The Online Simple Knapsack Problem with Reservation and Removability

Elisabet Burjons
York University, Toronto, Canada

Matthias Gehnen[1]
RWTH Aachen University, Germany

Henri Lotze
RWTH Aachen University, Germany

Daniel Mock
RWTH Aachen University, Germany

Peter Rossmanith
RWTH Aachen University, Germany

Abstract

In the *online simple knapsack problem*, a knapsack of unit size 1 is given and an algorithm is tasked to fill it using a set of items that are revealed one after another. Each item must be accepted or rejected at the time they are presented, and these decisions are irrevocable. No prior knowledge about the set and sequence of items is given. The goal is then to maximize the sum of the sizes of all packed items compared to an optimal packing of all items of the sequence.

In this paper, we combine two existing variants of the problem that each extend the range of possible actions for a newly presented item by a new option. The first is *removability*, in which an item that was previously packed into the knapsack may be finally discarded at any point. The second is *reservations*, which allows the algorithm to delay the decision on accepting or rejecting a new item indefinitely for a proportional fee relative to the size of the given item.

If both removability and reservations are permitted, we show that the competitive ratio of the *online simple knapsack problem* rises depending on the relative reservation costs. As soon as any nonzero fee has to be paid for a reservation, no online algorithm can be better than 1.5-competitive. With rising reservation costs, this competitive ratio increases up to the golden ratio ($\phi \approx 1.618$) that is reached for relative reservation costs of $1 - \frac{\sqrt{5}}{3} \approx 0.254$. We provide a matching upper and lower bound for relative reservation costs up to this value. From this point onward, the tight bound by Iwama and Taketomi for the *removable knapsack problem* is the best possible competitive ratio, not using any reservations.

2012 ACM Subject Classification Theory of computation

Keywords and phrases online algorithm, knapsack, competitive ratio, reservation, preemption

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.29

1 Introduction

Online problems model situations where an algorithm receives an input piece-wise. The analysis of online problems does not revolve around how much time and space is used when solving the problem, but how *good* the solution provided by an online algorithm is with respect to an optimal offline counterpart. In classical online models, an online algorithm receives a piece of input or *request* and must output some irrevocable part of the solution before the next request arrives, without room for changing the output later on.

[1] corresponding author

The *simple knapsack problem* is, as the name suggests, a simplification of the classical *knapsack problem*: Given a set of items I, a size function $w : I \to \mathbf{R}$ and a gain function $g : I \to \mathbf{R}$, the task is to find a subset of items $S \subseteq I$ such that $\sum_{i \in S} w(i) \leq 1$ and $\sum_{i \in S} g(i)$ is maximized. In the *simple knapsack problem*, $w : I \to [0, 1]$ and $w(i) = g(i)$ for all items $i \in I$. The offline knapsack problem is a classical hard optimization problem and has been shown to be NP-complete [14]. Both the knapsack problem and the simple knapsack problem admit fully polynomial time approximation schemes [11].

In this work we study the *online simple knapsack problem*. Given a knapsack with capacity 1, an algorithm receives a request sequence $S = \{x_1, \ldots, x_n\}$ consisting of items of size smaller than or equal to 1, and must decide at each step whether to take such an item or reject it. The length of the sequence is initially unknown to the algorithm and the decision on each item must be made before the next item arrives. The decisions are irrevocable, and once an item has been packed or rejected, the decision cannot be reversed in later steps.

Competitive analysis was introduced by Sleator and Tarjan [19] as a way to analyze the performance of online algorithms. The *strict competitive ratio* of an online algorithm for an online maximization problem, such as the *online simple knapsack problem*, is the ratio between the gain obtained by an optimal offline algorithm and the gain obtained by the online algorithm in the worst-case instance. For an introduction to online analysis, we refer the reader to the standard book by Borodin and El-Yaniv [4].

Several variants of online models have been introduced to discern which problems are truly complex, or have a high information content, and which problems are simple in general but contain some rare worst-case instances yielding large competitive ratios. Some of these approaches, such as randomized algorithms [1], advice complexity [7, 10, 15], randomization of the adversary [6, 16, 18], etc. have been applied to various online problems, including the *online simple knapsack problem*, with some success.

In the general online setting, it is well-known [17] that there cannot be an algorithm for the online simple knapsack problem – and thus also for the online knapsack problem – with a bounded competitive ratio. Thus, most research so far has been focused on analyzing some variants of the problem or of the online model. Iwama and Taketomi [12] introduced a variant that adds the possibility to remove previously packed items from the knapsack, without any additional costs. This can be done at any point during the run of an algorithm. They showed a tight competitive ratio of the golden ratio $\phi \approx 1.618$ for this model.

Recently Böckenhauer et al. [2] introduced a model for the online simple knapsack that adds the possibility to reserve items for a fee proportional to their size. In this model, if an item is reserved, the decision whether the item should be added to the knapsack can be postponed to a later point. However, if an algorithm decides to reserve an item, it has to pay some reservation cost $c = x\alpha$ that depends on a given parameter α between 0 and 1, and the item size x. Surprisingly, in this case, even for arbitrarily small values of the reservation parameter α, the competitive ratio is not better than 2.

