Competitive Algorithms for an Online Rent or Buy Problem with Variable Demand

Rohan Kodialam

High Technology High School, Lincroft, NJ | rkodialam@ctemc.org

1 Abstract

We consider a generalization of the classical Ski Rental Problem motivated by applications in cloud computing. We develop deterministic and probabilistic online algorithms for rent/buy decision problems with time-varying demand. We show that these algorithms have competitive ratios of 2 and 1.582 respectively. We also further establish the optimality of these algorithms.

2 Introduction

The Ski Rental problem is the canonical example of a class of online rent or buy problems [9]. In the traditional ski rental problem, a man visits a ski resort, not knowing how many days he will be able to ski. The man can either rent skis at a cost of r dollars per day, or buy the skis permanently for b dollars. The dilemma arises from the fact that the number of days available for skiing is not known in advance; if there are many days of skiing it is better to buy the skis to avoid paying the rental fee every day, but if there are only a few days available for skiing it may be more economical to rent the skis instead. Applications of the ski-rental problem include scheduling tasks on a computer (the system must decide if it should do a task immediately by paying a high cost in processing time, or if it should wait and pay a 'rental' cost for keeping the task in waiting)[7] and caching data (the system decides if it should read a block of data during every pass at a cost of 1 bus cycle, or if it should pass over the data block and use several bus cycles to access the data if it is needed later) [5].

Several extensions of the Ski Rental problem have already been studied, including variants where the player can switch bewteen two renting options at a cost [3], and more complex scenarios where there are more than two options to choose between [6]. In these cases, an e-competitive algorithm can be found. A natural extension of these situations - where there are multiple renting and multiple buying options - has also been studied [1]. Algorithms have also been developed to provide a strategy when the rental cost r is free to vary with time [2]. Another more complex extension studied is how to allocate capacity available through renting or buying to a graph's edges to allow for sufficient flow between sources and sinks [4]. These variants add complexity to the ski rental problem, and as a result the competitive ratio varies as a function of the new parameters introduced by each variation.

We consider a generalization of the ski-rental problem where demand varies across time. The motivation for solving this problem is the idea of *cloud bursting* that is used to address the computing needs for an enterprise [10]. The amount of computing that the enterprise requires (defined in terms of number of computer servers) varies across time. The enterprise can meet this requirement by buying these servers and creating a private network or can rent servers temporarily from a cloud service provider (public cloud) like Amazon. This idea of meeting computing requirement using a combination of internal and external cloud resources is called cloud bursting. The problem that the enterprise has to solve is to decide when and how many servers to buy and how many to rent from the cloud. Typically the computing resource requirement can vary significantly over time. It would be prohibitively expensive to handle the peak loads by buying servers. Similarly, it may not be economical to rent capacity from the public cloud to handle the entire computing requirement. Since the computing requirement will, in general, not be known ahead of time, the buy or rent decision has to be made in an online manner. There are several practical considerations that go into solving the problem in the real world but the model that we consider in this paper abstracts an important aspect of the buy/rent decision. Moreover, this abstraction models other online rent or buy decision problems with variable demand. Note that if the demand is one unit for a set of consecutive intervals and then becomes zero then it models the standard ski-rental problem.

3 Problem Definition

We now formally define the problem that we study in this paper. Consider demands for some good that arrives to a system. We assume that demands arrive once per time period. For simplicity, we call each time interval a day. The demand for day i will be represented by d_i, with $d_i \geq 0$. The demand can be satisfied in two ways. The user can *rent* an item at a cost of r per day, and this item will satisfy one unit of demand for one day. The user can also *buy* an item at a cost of $b \geq r$, in which case the item will satisfy one unit of demand for day i and all subsequent days. The system objective is to determine the buy and rent options to satisfy the demand at a minimal cost. There are two versions of the problem that can be considered.

- **Offline Problem:** In the offline version of the problem, all of the demands are known to the user ahead of time, and the cost optimization is therefore performed with complete knowledge of all the demands at the beginning.
- **Online Problem:** In the online version of the problem, only the values of b and r are known initially, but each demand d_i is given to the user on day i. The online user also does not know the number of time periods n during which demands will arrive. The user must perform the cost optimization without any knowledge of future demands.

