify the theoretical performance of MOBE algorithm. Then we conduct a trace-driven experiment to compare the performance with benchmarks.

6.1 Competitive Ratio Verification

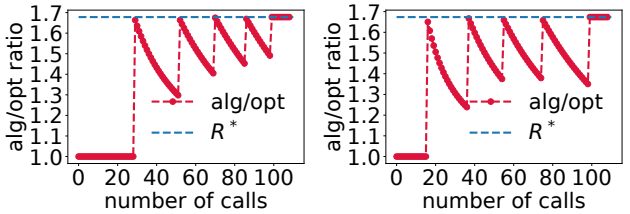
Define $\mathbb{S}_h \subset \mathbb{S}$ as the set of all commodity sets with homogeneous commodities, i.e. $\forall \mathbf{S} \in \mathbb{S}_h$ we have $d_i = d_j$, and $c_i = c_j$, $\forall s_i, s_j \in \mathbf{S}$.

We show how (9) minimizes the competitive ratio for $\mathbf{S} \in \mathbb{S}_h$ and $\mathbf{S} \in \mathbb{S}_r$ by Fig. 1. In Fig. 1-(a), we adopt 5 homogeneous commodities with $d = 1$ and $c = 100$, and in Fig. 1-(b), we adopt commodities with renting costs $1, 1.5, 1.5, 2, 3$; and $\frac{1}{100}$ as the rent-to-buy ratio. In both figures, we can see five peaks in the ratio between the MOBE's cost and the offline optimal algorithm's cost, which are caused by buying commodities as the number of uses grows. (9) ensures all the peaks have the same online/offline cost ratio, so it minimizes the maximum ratio between MOBE algorithm's cost and the optimal cost.

Fig. 2 verifies the numerical competitive ratio for non-uniform rent-to-buy ratio commodities. We demonstrate three groups of experiments by sub-figures (a), (b), and (c), and each group contains four rounds of experiments with different buying costs and rent-to-buy ratios. The yellow bar shows the numerical competitive ratio for MOBE algorithm, and the blue dots show experimental competitive ratios given *random inputs*. Each yellow bar (one round) contains 50 blue dots (experimental ratios). The detailed experiment parameters are shown in Table 1.

Group	Renting Cost	Round	Rent-to-buy Ratio
1	1,1,1,1,1	A	$\frac{1}{80}, \frac{1}{80}, \frac{1}{80}, \frac{1}{80}, \frac{1}{100}$
		B	$\frac{1}{80}, \frac{1}{80}, \frac{1}{80}, \frac{1}{100}, \frac{1}{100}$
		C	$\frac{1}{80}, \frac{1}{80}, \frac{1}{100}, \frac{1}{100}, \frac{1}{100}$
		D	$\frac{1}{80}, \frac{1}{100}, \frac{1}{100}, \frac{1}{100}, \frac{1}{100}$
2	1,1,1,1,1	E	$\frac{1}{50}, \frac{1}{50}, \frac{1}{50}, \frac{1}{50}, \frac{1}{100}$
		F	$\frac{1}{50}, \frac{1}{50}, \frac{1}{50}, \frac{1}{100}, \frac{1}{100}$
		G	$\frac{1}{50}, \frac{1}{50}, \frac{1}{100}, \frac{1}{100}, \frac{1}{100}$
		H	$\frac{1}{50}, \frac{1}{100}, \frac{1}{100}, \frac{1}{100}, \frac{1}{100}$
3	1,2,3,3,3	I	$\frac{1}{80}, \frac{1}{80}, \frac{1}{80}, \frac{1}{80}, \frac{1}{100}$
		J	$\frac{1}{80}, \frac{1}{80}, \frac{1}{80}, \frac{1}{100}, \frac{1}{100}$
		K	$\frac{1}{80}, \frac{1}{80}, \frac{1}{100}, \frac{1}{100}, \frac{1}{100}$
		L	$\frac{1}{80}, \frac{1}{100}, \frac{1}{100}, \frac{1}{100}, \frac{1}{100}$

Table 1: Experiment parameters for Figure 2



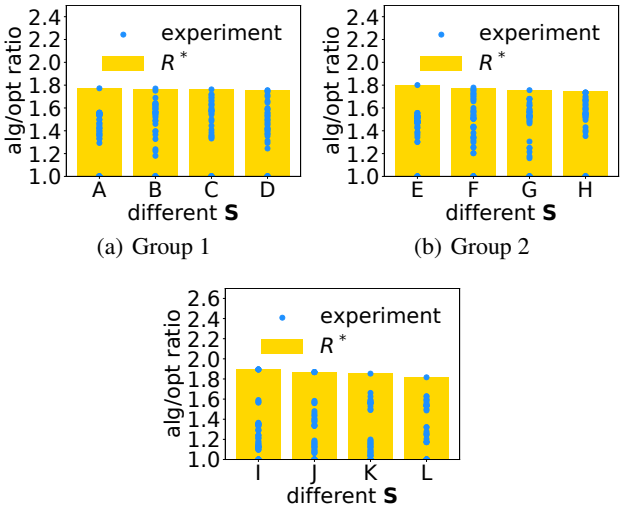
(a) Homogeneous service components (b) Same rent-to-buy ratio service components

Figure 1: ALG/OPT ratio with different number of uses

As shown in Fig. 2, all the experimental ratios are less or equal to the numerical competitive ratio. All the groups have the experimental ratio equal to the numerical competitive ratio, which shows that the numerical competitive ratio can be reached and the bound is tight. Comparing Group 1 and Group 2, in the case of the same renting price, the more the rent-to-buy ratio tends to be the same, the better the competitive ratio. Comparing Group 1 with Group 3, when the rent-to-buy ratios of the two groups are equal, the more the renting costs within the group tend to be equal, the better the competitive ratio.