In this paper, we consider a variant of the online simple knapsack problem where an algorithm is allowed to reserve items (reservation) or alternatively to remove items previously placed into the knapsack (removability). Both of these alternatives allow an algorithm to postpone or alter decisions on whether to pack an object. Our reason to choose this variant is that it is a variant of the online simple knapsack with bounded competitive ratio below 2, the best ratio that the reservation model allowed. Some of the questions we wanted to answer is whether the reservations would help at all, and if so, if it is possible to achieve a competitive ratio approaching 1 for low reservation costs. We also want to know if the model behaves similarly to the secretary problem with reservation [5], where for large reservation costs the competitive ratio is the same as without reservations.

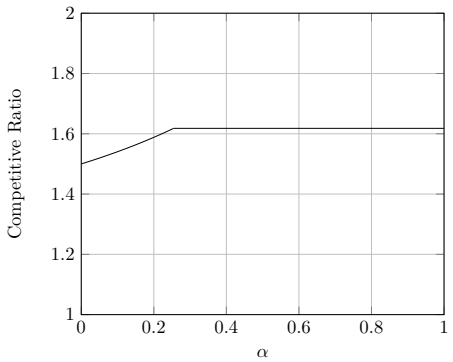

Figure 1 A schematic plot of the competitive ratio with respect to the reservation costs α for the online simple knapsack problem with reservation and removability.

Other variants of the knapsack problem include the work by Böckenhauer et al. [3], who studied the online simple knapsack problem with advice and randomization, and concluded that with one single advice or random bit the problem becomes 2-competitive, whereas further additional random bits do not improve the competitive ratio. Iwama and Zhang [13] looked at the problem with resource augmentation, which means that the size of the knapsack for the online problem is slightly larger than the knapsack size of the offline algorithm. Han, Kawase and Makino [8] considered a version where the items may be removed from the knapsack at the cost of a factor of the item size, and showed that if the factor is smaller than 1/2, the competitive ratio is 2, and otherwise it is a function depending on the factor itself. This is different from the variant we consider, as in this case one pays to remove items from the knapsack, whereas in our version the algorithm pays to reserve items, but not to remove them. Another variant of the problem allows for a buffer of constant size, in which items may be intermediately stored. Han et al. [9] studied the case where the buffer has at least as much capacity as the knapsack itself; the items presented may be allocated into the buffer or irrevocably rejected and only in the last step are the items selected for placement into the knapsack. The reservation model has also been recently applied to the secretary problem [5], where one can achieve a competitive ratio as close to 1 as possible with diminishing values of the reservation cost. For large reservation costs, the competitive ratio is the same as in the model without reservation.

For the online simple knapsack problem with both removability and reservation, we provide tight upper and lower bounds of $\frac{3-1.5\alpha}{2-1.5\alpha}$ for $0 < \alpha \leq 1 - \sqrt{5}/3 \approx 0.2546$, which goes from 1.5 to ϕ. For larger values of α, the competitive ratio stays constant at ϕ. These bounds are depicted in Figure 1. This means that, even if the reservation costs are arbitrarily small the competitive ratio is worse than 1.5, and for any $\alpha > 0.2546$ the best achievable competitive ratio is the one that can be achieved without reserving any items. This is in contrast to the behavior of the competitive ratio for the reservation model without removability, where the competitive ratio is constant at value 2 on the small range, and grows until being unbounded for large values of α.

1.1 Problem Definition

We now formally define the online simple knapsack problem with reservation and removability.

Just as in the standard online simple knapsack problem, defined in the introduction, an algorithm solving the problem with reservation and removability is given a knapsack with capacity 1, and receives a request sequence $S = \{x_1, \ldots, x_n\}$ of items. Every item x_i of the

request sequence only has one parameter, the size, thus it is possible to refer to the size of each item as x_i while keeping the context clear. At the beginning of the request sequence, the algorithm is also is given a parameter $0 \leq \alpha \leq 1$ for the reservation costs. After receiving an item of size x the algorithm has three options. It can either decide to pack this item – assuming the total size of already packed items together with the new item is smaller than 1 – , reserve this item at cost αx, or reject this item. Moreover, the algorithm can also decide to remove and reject any item currently in the knapsack at any point. The decision to reject an item is irrevocable.

Once the whole request sequence S has been processed, an algorithm A can decide to pack any reserved items, removing any items in the knapsack accordingly if necessary, as long as they fit into the knapsack. The goal of an algorithm is to maximize its total size of items packed in the knapsack after the whole instance has been processed minus the reservation costs, that is,

$$\text{gain}_S(A, \alpha) = \sum_{x_i \text{ packed}} x_i - \sum_{x_i \text{ reserved}} \alpha x_i \ .$$

The performance of an algorithm is measured against the optimal solution in a worst-case manner. Thus, the competitive ratio of an algorithm A is the highest ratio between the size of an optimal solution OPT and the gain of A, over all instances

$$\text{c(A}, \alpha) = \sup_S \left\{ \frac{\text{gain}_S(OPT, \alpha)}{\text{gain}_S(A, \alpha)} \right\} \ .$$

We now present matching upper and lower bounds for the competitive ratio of any algorithm solving the online simple knapsack problem with reservation and removability.

