Online Knapsack with Removal and Recourse

Hans-Joachim Böckenhauer, Ralf Klasing, Tobias Mömke, Peter Rossmanith, Moritz Stocker, David Wehner

Online Knapsack with Removal and Recourse

Hans-Joachim Böckenhauer[1], Ralf Klasing[2⋆], Tobias Mömke[3⋆⋆], Peter Rossmanith[4], Moritz Stocker[1], and David Wehner[1]

[1] Department of Computer Science, ETH Zürich, Zürich, Switzerland
{hjb,moritz.stocker,david.wehner}@inf.ethz.ch
[2] CNRS, LaBRI, Université de Bordeaux, Talence, France
ralf.klasing@labri.fr
[3] Institute of Computer Science, University of Augsburg, Augsburg, Germany
moemke@informatik.uni-augsburg.de
[4] Departement of Computer Science, RWTH Aachen University, Aachen, Germany
rossmani@cs.rwth-aachen.de

Abstract. We analyze the proportional online knapsack problem with removal and limited recourse. The input is a sequence of item sizes; a subset of the items has to be packed into a knapsack of unit capacity such as to maximize their total size while not exceeding the knapsack capacity. In contrast to the classical online knapsack problem, packed items can be removed and a limited number of removed items can be re-inserted to the knapsack. Such re-insertion is called *recourse*. Without recourse, the competitive ratio is known to be approximately 1.618 (Iwama and Taketomi, ICALP 2002). We show that, even for only one use of recourse for the whole instance, the competitive ratio drops to 3/2. We prove that, with a constant number of $k \geq 2$ uses of recourse, a competitive ratio of $1/\left(\sqrt{3}-1\right) \leq 1.367$ can be achieved and we give a lower bound of $1+1/(k+1)$ for this case. For an extended use of recourse, i.e., allowing a constant number of $k \geq 1$ uses per step, we derive tight bounds for the competitive ratio of the problem, lying between $1+1/(k+2)$ and $1+1/(k+1)$. Motivated by the observation that the lower bounds heavily depend on the fact that the online algorithm does not know the end of the input sequence, we look at a scenario where an algorithm is informed when the instance ends. We show that with this information, the competitive ratio for a constant number of $k \geq 2$ uses of recourse can be improved to strictly less than $1+1/(k+1)$. We also show that this information improves the competitive ratio for one use of recourse per step and give a lower bound of ≥ 1.088 and an upper bound of 4/3 in this case.

Keywords: online knapsack · proportional knapsack · recourse · semi-online algorithm

⋆ Partially supported by the ANR project TEMPOGRAL (ANR-22-CE48-0001).
⋆⋆ Partially supported by DFG Grant 439522729 (Heisenberg-Grant) and DFG Grant 439637648 (Sachbeihilfe).

1 Introduction

In the classical knapsack problem, we are given a knapsack of capacity B and a set of items, each of which has a size and a value. The goal is to pack a subset of items such that the total size does not exceed the capacity B, maximizing the value of the packed items. In the *proportional* variant of the problem (also called *unweighted* or *simple* knapsack problem), the size and the value of each item coincide. Stated as an online problem, B is given upfront, but the items are revealed one by one in a sequence of requests. The online algorithm has to decide whether the requested item is packed or discarded before the subsequent item arrives and cannot revoke the decision. To measure the quality of the solution, we use the competitive ratio, i.e., the value attainable by an optimal offline algorithm divided by the value of the solution computed by the online algorithm in the worst case. It is well-known that, even in the proportional variant, no online algorithm for the online knapsack problem achieves a bounded competitive ratio [20]. This indicates that the classical notion of online algorithms is overly restrictive for the knapsack problem. Unless otherwise stated, we focus on the proportional variant in this paper.

In the literature, several ways of relaxing the online requirements have been considered for various online problems, leading to so-called *semi-online* problems [7,10]. Such semi-online problems enable to study the effect of different degrees of online behavior on the hardness of computational problems. Semi-online problems can be roughly divided into two classes: On the one hand, one can equip the algorithm with some extra information about the instance, e.g., the size of an optimal solution. On the other hand, one can relax the irrevocability of the algorithm's decisions. Several models from the second class have already been considered for the online knapsack problem: In the model of *delayed decisions* [23], one grants the online algorithm the right to postpone its decisions until they are really necessary. This means that the algorithm is allowed to temporarily pack items into the knapsack as long as its capacity allows and to remove them later on to avoid overpacking of the knapsack [17]. In the *reservation model*, the algorithm has the option to reserve some items in an extra storage at some extra cost, with the possibility of packing them into the knapsack later [2].