6.2 Trace-driven Experiment

In this subsection, we adopt our results in the cloud service acquisition problem with multiple service components (i.e., commodities) with a real-world data set [Kamal, 2019], as described by example 2 in the introduction. Our commodity set implements a flexible fire alarm web application. It consists of service components that aim to provide fire information when a forest fire happens. The web service utilizes the AWS Fargate and AWS SageMaker notebook xlarge serverless technologies. We set three typical configurations for Fargate instances: (1) web server: 8 vCPU and 16 GB memory; (2)



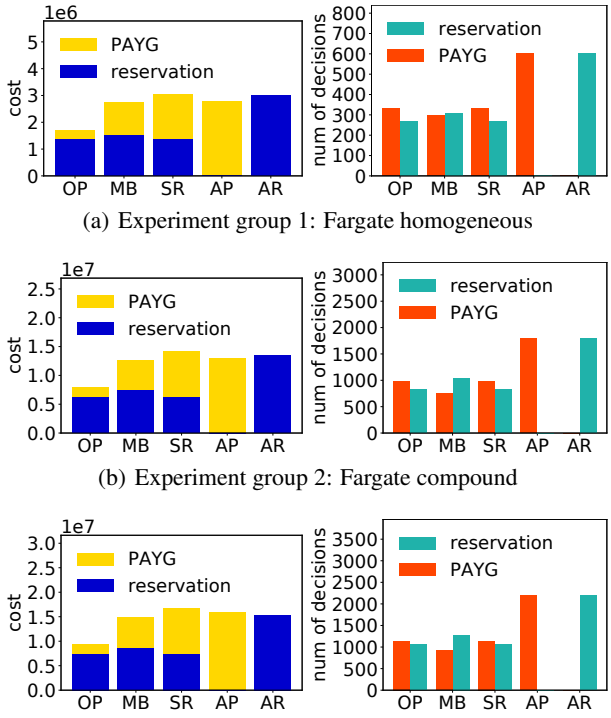
(a) Group 1 (b) Group 2

(c) Group 3

Figure 2: R^* of different rent-to-buy ratio service components

Component	PAYG/hour (rent)	Reservation (buy)
Web Server	$0.395	$4982
Micro Service	$0.790	$9964
Middle-Ware	$0.466	$5879
AWS SageMaker	$0.408	$4699

Table 2: Service Components Price



(a) Experiment group 1: Fargate homogeneous

(b) Experiment group 2: Fargate compound

(c) Experiment group 3: Fargate + Sagemaker

Figure 3: Performance comparison

micro service server: 16 vCPU and 32 GB memory; (3) middle ware (e.g. zookeeper, kafka) 8 vCPU and 32 GB memory. The pricing information is given in Table 2 [AWS, 2021b; AWS, 2021a].

We added up the number of forest fires from 2008 to 2011 in the data set and obtained the number of forest fires in each state in 3 years as the input. We assume that our flexible fire alarm system needs to be on duty for 8 hours every time a forest fire occurs.

We conduct the experiment in three groups. Group 1 uses six web servers to simulate a small web application, Group 2 consists of 4 web servers, 8 micro services, and 6 middlewares to emulate a large web application, and Group 3 adds 4 SageMaker notebooks for fire forecasting upon Group 2. In each experiment, we randomly select a state's three-year total number of forest fires from the data set. The results are shown in Fig. 3. We evaluate the following algorithms and benchmarks. (1) *OP*: the offline optimal algorithm; (2) *MB*: MOBE algorithm; (3) *SR*: the classical ski-rental algorithm for each service component; (4) *AP*: always choose PAYG (rent); (5) *AR*: always reserve (buy).

The experiment is repeated 100 times for each group, and we add up the total PAYG costs, the total reservation costs, the total number of reserved service components, and the total number of service components that are not reserved.

In all three groups, the accumulated cost of MOBE algorithm is smaller than all online benchmarks. Compared to the classical ski-rental algorithm, MOBE algorithm may reserve a service component when the accumulated PAYG cost is less than its reservation cost, and the performance is improved by this early reservation. The number of service components reserved by the classic ski-rental algorithm and the number reserved by the offline optimal algorithm are always the same. Compared with both of them, MOBE algorithm usually reserves more service components. So the early reservation also brings risks of over-reservation. The power of MOBE algorithm is the ability to balance the under-reservation and the over-reservation. In sum, MOBE algorithm outperforms the benchmarks in the average performance.