2 Lower Bound

First note that no algorithm can be better than optimal, in particular we have a lower bound of 1 for $\alpha = 0$.

In this section we show that no algorithm can reach a competitive ratio smaller than $\min\{\frac{3-1.5\alpha}{2-1.5\alpha}, \phi\}$ for all $0 < \alpha < 1$. The ratio $\frac{3-1.5\alpha}{2-1.5\alpha}$ is equal to ϕ at $\alpha = 1 - \frac{\sqrt{5}}{3} \approx 0.2546$. For any larger values of α, we prove that it is not possible to construct an algorithm that performs better than the strategy of Iwama and Taketomi for the knapsack with removability and without reservation [12].

▶ **Theorem 1.** *Given a parameter $0 < \alpha \leq 1 - \frac{\sqrt{5}}{3}$, there exists no algorithm for online simple knapsack with reservation and removability achieving a competitive ratio better than $\frac{3-1.5\alpha}{2-1.5\alpha}$.*

Proof. We present an adversarial strategy to show that no algorithm can reach a competitive ratio of $\frac{3-1.5\alpha}{2-1.5\alpha} - \varepsilon$ for any given $\varepsilon > 0$. For the following analysis, we will abbreviate $\frac{2-1.5\alpha}{3-1.5\alpha}$ with k. Consider the following set of adversarial instances with $0 < \delta \ll \varepsilon$:

The adversary starts by presenting the first item x_1 of size $1 - k + \delta$. In this situation an arbitrary algorithm A can choose between the following options:

Case 1, Pack x_1. A packs the first item of size $1 - k + \delta$ into the knapsack. The adversary next presents the second item x_2 of size $k + \delta$ that barely does not fit together with the first item.

Case 1.1, Remove x_1 and pack x_2. A discards the item of size $1-k+\delta$ and packs x_2. The adversary then presents an item of size $1-x_1 = k-\delta$. This would perfectly fit together with the discarded item x_1, so OPT has a solution of size 1. Since this item does not fit together with x_2, A only holds x_2 as the larger one of the two items, which results in the desired competitive ratio.

Case 1.2, Reserve x_2. A reserves the item of size $k+\delta$. The adversary presents the item x_3 of size $k+\delta^2$.

Case 1.2.1, Remove x_1 and pack x_3. Assuming A discards the first item that is still held in its knapsack to pack the item of size $k+\delta^2$. Similar to Case 1.1, the adversary then presents the counterpart to the discarded item of size $1-x_1 = k-\delta$. A cannot pack its reserved item together with the newly packed item, since $k-\delta+k+\delta^2 > 1$ for all $\alpha \in (0, 1-\frac{\sqrt{5}}{3})$. The gain of A is even smaller than in Case 1.1, due to the costs of reserving an item.

Case 1.2.2, Reserve x_3. A reserves the item of size $k+\delta^2$. The adversary presents x_4 of size $k+\delta^3$ and will continue to present items x_{j+1} of size $k+\delta^j$ as long as A reserves those items. At some point A must stop reserving these items due to the accumulating reservation costs exceeding any possible gain. As soon as A rejects an item, or discards the item in its knapsack to pack an item, the case can be handled analogously to 1.2.1 or 1.2.3. The gain of the algorithm will be lower than in these cases due to the added reservation costs.

Case 1.2.3, Reject x_3. The algorithm does not pack the item of size $k+\delta^2$. The adversary presents an item of size $1-x_3$. So while OPT reaches 1, A can at the most pack x_1 and the item of size $1-x_3$. Together, with subtracted reservation costs, the algorithm then has a gain of $2-\frac{4-3\alpha}{3-1.5\alpha}-\alpha k$. A simple analysis shows that this yields a ratio higher than $\frac{3-1.5\alpha}{2-1.5\alpha}$.

Case 1.3, Reject x_2. A does not pack or reserve the item of size $k+\delta$. No further items are presented. The algorithm thus only holds x_1, while an optimal solution would be to pack the item x_2. The ratio of k to $1-k$ is worse than ϕ for every $\alpha < 1-\frac{\sqrt{5}}{3}$ and therefore, in particular, larger than the desired ratio.

Case 2, Reserve x_1. A reserves the first item of size $1-k+\delta$. The adversary presents the item x'_2 of size $\frac{2}{3}+2\delta$.

Case 2.1 Pack x'_2. The algorithm packs the item of size $\frac{2}{3}+2\delta$. The next item is x'_3 of size $\frac{1}{3}+2\delta$.

Case 2.1.1 Remove x'_2 and pack x'_3. The algorithm discards the item of size $\frac{2}{3}+2\delta$ from its knapsack and instead packs the item of size $\frac{1}{3}+2\delta$. The adversary next presents an item of size $1-x'_2$. OPT then has a solution of size 1. A can at most reach a gain of $x_1+x'_3-\alpha x_1 = \frac{1}{3}+1-k+3\delta-\alpha(1-k) = k$.

Case 2.1.2 Reserve x'_3. A reserves the item of size $\frac{1}{3}+2\delta$. The adversary continues to present items x'_{j+1} of size $\frac{1}{3}+\delta+\delta^{j-1}$, as long as A reserves those items. As soon as A rejects an item or discards the item already in its knapsack to pack an item, these cases can be handled analogously to 2.1.1 or 2.1.3. The gain of the algorithm will be lower than in these cases due to the added reservation costs.