The optimal offline cost provides a lower bound on the cost achieved by any online algorithm since the offline algorithm has more knowledge than the online algorithm. In this paper, we are interested in deriving algorithms to solve the online problem and measuring their performance.

3.1 Measuring the Performance of Online Algorithms

We measure the performance of an online algorithm by comparing the cost incurred by that algorithm to that of the cost incurred by the optimum offline algorithm. The optimal cost will be a function of the input vector of demands. The demand for n days can be represented as an n-dimensional vector $\mathcal{D} = (d_1, d_2, \ldots, d_n)$. Let $C_{on}(\mathcal{D})$ represent the cost of an online algorithm and

$C_{off}(\mathcal{D})$ represent the cost of the optimum offline algorithm operating on input $\mathcal{D}$. The *competitive ratio* $\rho \geq 1$ of the online algorithm is then defined as

$$\rho = \max_{\mathcal{D}} \frac{C_{on}(\mathcal{D})}{C_{off}(\mathcal{D})}.$$

In other words, the competitive ratio is the worst case ratio of the costs of the online algorithm to the optimum offline algorithm over all demand vectors over any number of days. Note that the costs incurred by the algorithms and hence the competitive ratio will be a function of the per-day rental cost r, the cost of buying b and the demand set $\mathcal{D}$. Note that without loss of generality we can assume that the per-day rental cost $r = 1$. This can be done by rescaling b, that is, setting the value of b to b/r. For the ease of presentation, we also assume that b is integral. This is purely done for convenience and all the results in the paper can applied to the case where the rescaled value of b is not integral by setting b to $\lceil b \rceil$. We use the following notation in the analysis of the algorithms: Consider the demands for the first k days $(d_1, d_2, \ldots, d_k)$. Let π be a permutation of the set $\{1,2,\ldots,k\}$ such that $d_{\pi(1)} \geq d_{\pi(2)} \geq \ldots \geq d_{\pi(k)}$. Assume that ties are broken arbitrarily. We use h_b^k to denote the b^{th} highest of the first k demands. If $b > k$, then $h_b^k = 0$. Therefore $h_b^k = d_{\pi(b)}$. We define $S_b^k = \{\pi(1), \pi(2) \ldots, \pi(b)\}$ to denote the indices of the highest b values out of the k arrivals. We now state a simple result that is used in Section 5.

Lemma 3.1 *Let S_b^k denote the indices of the b highest demands out of the first k arrivals and h_b^k denote the b^{th} highest value out of the first k arrivals. If $d_k > h_b^{k-1}$ then*

$$\sum_{i \in S_b^k} d_i = \sum_{i \in S_b^{k-1}} d_i + d_k - h_b^{k-1}.$$

Proof Since $d_k > h_b^{k-1}$, index $k \in S_b^k$ and an index with value h_b^{k-1} is not in the set S_b^k.

We now study the offline optimization problem and derive the optimal offline cost.

4 Offline Problem

In the offline problem, all of the demands are known ahead of time. Since the value of b does not vary over time and all demands are known initially, we can assume that the optimum offline solution buys all the needed goods initially. In other words, if an optimum offline solution buys x_t units on some day t, then the solution value does not increase if x_t is bought initially. The cost of buying still remains bx_t for these x_t items and there is an opportunity to use these items at some day before day t and perhaps reduce the rental costs. Therefore, the offline algorithm consists of purchasing x items at the beginning and renting all other items as needed. As before, let the demands form the vector $\mathcal{D}$. The optimum offline algorithm is called $A_{off}(\mathcal{D})$ and the total cost it incurs is defined as $C_{off}(\mathcal{D})$. Let x denote the number of items bought initially and y_i, the number rented on day

i. The minimal cost can be found by setting up the following linear program:

$$\begin{aligned}\text{minimize } & bx + \sum_{i=1}^{n} y_i \\ \text{subject to } & x + y_i \geq d_i \quad i \in [1,n] \\ & x \geq 0 \\ & y_i \geq 0 \quad i \in [1,n].\end{aligned}$$