7 Conclusion

In this paper, we investigate the multi-commodity ski-rental problem to resolve the dilemma of renting or buying a set of commodities. We address this problem with competitive analysis and extend the results of the classical ski-rental problem. We propose MOBE algorithm to compute the numerical competitive ratio, and show that the tight lower and upper bounds of MOBE algorithm's competitive ratios are $\frac{e}{e-1}$ and 2 respectively. This bound neatly extends the 2 competitive ratio given by the classical ski-rental algorithm. Furthermore, we prove that the competitive ratio is minimized and MOBE is the optimal online algorithm when $\mathbf{S} \in \mathbb{S}_r$ (same rent-to-buy ratio). The MOBE algorithm is also efficient as the threshold values can be derived in $O(|\mathbf{S}|^2)$. Finally, we perform a trace-driven experiment to verify the theoretical competitive ratio and the results show that MOBE algorithm also outperforms benchmarks in terms of the average performance.

References

[Ai *et al.*, 2014] L. Ai, X. Wu, L. Huang, L. Huang, P. Tang, and J. Li. The multi-shop ski rental problem. *SIGMETRICS 2014 - Proceedings of the 2014 ACM SIGMETRICS International ARTICLE on Measurement and Modeling of Computer Systems*, pages 463–475, 2014.

[AWS, 2021a] AWS. Amazon sagemaker pricing. https://aws.amazon.com/sagemaker/pricing/, 2021. Accessed: 2021-10-10.

[AWS, 2021b] AWS. Aws fargate pricing. https://aws.amazon.com/fargate/pricing/?nc=sn&loc=2, 2021. Accessed: 2021-10-10.

[AWS, 2021c] AWS. Compute savings plans. https://aws.amazon.com/savingsplans/compute-pricing/?nc1=h_ls, 2021. Accessed: 2021-10-10.

[Boyar *et al.*, 2020] J. Boyar, F. Ellen, and K.S. Larsen. Randomized distributed online algorithms against adaptive offline adversaries. *Information Processing Letters*, 161, 2020.

[Fleischer, 2001] R. Fleischer. On the bahncard problem. *Theoretical Computer Science*, 268(1):161–174, 2001.

[Fujiwara *et al.*, 2020] H. Fujiwara, K. Shibusawa, K. Yamamoto, and H. Yamamoto. Bounds for the multislope ski-rental problem. *IEICE Transactions on Information and Systems*, E103D(3):481–488, 2020.

[Guo *et al.*, 2019] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and Y. Bao. Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces. *Proceedings of the International Symposium on Quality of Service, IWQoS 2019*, 2019.

[Hu and Xu, 2017] M. Hu and W. Xu. A better bound of randomized algorithms for the multislope ski-rental problem. *RAIRO - Theoretical Informatics and Applications*, 51(2):91–98, 2017.

[Kamal, 2019] Sani Kamal. Forest fires in india. https://www.kaggle.com/sanikamal/forest-fires-in-india, 2019. Accessed: 2021-09-19.

[Karlin *et al.*, 2003] A.R. Karlin, C. Kenyon, and D. Randall. Dynamic tcp acknowledgement and other stories about e/(e - 1). *Algorithmica (New York)*, 36(3):209–224, 2003.

[Khanafer *et al.*, 2013] A. Khanafer, M. Kodialam, and K.P.N. Puttaswamy. The constrained ski-rental problem and its application to online cloud cost optimization. *Proceedings - IEEE INFOCOM*, pages 1492–1500, 2013.

[Patt-Shamir and Yadai, 2020] Boaz Patt-Shamir and Evyatar Yadai. Non-linear ski rental. *Annual ACM Symposium on Parallelism in Algorithms and Architectures*, page 431 – 440, 2020.

[Saha *et al.*, 2018] G. Saha, A.A. Abouzeid, and M. Matinmikko-Blue. Online algorithm for leasing wireless channels in a three-tier spectrum sharing framework. *IEEE/ACM Transactions on Networking*, 26(6):2623–2636, 2018.

[Stewart *et al.*, 2007] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. *Operating Systems Review (ACM)*, pages 31–44, 2007.

[Wang *et al.*, 2015] W. Wang, B. Liang, and B. Li. Optimal online multi-instance acquisition in iaas clouds. *IEEE Transactions on Parallel and Distributed Systems*, 26(12):3407–3419, 2015.

[Wang *et al.*, 2020] S. Wang, J. Li, and S. Wang. Online algorithms for multi-shop ski rental with machine learned advice. *Advances in Neural Information Processing Systems*, 2020-December, 2020.

[Wu *et al.*, 2021] Binghan Wu, Wei Bao, and Dong Yuan. Competitive analysis for two-level ski-rental problem. *Proceedings of the AAAI ARTICLE on Artificial Intelligence*, 35(13):12034–12041, May 2021.

[Zhang and Conitzer, 2020] H. Zhang and V. Conitzer. Combinatorial ski rental and online bipartite matching. *EC 2020 - Proceedings of the 21st ACM ARTICLE on Economics and Computation*, pages 879–910, 2020.