Case 2.1.3 Reject x'_3. The algorithm does not pack the item of size $\frac{1}{3}+2\delta$. The adversary then presents an item of size $1-x'_3$. Since x'_3 was the smallest item of all that have been presented so far, the counterpart will not fit together with any other item of A. Therefore OPT has a gain of 1, whereas the largest packing of A consists of the item x'_2 which is of smaller size than k.

Case 2.2 Reserve x'_2. A reserves the item of size $\frac{2}{3}+2\delta$. The adversary presents x''_2 of size $\frac{2}{3}+\delta+\delta^2$.

Case 2.2.1 Pack x''_2. A packs the item of size $\frac{2}{3}+\delta+\delta^2$ into the knapsack. The next item is x'_3 of size $\frac{1}{3}+2\delta$.

Case 2.2.1.1 Remove x''_2 and pack x'_3. The algorithm discards the item of size $\frac{2}{3}+\delta+\delta^2$ from its knapsack and instead packs the item of size $\frac{1}{3}+2\delta$. The adversary next presents an item of size $1-x''_2$. OPT then has a solution of size 1. A can at most reach a gain of $x_1+x'_3-\alpha x_1=\frac{1}{3}+1-k+3\delta-\alpha(1-k)=k$.

Case 2.2.1.2 Reserve x'_3. A reserves the item of size $\frac{1}{3}+2\delta$. The adversary continues to present items x'_{j+1} of the size $\frac{1}{3}+\delta+\delta^{j-1}$, as long as A reserves those. As soon as A rejects an item or discards the item already in its knapsack to pack an item, these cases can be handled analogously to 2.2.1.1 or 2.2.1.3. The gain of the algorithm will be lower than in these cases due to the added reservation costs.

Case 2.2.1.3 Reject x'_3. The algorithm does not pack the item of size $\frac{1}{3}+2\delta$. The adversary then presents an item of size $1-x'_3$. Since x'_3 was the smallest item of all that have been presented so far, the counterpart will not fit together with any other item of A. Therefore OPT has a gain of 1, whereas the largest packing of A consists of the item $x'_2<k$.

Case 2.2.2 Reserve x''_2. The algorithm reserves the item of size $\frac{2}{3}+\delta+\delta^2$. The adversary continues to present items x''_j of the size $\frac{2}{3}+\delta+\delta^j$, as long as A reserves these items. As soon as A rejects an item or discards the item already in its knapsack to pack an item, these cases can be handled analogously to 2.2.1 or 2.2.3. The gain of the algorithm will be lower than in these cases due to the added reservation costs.

Case 2.2.3 Reject x''_2. A does not pack the item of size $\frac{2}{3}+\delta+\delta^2$. The adversary then presents an item of size $1-x''_2$. The gain of OPT is then 1. The maximum gain of A is then $x_1+1-x''_2-\alpha(x_1+x'_2)=1-k+\frac{1}{3}-\delta-\delta^2-\alpha(1-k+\frac{1}{3}-\delta-\delta^2)$ which is slightly smaller than the gain of A in Case 2.1.1 due to increased reservation costs and thus especially smaller than k.

Case 2.3 Reject x'_2. The algorithm does not pack the item of size $\frac{2}{3}+2\delta$. No further items are presented. Similarly to Case 1.3, the algorithm only holds x_1, while an optimal packing would be x'_2. Since even $x'_2>x_2$, the same argumentation from Case 1.3 concludes this case.

Case 3, Reject x_1. No further items are presented. Therefore the optimal solution is exactly the item $x_1=1-k$, while A has packed no items. The competitive ratio is then unbounded. ◀

It is not surprising that for increasing reservation costs, the competitive ratio is increasing, too. In particular, it is easy to see that the competitive ratio cannot decrease if the reservation costs increase.

▶ **Corollary 2.** *For $\alpha>1-\frac{\sqrt{5}}{3}$, no algorithm can be better than ϕ-competitive for the online simple knapsack problem with reservation and removability.*

Proof. The competitive ratio of Theorem 1 reaches ϕ at $\alpha=1-\frac{\sqrt{5}}{3}$. For every $\alpha>1-\frac{\sqrt{5}}{3}$ it is possible to present the lower bound with exactly the same items as in Theorem 1 for $\alpha=1-\frac{\sqrt{5}}{3}$. The only difference are the higher costs for every reservation. Thus, in every case, the gain of an optimal offline solution stays the same, while the algorithm gets at most the gain of the case $\alpha=1-\frac{\sqrt{5}}{3}$. ◀

3 Upper Bound

If $\alpha = 0$, then the problem becomes an offline problem, since an algorithm can reserve every item for free. Thus a competitive ratio of 1 is achievable. Trivially, this matches the lower bound.

However, as we will see now, as soon as the value of α is positive, the best achievable competitive ratio is worse than 1.5-competitive.

To handle the range of $0 < \alpha \leq 1 - \frac{\sqrt{5}}{3}$, we present Algorithm 1 with a competitive ratio of $\frac{3-1.5\alpha}{2-1.5\alpha}$, matching the lower bound established in the previous subsection.