Where $[1,n]$ represents the integers between 1 and n, inclusive of both endpoints. The variables x and y_i must be integral, but since the linear programming relaxation above leads to integral optimal solutions we solve the linear problem instead of the integer program. The objective function of this linear program represents the total cost of the algorithm $C_{off}(\mathcal{D})$, which we want to minimize. The first constraint ensures that the demand is met each day. The other constraints ensure that x and y_i are non-negative. To find the optimal solution, let us take the dual of the linear program. The dual [8] is

$$\begin{aligned}\text{maximize } & \sum_{i=1}^{n} d_i z_i \\ \text{subject to } & \sum_{i=1}^{n} z_i \leq b \\ & 0 \leq z_i \leq 1, i \in [1,n].\end{aligned}$$

This problem is solved by assigning $z_i = 1$ for the b highest demands. Recall that h_b^k represents the b^{th} highest demand value among the first k arrivals. The optimal solution to the dual problem is $\sum_{i \in S_b^n} d_i$. If a primal solution can be constructed with the same objective function value as the dual, that primal solution is optimal by strong duality. To construct a primal solution with the same optimal value, the variables x and y_i are set so that

$$x = h_b^n$$

$$y_i = \begin{cases} 0 & \text{if } i \notin S_b^n \\ d_i - h_b^n & \text{if } i \in S_b^n. \end{cases}$$

This corresponds to buying h_b^n items immediately and renting any more items if $d_i > h_b^n$. The cost $C_{off}(\mathcal{D})$ is then just the optimal value of the primal objective function, so

$$C_{off}(\mathcal{D}) = \sum_{i \in S_b^n} d_i. \tag{1}$$

As this value is equal to that of the dual objective function, it is optimal by strong duality.

4.1 Offline Algorithm Example

Let us consider the example where $b = 3$, and $\mathcal{D}$ is {3,5,2,4,8,1,6}. The algorithm suggests that the user should buy $h_b^n = h_3^7$ items immediately. In this case, $h_3^7 = 5$ items should be bought. Doing so adds a cost of $5b = 15$ to the cost incurred. With regards to the renting costs, it is only necessary to rent on days i when $d_i > 5$, so no items will be rented for $i = 1,2,3,4,6$. Three items must be rented on day $i = 5$ and 1 on day $i = 7$, so a total of 4 items at a total cost of 4 must be rented. The total cost, which is the sum of buying and renting costs, is therefore $15 + 4 = 19$.

5 Deterministic Online Algorithm

We now consider the online optimization problem. In the online optimization problem, the demand is given one day at a time and the rent/buy decisions have to be made on demand arrival. Therefore, on day k, only the demands $d_1, d_2, ..., d_k$ are known. We now outline algorithm A_{on} and analyze its competitive ratio. In the description of the algorithm we use $\tilde{x}_k$ to denote the number of items bought on day k and $\tilde{y}_k$ to denote the number of items rented on day k. Recall that we use h_b^k to denote the b^{th} highest demand among the first k demands.

Algorithm A_{on}

```
for $k$=1 to $n$ do
if $k<b$ then
    $\tilde{x}_k = 0$
    $\tilde{y}_k = d_k$
else
    Compute $h_b^k$
    $\tilde{x}_k = h_b^k - h_b^{k-1}$
    $\tilde{y}_k = \max\{0, d_k - h_b^k\}$
end
```

The online algorithm A_{on} works as follows: For the first $b - 1$ days, the online algorithm rents all the demands. From each day $k \geq b$ onwards, the online algorithm first computes h_b^k. It then buys $h_b^k - h_b^{k-1}$ items. It is easy to see that the total number of items that have been bought up to (and including) day k is h_b^k. If $d_k > h_b^k$, it rents $d_k - h_b^k$ items to meet the demand d_k. In the next theorem, we derive a closed form expression for the cost incurred by the online algorithm after processing $k \geq b$ demands.