In this paper, we consider a semi-online model called *recourse*. In the model of recourse, the algorithm is allowed to withdraw a limited number of its previous decisions. Recourse has mainly been studied for the Steiner tree problem, MST, and matchings [16,13,22,12,21,1,6]. In case of the knapsack problem, we distinguish two types of recourse: (i) An item that has been selected previously is discarded (to make space for a new item); and (ii) an item was previously discarded and is added afterwards. The second type of recourse is costlier than the first type, as an unlimited number of items has to stay at disposal.

Applying the first type of recourse directly to the classical online knapsack problem does not get us too far: The same hard example as for the problem without recourse, i.e., one instance consisting of the items ε and 1 (for some arbitrarily small $\varepsilon > 0$) and another one consisting of the ε only, also proves

	Upper bound		Lower bound	
1 recourse in total	$\frac{3}{2}$	(Theorem 2)	$\frac{3}{2}$	(Theorem 1)
$k \geq 2$ recourses in total	$\frac{1}{\sqrt{3}-1} \leq 1.367$	(Theorem 3)	$1 + \frac{1}{k+1}$	(Theorem 1)
k recourses per step	$f(k) \leq 1 + \frac{1}{k+1}$	(Theorem 4)	$f(k) \geq 1 + \frac{1}{k+2}$	(Theorem 5)

Table 1. Our results on the competitive ratio for online knapsack with removal and limited recourse. Here, $f(k) = 2/(\sqrt{k^2+6k+5} - k - 1)$.

	Upper bound		Lower bound	
$k \geq 2$ recourses in total	$f(k) \leq 1 + \frac{1}{k+1}$	(Theorem 6)		
1 recourse per step	$\frac{4}{3}$	(Theorem 8)	$\frac{18}{5+\sqrt{133}} \geq 1.088$	(Theorem 7)

Table 2. Our results on the competitive ratio for online knapsack with removal and limited recourse, given information on the end of an instance. Here, $f(k) = 2/(\sqrt{k^2+6k+5} - k - 1)$.

an unlimited competitive ratio here since discarding the first item makes the instance stop and does not leave room for any recourse.

We combine the option of unlimited removal with limited recourse, that is, a limited number of re-packings of discarded items. The resulting upper and lower bounds on the competitive ratio are shown in Table 1. Classically, in the online model, upon arrival of an item, the algorithm does not know whether this will be the last item in the sequence or not. Besides analyzing this standard model, we additionally consider various ways of communicating information about the end of the sequence to the algorithm. The problem exhibits a surprisingly rich structure with respect to this parameter. Our respective bounds are shown in Table 2. Due to space constraints, some proofs are omitted in this paper.

1.1 Preliminaries

In the online knapsack problem OnlineKP used in this paper, an instance is given as a sequence of items $I = (x_1, \ldots, x_n)$. For convenience of notation, we identify an item with its size. In each step $1 \leq i \leq n$, an algorithm receives the item $x_i > 0$. At this point, the algorithm has no knowledge of the items $(x_{i+1}, \ldots, x_n)$ and no knowledge of the length n of the instance. It maintains a *knapsack* $S \subseteq I$ such that, in each step, the items in the knapsack do not exceed the capacity of the knapsack. We normalize this size to 1 and thus assume that $x_i \in [0, 1]$ for all i. All items x_j for $j < i$ that are not in S are considered to be in a *buffer* of infinite size.

In the framework of this paper, given the item x_i, an algorithm first adds it to the knapsack S, potentially exceeding the size limit. It may then *remove* any number of items from S, moving them to the buffer, and possibly return a certain number of items from the buffer to S afterwards, expending one use of

recourse for each returned item. For simplicity, we say that the algorithm *packs* an item if it is kept in the knapsack upon its arrival, and that it *discards* an item if it is immediately removed in the step of its arrival. The number of such uses available to the algorithm will vary between various scenarios. After this process, the condition $\sum_{x_i \in S} x_i \leq 1$ must hold, i.e., the selected items respect the capacity of the knapsack. In our algorithms, we frequently classify the items by their sizes: For a bound b with $1/2 < b < 1$, we call an item *small* if it is of size at most $1 - b$, *medium* if its size is greater than $1 - b$ but smaller than b, and *large* if its size is at least b.

The *gain* $\text{gain}_{\text{ALG}}(I)$ of the algorithm ALG on the instance I is given as the sum $\sum_{x_i \in S} x_i$ of the item sizes in S after the last step n. Its *strict competitive ratio* $\rho_{\text{ALG}}(I)$ on I is defined as $\rho_{\text{ALG}}(I) = \text{gain}_{\text{OPT}}(I)/\text{gain}_{\text{ALG}}(I)$. The smaller the ratio, the better the algorithm performs. The strict competitive ratio ρ_{ALG} of the algorithm is then defined as the worst case over all possible instances, $\rho_{\text{ALG}} = \sup_I \rho_{\text{ALG}}(I)$. This definition is often generalized to the *competitive ratio* of an algorithm, which allows for a constant additive term to be added to $\text{gain}_{\text{ALG}}(I)$ in the definition of $\rho_{\text{ALG}}(I)$. However, since in OnlineKP the optimal solution for any instance is bounded by 1, this relaxation does not add any benefit to the analysis of this problem. We will therefore only consider the strict version and refer to it simply as *competitive ratio*.