Algorithm 1, analyzed in this section, can be split in three phases. All of the phases work similarly: First, if small items arrive, they are packed since they will not cause harm (we can throw them out at any time).

Then there are two kind of interesting ranges for medium-sized items: A large-medium range, where the algorithm tries to keep the smallest items to maintain the possibility to get fitting counterparts (similar to [12]). And a small-medium range, where an item gets reserved, because otherwise the ratio between large and small items in the large-medium range, where only the smallest items are kept, gets too large by itself. After the first reservation, the algorithm continues in the second phase; after the second reservation in the third phase.

Finally, very large items, which are sufficiently large to guarantee the desired competitive ratio by themselves, are the easiest case since the algorithm can just pack these items to achieve the desired competitive ratio immediately.

The main difference between the three phases is, that the ranges for the different actions are shifted – due to the ability to use the reserved items, but also due to the paid reservation costs that need to be compensated. Due to these shifts, in the third phase there is no small-medium range, and there is no need to reserve any more items, thus Algorithm 1 reserves at most two items.

▶ **Theorem 3.** *For $0 < \alpha \leq 1 - \frac{\sqrt{5}}{3}$, there exists an algorithm solving the online simple knapsack problem with reservation and removability that is not worse than $\frac{3-1.5\alpha}{2-1.5\alpha}$ competitive.*

Proof. We prove that Algorithm 1 achieves the competitive ratio on the statment. For ease of notation, we define $c := \frac{3-1.5\alpha}{2-1.5\alpha}$. If Algorithm 1 is at any point able to pack a knapsack such that, even when paying the reservation costs, the desired competitive ratio is reached, it stops afterwards. This is the case in lines 4, 9 and 16.

Depending on the instance, Algorithm 1 reserves zero, one or two items. In the following, we show that the algorithm yields the desired competitive ratio in those three cases.

Case 1: Algorithm 1 ends in line 4 or 22, before the first reservation is made. When presented with a new item, the algorithm first checks if this item together with a subset of its packed items already yields a packing of size $\frac{1}{c}$. In the following analysis, assume that this is not the case, since if this were the case, the algorithm would have achieved the desired competitive ratio anyways.

Items that are smaller than $1 - \frac{1}{c}$ are packed by the algorithm in line 5. Items with sizes between $1 - \frac{1}{c}$ and $\frac{1}{c^2}$ will be reserved in line 7, if the condition in line 4 does not hold: We investigate this case as Case 2.

The first item of size between $\frac{1}{c^2}$ and $\frac{1}{c}$ is packed. If further items in this range arrive, only the smallest item of this range is packed (or kept) in the knapsack and the other items from this range are discarded. If an item x_k in this range, which should be kept in the knapsack according to line 6, does not fit, it must be due to the items already in the knapsack, which are each smaller than $1 - \frac{1}{c}$. Thus, in this case, it is possible to combine this item x_k with a subset of the small items in the knapsack to get a packing of size at least $\frac{1}{c}$, triggering the end of the packing in line 4.

Algorithm 1 Upper bound Algorithm for $0 < \alpha \leq 1 - \frac{\sqrt{5}}{3}$. $\mathrm{OPT}(S)$ denotes the size of the optimal packing of the set S, $c := \frac{3-1.5\alpha}{2-1.5\alpha}$.

1: $K := \emptyset$; $R := \emptyset$
2: **for** $k = 1, \ldots, n$ **do**
3: **if** $|R| = 0$ **then**
4: **if** $\mathrm{OPT}(K \cup \{x_k\}) \geq \frac{1}{c}$ **then** Pack $\mathrm{OPT}(K \cup \{x_k\})$ **END**
5: **else if** $x_k \leq 1 - \frac{1}{c}$ **then** $K := K \cup \{x_k\}$;
6: **else if** $x_k \geq \frac{1}{c^2}$ **then** Keep only smallest $x'_k \in K \cup x_k$ with $x'_k \in [\frac{1}{c^2}, \frac{1}{c}]$;
7: **else**$(1 - \frac{1}{c} < x_k < \frac{1}{c^2})$ $R := R \cup x_k$, $x_f := x_k$;
8: **else if** $|R| = 1$ **then**
9: **if** $\mathrm{OPT}(K \cup \{x_f, x_k\}) \geq \frac{1}{c} + \alpha x_f$ **then** Pack $\mathrm{OPT}(K \cup \{x_f, x_k\})$ **END**
10: **else if** $1 - \frac{1}{c} - \alpha x_f < x_k \leq \frac{1}{3}$ **then** $R := R \cup x_k$, $g := k$;
11: **else if** $x_k \leq 1 - \frac{1}{c} - \alpha x_f$ **then** $K := K \cup \{x_k\}$;
12: **else** Keep only smallest $x'_k \in K \cup x_k$ with $x'_k \in [1 - x_f, \frac{1}{c} + \alpha x_f]$;
13: **else**
14: $M := \emptyset$; $L := \emptyset$
15: **if** $\mathrm{OPT}(K \cup \{x_f, x_g, x_k\}) \geq \frac{1}{c} + \alpha(x_f + x_g)$ **then**
16: Pack $\mathrm{OPT}(K \cup \{x_f, x_g, x_k\})$ **END**
17: **else if** $x_k \leq 1 - \frac{1}{c} + \alpha(x_f + x_g)$ **then** $K := K \cup \{x_k\}$;
18: **else if** $1 - x_g - x_f < x_k < \frac{1}{c} + \alpha(x_g + x_f) - x_f$ **then**
19: Keep x_k if $x_k + x_g < \min(x_l \in L) \wedge x_k < \min(x_m \in M)$; $M := M \cup x_k$;
20: **else if** $1 - x_g < x_k < \frac{1}{c} + \alpha(x_f + x_g)$ **then**
21: Keep x_k if $x_k + x_g < \min(x_m \in M) \wedge x_k < \min(x_l \in L)$; $L := L \cup x_k$;
22: Pack $OPT(K \cup R)$