Theorem 5.1 *For any $k \geq b$, the cost of the online algorithm after k demands have been processed, denoted by C_{on}^k, is given by*

$$C_{on}^k = \sum_{i \in S_b^k} d_i + (b-1)h_b^k. \tag{2}$$

Proof We prove the theorem via induction on k when $k \geq b$. Before the arrival b, the online algorithm only rents. Therefore after processing arrival $b-1$, the total cost will be $\sum_{i=1}^{b-1} d_i$. When $k = b$, then the online algorithm buys h_b^b items each at cost b and rents $d_b - h_b^b$ incurring a total cost of $bh_b^b + d_b - h_b^b$ in day b. Therefore the total cost from day one until after arrival b is processed is given by

$$\begin{aligned}C_{on}^{b} &= \sum_{i=1}^{b-1} d_i + bh_b^b + d_b - h_b^b \\ &= \sum_{i\in S_b^b} d_i + (b-1)h_b^b.\end{aligned}$$

This establishes the base step of the induction. Let us now assume that the cost formula holds until arrival $k-1$ is processed, i.e.,

$$C_{on}^{k-1} = \sum_{i\in S_b^{k-1}} d_i + (b-1)h_b^{k-1}.$$

When arrival k is processed, the algorithm buys $h_b^k - h_b^{k-1}$ items at a cost of $b\left(h_b^k - h_b^{k-1}\right)$. It then rents (if necessary) $d_k - h_b^k$ items. We consider two separate cases:

1. If $d_k \le h_b^{k-1}$, then the demand d_k is less than the total number of items already purchased and therefore no items will be bought or rented. In this case $h_b^k = h_b^{k-1}$ and $S_b^k = S_b^{k-1}$ and therefore $C_{on}^{k-1} = C_{on}^{k} = \sum_{i\in S_b^k} d_i + (b-1)h_b^k$.

2. If $d_k > h_b^{k-1}$, then $h_b^k > h_b^{k-1}$ and $h_b^k - h_b^{k-1}$ items will be bought at a cost of $b\left(h_b^k - h_b^{k-1}\right)$. Moreover $d_k - h_b^k$ items will be rented. Therefore the total cost incurred by the online algorithm after demand k is processed is

$$\begin{aligned}C_{on}^{k} &= C_{on}^{k-1} + b\left(h_b^k - h_b^{k-1}\right) + \left(d_k - h_b^k\right) \\ &= \sum_{i\in S_b^{k-1}} d_i + (b-1)h_b^{k-1} + b\left(h_b^k - h_b^{k-1}\right) + \left(d_k - h_b^k\right) \\ &= \sum_{i\in S_b^{k-1}} d_i + (b-1)h_b^{k-1} + (b-1)h_b^k - bh_b^{k-1} + d_k \\ &= \sum_{i\in S_b^{k-1}} d_i + (b-1)h_b^k + d_k - h_b^k \\ &= \sum_{i\in S_b^{k}} d_i + (b-1)h_b^k\end{aligned}$$

where the last equality follows from Lemma 3.1.

Note that the total cost incurred by the online algorithm is independent of the order in which the demands arrive into the system. It is just a function of the ordered set of demands. We now can give the competitive ratio of the online algorithm.

Theorem 5.2 *Given any non-negative n-vector $\mathcal{D}$ of demands, let $C_{off}(\mathcal{D})$ be the cost incurred by the offline algorithm and $C_{on}(\mathcal{D})$ be the cost incurred by the online algorithm. Then*

$$\rho = \frac{C_{on}(\mathcal{D})}{C_{off}(\mathcal{D})} \le 2 - \frac{1}{b}.$$

Proof If the number of demands $n < b$, then the optimal offline solution is to rent all the demands and the online algorithm also does the same. The competitive ratio $\rho = 1$ if $n < b$. If $n \geq b$, then from Equations (1) and (2), we can write

$$\begin{aligned}\frac{C^n_{on}}{C^n_{off}} &= \frac{\sum_{i \in S^n_b} d_i + (b-1)h^n_b}{\sum_{i \in S^n_b} d_i} \\ &= 1 + \frac{(b-1)h^n_b}{\sum_{i \in S^n_b} d_i} \\ &\leq 1 + \frac{(b-1)h^n_b}{bh^n_b} \\ &= 1 + \frac{(b-1)}{b} = 2 - \frac{1}{b}.\end{aligned}$$

The inequality follows from the fact that $d_i \geq h^n_b$ for all $i \in S^n_b$. Algorithm A_{on} can also be shown to be the best possible deterministic online algorithm to solve this problem, or, equivalently, that no other deterministic online algorithm can have a competitive ratio lower than $2 - \frac{1}{b}$. This follows directly from the proof of the optimality of the standard item rental problem [9] and we give the proof for completeness.