On the other hand, if the instance ends before reserving the first item, the only possible difference between an optimal solution and a solution of the algorithm is that the algorithm only has the smallest item in the range between $\frac{1}{c^2}$ and $\frac{1}{c}$, while the optimal solution might have a different item in this range. Since the smallest item in this range is at least of size $\frac{1}{c^2}$ and the largest item is at most of size $\frac{1}{c}$, the ratio is smaller than $\frac{1/c}{1/c^2} = c$. Additional small items can be part of both the optimal solution and the solution provided by the algorithm, which only makes the ratio smaller.

Note that an optimal solution cannot contain two items in the range between $\frac{1}{c^2}$ and $\frac{1}{c}$, since the algorithm would also be able to combine the two items in this range. The algorithm always keeps the smallest possible item within the range, a second item within the range that could fit into the knapsack would trigger the condition in line 4, since two items of size at least $\frac{1}{c^2}$ are sufficient to reach $\frac{1}{c}$ ($\frac{2}{c^2} \geq \frac{1}{c}$ for $c \leq 2$).

Case 2: Algorithm ends in line 9 or 22 with one reserved item. Assume one item x_f in the range $1 - \frac{1}{c}$ to $\frac{1}{c^2}$ was reserved. The reservation costs for this item are αx_f. Thus, the algorithm needs to pack at least $\frac{1}{c} + \alpha x_f$ to guarantee the desired competitive ratio independently to the gain achieved by an optimal solution.

For items smaller than $1 - \frac{1}{c} - \alpha x_f$ the same argument for the small items in Case 1 holds: These items can be packed in a greedy manner until the threshold $\frac{1}{c} + \alpha x_f$ is reached. An item of size between $1 - \frac{1}{c} - \alpha x_f$ and $\frac{1}{3}$ will be reserved in line 10, if the algorithm cannot pack a knapsack of size $\frac{1}{c} + \alpha x_f$ at this point. This case is discussed in the next part of the proof, as Case 3.

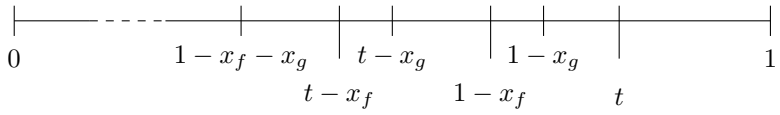

Figure 2 A schematic plot of the size ranges in the algorithm when two items have already been reserved. The targeted size for the items is $t = \frac{1}{c} + \alpha(x_g + x_f)$.

Note that it is sufficient to combine the first reserved item with a single item of size at least $\frac{1}{3}$ to reach the desired competitive ratio, even after paying the reservation costs, as

$$\frac{1}{3} + (1-\alpha)x_f \geq \frac{1}{3} + (1-\alpha)\left(1 - \frac{1}{c}\right) = \frac{1}{c},$$

in the considered range of α. In addition, any item that reaches $\frac{1}{c} + \alpha x_f$ either by its own or together with small items is sufficient as well. Thus, with every item between $\frac{1}{3}$ and $1 - x_f$, or larger than $\frac{1}{c} + \alpha x_f$, the threshold is reached already and the algorithm stops in line 9.

We are left to consider items of sizes between $1 - x_f$ and $\frac{1}{c} + \alpha x_f$, which will reach line 12 in the algorithm. With the items in this range, we follow the same strategy as for the large-medium items before the first reservation: The smallest of those items will be kept in the knapsack. This works for the same reasons as in Case 1, as explained below.

Assume the instance stops here and the algorithm was not able to reach $\frac{1}{c} + \alpha x_f$: The only difference between the solution of the algorithm and an optimal solution is that the algorithm sticks to the smallest item in the range between $1 - x_f$ and $\frac{1}{c} + \alpha x_f$, which gives us the worst possible case. Note that no item of size smaller than $1 - x_f$ can be rejected before the first reservation, since it otherwise would have been packed in combination with f. The ratio between $\frac{1}{c} + \alpha x_f$ and $1 - x_f$ is less than c, since x_f is at most $\frac{1}{c^2}$. Again, additional small items can be added to both the optimal solution and the solution provided by the algorithm, only decreasing the ratio.