Theorem 5.3 *No deterministic online algorithm operating with the same demands as A_{on} to create a rent/buy schedule to solve the ski-rental problem with demands can have a competitive ratio less than $2 - \frac{1}{b}$.*

Proof Let us construct a simple example for which no deterministic algorithm can have a competitive ratio lower than $2 - \frac{1}{b}$. We consider a special case of our problem where the demands are one up to some day after which the demand becomes zero. Consider an online algorithm for this problem. Assume that the online algorithm buys one unit of the item on day m. In this case the cost of the online algorithm is $m - 1 + b$ where $m - 1$ is the cost of renting for the first $m - 1$ days and b is the cost of buying on day m. Assume that the demand generator sets the demands to zero from day $m + 1$. The optimal offline cost to this problem is $\min\{m, b\}$. Therefore the competitive ratio is

$$\frac{m - 1 + b}{\min\{m, b\}} = \frac{\max\{m, b\} + \min\{m, b\} - 1}{\min\{m, b\}} \geq \frac{2b - 1}{b} = 2 - \frac{1}{b}.$$

The inequality follows from the fact that the ratio is minimized when $m = b$.

5.1 Online Deterministic Algorithm Example

Let us again consider the example where $b = 3$, and $\mathcal{D}$ is {3,5,2,4,8,1,6}. Let t represent the current day. For the first two days, while $t < b$, we will only rent to satisfy the demands. At all further days t, we will buy $\tilde{x} = h^t_3 - h^{t-1}_3$ items and rent $\tilde{y} = h^t_3 - d_t$ items, as in the table below:

t	$\tilde{x}_t$	$\tilde{y}_t$	C_{on}^t
1	0	3	3
2	0	5	8
3	2	0	14
4	1	1	18
5	1	4	25
6	0	0	25
7	1	1	29

Thus the total cost incurred by the online algorithm is 29, which is within a factor of 2 of the optimal offline cost of 19. The ratio of the online and offline costs in this example is $\frac{29}{19} \approx 1.526$.

6 Probabilistic Online Algorithm

It has been shown that the best online deterministic algorithm will incur no more than about twice the cost of an offline algorithm that knows all the demands in advance. A probabilistic online algorithm randomizes over multiple deterministic strategies can potentially achieve better expected performance. We first define how the performance of probabilistic online algorithms is measured. Assume that the algorithm has set A of deterministic strategies that it randomizes over. Assume that it uses strategy $a \in A$ with probability p_a. Since the probabilistic algorithm randomizes over multiple deterministic strategies, we have to weight the competitive ratio achieved with strategy a with the probability that strategy a is used in order to get the expected competitive ratio $\bar{\rho}$. The *expected competitive ratio* is given by

$$\bar{\rho} = \max_{\mathcal{D}} \sum_{a \in A} p_a \left[\frac{C_{on}^a(\mathcal{D})}{C_{off}(\mathcal{D})} \right].$$

Note that the performance of the probabilistic online algorithm is measured over the worst case demand input. From the worst-case analysis of the deterministic algorithm in the previous section, an obvious weakness of the deterministic algorithm is that it often buys too little. In general, the probabilistic algorithm we are about to describe will attempt to correct this by buying more items than the deterministic algorithm. In the deterministic algorithm, the user will try to buy items so that the total amount bought is equal to the b^{th} highest demand seen so far at each time interval. The probabilistic algorithm will, instead of buying only up to the b^{th} highest, will buy up to the a^{th} highest demand where the value of a is chosen randomly between 1 and b. By choosing the distribution of a carefully, the probabilistic algorithm attempts the minimize the expected competitive ratio. Let $C_{on}^a(\mathcal{D})$ represent the cost of the online algorithm which rents for the first $a-1$ demands and then chooses the a^{th} highest demand each day. On day k, the algorithm buys $h_a^k - h_a^{k-1}$ items and rents $\max\{0, d_k - h_a^k\}$ items.