Case 3: Algorithm ends in line 16 or 22 with two reservations made. Assume two items were reserved, the first item x_f of size between $1 - \frac{1}{c}$ and $\frac{1}{c^2}$, and the second item x_g of size between $1 - \frac{1}{c} - \alpha x_f$ and $\frac{1}{3}$. Therefore, the algorithm has already paid $\alpha(x_g + x_f)$ as reservation costs, and now needs a packing of size $\frac{1}{c} + \alpha(x_g + x_f)$ to guarantee the competitive ratio of c independent of any optimal solution size.

First note that, an item or a subset of items is sufficient to reach $\frac{1}{c} + \alpha(x_f + x_g)$ in three cases: it either reaches the threshold by its own, reaches the threshold together with one of the reserved items, or with the two reserved items combined.

To reach the threshold with one of the reserved items, a subset of items must have a size between $\frac{1}{c} + \alpha(x_f + x_g) - x_f$ and $1 - x_f$, or between $\frac{1}{c} + \alpha(x_f + x_g) - x_g$ and $1 - x_g$ (depending on which of the reserved items will be used to combine this subset with). The subsets sizes, that combined with x_f are sufficient to reach the necessary threshold can obviously be smaller than the subsets that can be combined with x_g, since $x_g < \frac{1}{3}$ and $x_f > \frac{1}{3}$. With simple analysis, it can be shown that for any α between 0 and $1 - \frac{\sqrt{5}}{3}$ and for every choice of x_g and x_f in the given ranges, $1 - x_g$ must be larger than $\frac{1}{c} + \alpha(x_f + x_g) - x_f$. Therefore these intervals overlap, as displayed in Figure 2.

It follows that there are three intervals in which an item can be presented and the algorithm might not stop immediately:

1. Items smaller than $\frac{1}{c} + \alpha(x_g + x_f) - x_g - x_f$ are too small to reach the necessary size in combination with both of the reserved items. Note that those are smaller than $1 - \frac{1}{c} - \alpha(x_g + x_f)$, thus they can be packed greedily. As soon as the knapsack is overfull due to those items a subset of the small items can be selected to reach $\frac{1}{c} + \alpha(x_g + x_f)$. This argumentation is analogous to the small items argument in the two previous cases.

2. Items between $1 - x_g - x_f$ and $\frac{1}{c} + \alpha(x_g + x_f) - x_f$, which are too large to fit together with both reserved items, but too small to reach the threshold in combination with the largest reserved item. We will call those items *mid-sized*.
3. Items that cannot be combined any of the reserved items but also do not reach the threshold by themselves, i.e., items between $1 - x_g > \frac{2}{3}$ and $\frac{1}{c} + \alpha(x_f + x_g)$. Those items will be called *large*.

As we have already justified, the small items in 1. are not a concern. We argue now that the desired competitive ratio is also achieved when mid-sized and large items appear in the request sequence after reserving two items.

The algorithm keeps either the smallest large item, or the smallest mid-sized item. This is dependent on the situation, as detailed in lines 18 to 21: If the smallest large item is smaller than the smallest mid-sized item in combination with x_g, the algorithm keeps the smallest large item. On the other hand, if the smallest mid-sized item in combination with x_g is smaller than the smallest large item, the algorithm keeps the smallest mid-sized item. Note that we can assume that every large item is larger than $\frac{2}{3}$, since otherwise it can be combined with x_g, reaching the threshold $\frac{1}{c} + \alpha(x_g + x_f)$.

Therefore the following crucial observation holds: If the algorithm is able to pack either one large item together with x_f, x_g or a mid-sized item, or the algorithm is able to pack three items among x_f, x_g and mid-sized items, the algorithm will also reach the desired threshold and competitive ratio.

To see this, first note that every mid-sized item of size smaller than or equal to $\frac{1}{3}$ will be packed into the knapsack and will never be replaced by a large item. This is the case because every large item must be larger than $\frac{2}{3}$, and an item of size smaller than $\frac{1}{3}$ in combination with x_g must be smaller or equal than $\frac{2}{3}$.

Assume that, at some point, an arbitrary algorithm is about to pack the last item of a combination of one large and one mid-sized or reserved item, or a combination of three mid-sized or reserved items. Due to the construction and the selection of items before, we are able to do so as well:

- If the last item is a large item, the algorithm is able to pack such a combination, since the smallest mid-sized item is in the knapsack if it is smaller than the reserved x_g. Otherwise the reserved x_g is available for a combination.
- If the last item is a mid-sized one, it fits together with the current combination of items selected by Algorithm 1 (either two mid-sized or one large item), since it is the smallest possible combination by construction.

A simple calculation then shows that every such combination of items (one large and one mid-sized/reserved item, or three mid-sized/reserved items) is of size at least $\frac{1}{c} + \alpha(x_g + x_f)$, even if chosen as small as possible.

The only thing left to prove is that even if there is no such combination, the algorithm still reaches the designated competitive ratio:

First note that large items are only rejected or removed from the knapsack in the first two phases. Those items had at most size $\frac{1}{c} + \alpha x_f$. Items that remained in the knapsack can be considered as either small items that can be packed in a greedy manner, or as large items. If there were an item that is not in the allowed range for large or small items in this phase, it would have been used to reach the threshold for a guaranteed competitive ratio immediately, just like it would have been possible if the item was presented in the third phase.

By using mid-sized and large items together with the reserved ones, every optimal solution can only reach $\frac{1}{c} + \alpha(x_g + x_f)$, while Algorithm 1 cannot be worse than $x_g + x_f$ on any instance. An easy calculation shows that even for smallest possible sizes of x_g and x_f, their

combination is sufficient to reach the competitive ratio of c, since an optimal solution cannot be better than $\frac{1}{c} + \alpha(x_g + x_f)$. Since Algorithm 1 can use small items in the same manner as an optimal solution, the appearence of small items cannot worsen the ratio. ◀

This concludes the analysis for an upper bound that matches the lower bound for values of α smaller than $\alpha \geq 1 - \sqrt{5}/3$.

For any $\alpha \geq 1 - \sqrt{5}/3$, note that the algorithm for knapsack with removable objects (without reservation) presented by Iwama and Taketomi [12] achieves a competitive ratio of ϕ, thus matching our lower bound as well.

4 Concluding Remarks

The upper and lower bounds for the competitive ratio of the online simple knapsack with reservation and removability show that, as long as the value of α is positive the competitive ratio is above 1.5. For large values of α the competitive ratio stays the same as in the removability model without reservation, which is somehow intuitive, showing that at some point reserving items is not worth the costs. More surprising are the results for small values of α. The algorithm with reservation that achieves the desired upper bound rarely makes use of the reservation, even in the case with very small α only two items are reserved in total. This behavior is similar to that of online simple knapsack with reservations but without removability for small values of α.

For future research, a natural variation of the problem would be to allow items to be reserved after they were added to the knapsack, i.e. to allow removing items from the knapsack and reserve them instead of only having the option to discard them (like we considered here).

It is known that the classic general online knapsack does not yield a finite competitive ratio and also the option to remove items of the knapsack does not help. Thus, it remains an interesting question if the reservation model changes the situation. We think that, at least for small reservation costs, it should be possible to achieve a constant competitive ratio, but this is still open and could also be a topic for further research.

References

1 Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in online algorithms (extended abstract). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 379–386. ACM, 1990. doi:10.1145/100216.100268.

2 Hans-Joachim Böckenhauer, Elisabet Burjons, Juraj Hromkovic, Henri Lotze, and Peter Rossmanith. Online simple knapsack with reservation costs. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.16.

3 Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theor. Comput. Sci.*, 527:61–72, 2014. doi:10.1016/j.tcs.2014.01.027.

4 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

5 Elisabet Burjons, Matthias Gehnen, Henri Lotze, Daniel Mock, and Peter Rossmanith. The secretary problem with reservation costs. In Chi-Yeh Chen, Wing-Kai Hon, Ling-Ju Hung, and Chia-Wei Lee, editors, *Computing and Combinatorics – 27th International Conference, COCOON 2021, Tainan, Taiwan, October 24-26, 2021, Proceedings*, volume 13025 of *Lecture Notes in Computer Science*, pages 553–564. Springer, 2021. doi:10.1007/978-3-030-89543-3_46.

6 Elisabet Burjons, Juraj Hromkovic, Rastislav Královic, Richard Královic, Xavier Muñoz, and Walter Unger. Online graph coloring against a randomized adversary. *Int. J. Found. Comput. Sci.*, 29(4):551–569, 2018. doi:10.1142/S0129054118410058.

7 Stefan Dobrev, Rastislav Královic, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO Theor. Informatics Appl.*, 43(3):585–613, 2009. doi:10.1051/ita/2009012.

8 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Online unweighted knapsack problem with removal cost. *Algorithmica*, 70(1):76–91, 2014. doi:10.1007/s00453-013-9822-z.

9 Xin Han, Yasushi Kawase, Kazuhisa Makino, and Haruki Yokomaku. Online knapsack problems with a resource buffer. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPIcs*, pages 28:1–28:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ISAAC.2019.28.

10 Juraj Hromkovic, Rastislav Královic, and Richard Královic. Information complexity of online problems. In Petr Hlinený and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 24–36. Springer, 2010. doi:10.1007/978-3-642-15155-2_3.

11 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975. doi:10.1145/321906.321909.

12 Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 293–305. Springer, 2002. doi:10.1007/3-540-45465-9_26.

13 Kazuo Iwama and Guochuan Zhang. Online knapsack with resource augmentation. *Inf. Process. Lett.*, 110(22):1016–1020, 2010. doi:10.1016/j.ipl.2010.08.013.

14 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

15 Dennis Komm. *An Introduction to Online Computation – Determinism, Randomization, Advice*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016. doi:10.1007/978-3-319-42749-2.

16 Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. *SIAM J. Comput.*, 30(1):300–317, 2000. doi:10.1137/S0097539796299540.

17 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Math. Program.*, 68:73–104, 1995. doi:10.1007/BF01585758.

18 Prabhakar Raghavan. A statistical adversary for on-line algorithms. In Lyle A. McGeoch and Daniel Dominic Sleator, editors, *On-Line Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, February 11-13, 1991*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 79–84. DIMACS/AMS, 1991. doi:10.1090/dimacs/007/05.

19 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update rules. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 – May 2, 1984, Washington, DC, USA*, pages 488–492. ACM, 1984. doi:10.1145/800057.808718.