Lemma 6.1 *Let $C_{on}^a(\mathcal{D})$ represent the cost of the online algorithm that buys to the a^{th} highest demand each day and rents if necessary when the demand vector is $\mathcal{D}$. Then*

$$C_{on}^a(\mathcal{D}) = \sum_{i \in S_a^n} d_i + (b-1)h_a^n. \tag{3}$$

Proof The proof follows exactly the same argument as in the last section for the deterministic online algorithm. We replace b with a keeping in mind that the cost of buying the items is still b.

6.1 Picking the Optimal Distribution

We now derive the optimal distribution of a in order to minimize the competitive guarantee. Towards this end, we first make a notational simplification in order to keep the analysis clean. Since both the offline costs as well as the online cost for any strategy does not depend on the order of demands, without loss of generality, we assume that $d_1 \geq d_2 \geq \ldots \geq d_n$. With the demands ordered in this fashion, we can now write

$$C^a_{on}(\mathcal{D}) = \sum_{i=1}^{a} d_i + (b-1)d_a. \tag{4}$$

$$C_{off}(\mathcal{D}) = \sum_{i=1}^{b} d_i. \tag{5}$$

This due to the fact that the set $S^n_a = \{1,2,\ldots,a\}$. The objective of the *algorithm designer* is to pick the probabilities $p_a \geq 0$ where $\sum_{a=1}^{b} p_a = 1$ such $\bar{\rho}$ is minimized. Assume that the algorithm designer has fixed the probability vector p_a. The adversary for the algorithm designer is the *demand generator*. For a fixed probability vector, the demand generator solves the following optimization problem:

$$\max_{\mathcal{D}} \sum_{a=1}^{b} p_a \left[\frac{C^a_{on}(\mathcal{D})}{C_{off}(\mathcal{D})}\right].$$

In other words, the demand generator attempts to find the demand input that maximizes the competitve ratio. From Equations (4) and (5), note that the competitive ratio is unchanged if all the demands are scaled. Therefore, without loss of generality, we can assume that the demand generator normalizes the demands such that

$$C_{off}(\mathcal{D}) = \sum_{a=1}^{b} d_a = 1.$$

Using the value of C^a_{on} from Equation (3),

$$\sum_a p_a C^a_{on}(\mathcal{D}) = \sum_{a=1}^{b} p_a \left[\sum_{i=1}^{a} d_i + (b-1)d_a\right] \tag{6}$$

$$= \sum_{i=1}^{b} d_i \left[b p_i + \sum_{a=i+1}^{b} p_a\right]. \tag{7}$$

Equation (7) is just a re-arrangement of the terms in Equation (6) in order to collect the terms corresponding to d_i. Therefore the optimization problem solved by the demand generator is

$$\max \sum_{a=1}^{b} d_a \left[b p_a + \sum_{j=a+1}^{b} p_j\right]$$

$$\begin{aligned} s.t. \sum_{a=1}^{b} d_a &= 1 \\ d_a &\geq 0 \ \forall a. \end{aligned}$$

Let $c_a = bp_a + \sum_{j=a+1}^{b} p_j$ be the coefficient of d_a in the objective function. The optimal solution to this problem is for the demand generator to set $d_a = 1$ corresponding to the maximum c_a. Therefore, it is in the interest of the algorithm designer to make all the coefficients c_a equal. In this case $\bar{\rho}$ will equal this (common) value of c_a. The algorithm designer, picks p_a such that

$$c_a = bp_a + \sum_{j=a+1}^{b} p_j = \bar{\rho} \;\; \forall a. \tag{8}$$

Equating the coefficients c_a and c_{a+1}, we get

$$bp_a + \sum_{i=a+1}^{n} p_i = bp_{a+1} + \sum_{i=a+2}^{b} p_i.$$

The equation can be simplified by cancelling the common terms on both sides, giving

$$bp_a + p_{a+1} = bp_{a+1},$$

which further simplifies to

$$\frac{b}{b-1} p_a = p_{a+1}.$$

This shows that the probabilities p_a are in a geometric progression with common ratio $r = \frac{b}{b-1}$. Therefore,

$$p_a = p_1 r^{a-1} \;\; a = 1,2,3, \ldots b.$$

Since $\sum_{a=1}^{b} p_a = 1$, we solve for p_1 to get

$$p_1 = \frac{r-1}{r^b - 1}$$

and

$$p_a = \frac{r-1}{r^b - 1} \, r^{a-1} \;\; , \; a = 1,2, \ldots , b. \tag{9}$$

With the distribution now known, we can fully outline the probabilistic online algorithm $A_{prob}(\mathcal{D})$, using the same notation as used in the description of algorithm A_{on} in section 5.

Algorithm A_{prob}

```
pick $\tilde{a}$ randomly using distribution (9)
for $k$=1 to $n$ do
if $k < \tilde{a}$ then
    $\tilde{x}_k = 0$
    $\tilde{y}_k = d_k$
else
    Compute $h_{\tilde{a}}^{k}$
    $\tilde{x}_k = h_{\tilde{a}}^{k} - h_{\tilde{a}}^{k-1}$
    $\tilde{y}_k = \max\{0, d_k - h_{\tilde{a}}^{k}\}$
end
```

We now establish the probabilistic expected competitive ratio. To solve for the expected competitive ratio, we use Equation (8) when $a = b$, and we get

$$\begin{aligned}
\bar{\rho} &= bp_b \\
&= b\left[\frac{r-1}{r^b-1}\right]r^{b-1} \\
&= \frac{r^b}{r^b-1} \\
&= \frac{1}{1-r^{-b}} = \frac{1}{1-\left(1+\frac{1}{b-1}\right)^{-b}} \\
&\leq \frac{1}{1-\frac{1}{e}} = \frac{e}{e-1}. \qquad (10)
\end{aligned}$$

Inequality (10) follows from the fact that

$$\left(1+\frac{1}{b-1}\right)^{-b} \leq \frac{1}{e}.$$

Therefore, we can state the following theorem about the probabilistic online algorithm.

Theorem 6.2 *The probabilistic online algorithm $A_{prob}(\mathcal{D})$ achieves an expected competitive ratio $\bar{\rho}$ of $\frac{e}{e-1}$.*

Since this problem generalizes the standard ski-rental problem, we can state the following known result which follows directly from the ski rental problem [9].

Theorem 6.3 *No probabilistic online algorithm operating with the same inputs as $A_{prob}(\mathcal{D})$ to create a rent/buy schedule to solve the ski-rental problem with demands can have an expected competitive ratio less than $\frac{e}{e-1}$.*

6.2 Probabilistic Algorithm Example

Let us again consider the example where $b = 3$, and $\mathcal{D}$ is {3,5,2,4,8,1,6}. We now have three pure strategies to use: strategy $a = 1$, strategy $a = 2$, and strategy $a = 3$. For strategy $a = 1$, we will rent if $t < 1$, and if $t \geq 1$ we will buy $\tilde{x}_t^1 = h_1^t - h_1^{t-1}$ items and rent $\tilde{y}_t^1 = h_1^t - d_t$ items, as in the table below:

t	$\tilde{x}_t^1$	$\tilde{y}_t^1$	$C_{prob}^1(t)$
1	3	0	9
2	2	0	15
3	0	0	15
4	0	0	15
5	3	0	24
6	0	0	24
7	0	0	24

For strategy $a = 2$, we will rent if $t < 2$, and if $t \geq 2$ we will buy $\tilde{x}_t^2 = h_2^t - h_2^{t-1}$ items and rent $\tilde{y}_t^2 = h_2^t - d_t$ items, as in the table below:

t	$\tilde{x}_t^2$	$\tilde{y}_t^2$	$C_{prob}^2(t)$
1	0	3	3
2	3	2	14
3	0	0	14
4	1	0	17
5	1	2	22
6	0	0	22
7	1	0	25

Finally, for strategy $a = 3$, while $t < b$, we will only rent. At times $t \geq b$, we will buy $\tilde{x}_t^3 = h_3^t - h_3^{t-1}$ items and rent $\tilde{y}_t^3 = h_3^t - d_t$ items, as in the table below:

t	$\tilde{x}_t^3$	$\tilde{y}_t^3$	$C_{prob}^3(t)$
1	0	3	3
2	0	5	8
3	2	0	14
4	1	1	18
5	1	4	25
6	0	0	25
7	1	1	29

The probability that the user chooses a strategy can be computed from Equation (9). So,

$$\begin{aligned}
p_1 &= (\frac{2}{3})^2(3(1-(1-3^{-1})^3))^{-1} \\
&\approx 0.2105 \\
p_2 &= (\frac{2}{3})^1(3(1-(1-3^{-1})^3))^{-1} \\
&\approx 0.3158 \\
p_3 &= (\frac{2}{3})^0(3(1-(1-3^{-1})^3))^{-1} \\
&\approx 0.4737
\end{aligned}$$

Now, we can find the expected cost as $p_1C_{prob}^1 + p_2C_{prob}^2 + p_3C_{prob}^3$, which is $(0.2105)(24) + (0.3158)(25) + (0.4737)(29) \approx 26.684$. We can now see that the expected cost of the probabilistic algorithm is lower than that of the deterministic algorithm, which has a cost of 29. The expected ratio of the probabilistic algorithm's cost to that of the offline algorithm in this example is $\frac{26.684}{19} \approx 1.404$.

7 Conclusion and Further Research

In this paper, we considered an online buy or rent decision problem with variable demands and we developed

- A deterministic online algorithm with a competitive ratio of $2 - \frac{1}{b}$.
- A probabilistic online algorithm with an expected competitive ratio of $\frac{e}{e-1}$.

We further established the optimality of these two algorithms.
Currently, we are studying variants of this problem under more general cost models as well as realistic constraints.

References

[1] L. AI, X. WU, LINGXIAO HUANG, LONGBO HUANG, P. TANG, and J. LI, The Multi-shop Ski Rental Problem, in Computing Research Repository, 2014

[2] M. BIENKOWSKI, Ski Rental Problem with Dynamic Pricing, Institute Of Computer Science, University Of Wroclaw, Report 03/08, 2008

[3] H. FUJIWARA, T. KITANO, and T. FUJITO, On the Best Possible Competitive Ratio for Multislope Ski Rental, in ISAAC'11 Proceedings of the 22nd international Conference on Algorithms and Computation, 2011, pp. 544-553.

[4] A. GUPTA, A. KUMAR, M. PAL, and T. ROUGHGARDEN, Approximation via cost sharing: Simpler and better approximation algorithms for network design, in Journal of the ACM (JACM), Vol 54, No. 3, 2007, pp. 11-20.

[5] A. R. KARLIN, M. S. MANASSEE, L. A. MCGEOCH, and S. OWICKI, Competitive randomized algorithms for non-uniform problems, in Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1990, pp. 301-309.

[6] Z. LOTKER, B. PATT-SHAMIR, and D. RAWITZ, Rent, Lease or Buy: Randomized Algorithms for Multislope Ski Rental, in Symposium on Theoretical Aspects of Computer Science. STACS, Bordeaux, France, 2008, pp. 503-51.

[7] S. S. SEIDEN, A guessing game and randomized online algorithms, in Proceedings of the thirty-second annual ACM symposium on Theory of computing (STOC '00). ACM, New York, NY, USA, 2000, pp. 592-601.

[8] S. LAHAIE, How to take the Dual of a Linear Program, <www.cs.columbia.edu/coms6998-3/lpprimer.pdf>, 2008

[9] M. QUEYRANNE, An Introduction to Competitive Analysis for Online Optimization, <www.ima.umn.edu/~mali/Online_Brown-Bag_Slides.pdf>, 2002

[10] G. TIAN, U. SHARMA, T. WOOD, S. SAHU, and P. SHENOY, Seagull: intelligent cloud bursting for enterprise applications, in Proceedings of the Usenix Annual Technical Conference, <www.usenix.org/system/files/conference/atc12/atc12-final57.pdf>, 2012.