1.2 Related Work

Online problems with recourse date back to Imase and Waxman [16] who have studied the online Steiner tree problem and utilized the benefit of a limited number of rearrangements. The number of required recourse steps was subsequently reduced by a sequence of papers [13,22,12]. Recently, recourse was considered for further problems, in particular online matching [21,1,6].

Many different kinds of semi-online problems have been considered in the literature; Boyar et al. [7] give an overview of some of these. In particular, many results on semi-online algorithms focus on makespan scheduling problems with some extra information, starting with the work by Kellerer et al. [18]; see the survey by Dwibety and Mohanty [10] for a recent overview of this line of research.

Many semi-online settings assume the availability of some extra information, e.g., the total makespan in scheduling problems. In the model of *advice complexity*, one tries to measure the performance of an online algorithm in the amount of any information conveyed by some oracle that knows the whole input in advance. This very general approach to semi-onlineness provides a powerful tool for proving lower bounds. The model was introduced by Dobrev et al. [9] in 2008 and shortly afterwards revised by Emek et al. [11], Böckenhauer et al. [4], and Hromkovič et al. [15].

Since then, it has been applied to many different online problems; for a survey, see the work by Boyar et al. [7] and the textbook by Komm [19].

In this paper, we consider a slightly different kind of semi-online problems, where the online condition is not relaxed by giving some extra information to the algorithm, but by relaxing the irrevocability of its decisions. In one approach,

the online algorithm is allowed to delay its decisions until there is a real need for it. For instance, in the knapsack problem, the algorithm is allowed to pack all items into the knapsack until it is overpacked, and only then has to decide which items to remove. Iwama and Taketomi [17] gave the first results for online knapsack with removal. A version in which the removal is not completely for free, but induces some extra cost was studied by Han et al. [14]. *Delayed decisions* were also studied for other online problems by Rossmanith [23] and by Chen et al. [8]. Böckenhauer et al. [2] analyzed another semi-online version of online knapsack which gives the algorithm a third option besides packing or rejecting an item, namely to reserve it for possible later packing at some reservation cost. The advice complexity of online knapsack was analyzed by Böckenhauer et al. [5] in the normal model and later in the model with removal [3].

2 Number of Uses of Recourse Bounded per Instance

In this section, we analyze the scenario in which an algorithm can only use recourse a limited number of $k \geq 1$ times in total. Even if we just allow one use of recourse per instance, the upper bound of $(\sqrt{5}+1)/2 \approx 1.618$ proven by Iwama and Taketomi [17] for the online knapsack problem with removal (but without recourse) can be improved, as we show in the following. We find a lower bound of $1 + 1/(k+1)$ for the competitive ratio of any algorithm. In the case $k = 1$ where the algorithm can use recourse exactly once, we present an algorithm that matches this lower bound of $3/2$. In the case $k > 1$, we find an upper bound of $1/(\sqrt{3}-1) \leq 1.367$ that does not improve as k gets larger.

2.1 Lower Bound

Theorem 1. *Any algorithm that uses recourse at most $k \geq 1$ times in total cannot have a competitive ratio of less than $1 + 1/(k+1)$.*

Proof. We present a family of instances dependent on $\varepsilon > 0$, such that any algorithm that uses at most k recourses in total cannot have a competitive ratio of less than $(k+2)/(k+1)$ for at least one of these instances in the limit $\varepsilon \to 0$. These instances are given in Table 3; they all start with k copies of the item $x_1 = \frac{1}{k+2} + \varepsilon$. For the proof to work as intended, ε is chosen such that $0 < \varepsilon < \frac{1}{(k+2)^2}$. Note that any deterministic algorithm must act identically on these instances up to the point where they differ.

Now, let ALG be any algorithm that uses at most k recourses in total and assume that it has a competitive ratio strictly less than $(k+2)/(k+1)$.

1. The algorithm must pack item x_2 in each instance, removing all previously packed copies of x_1: otherwise, its competitive ratio on instance I_1 is at least

$$\rho_{\text{ALG}} \geq \frac{(k+1)/(k+2) + (k+2)\varepsilon}{k \cdot (1/(k+2) + \varepsilon)} \to \frac{k+1}{k} > \frac{k+2}{k+1} \text{ as } \varepsilon \to 0\,.$$

	$\underbrace{\quad}_{k \text{ copies}}$					
I_1	x_1	x_2				
I_2	x_1	x_2	x_3			
I_3	x_1	x_2	x_3	$y_3 = \frac{1}{k+2} - (k+1)\varepsilon$		
I_4	x_1	x_2	x_3	$x_4 = \frac{k+1}{k+2} + \varepsilon$	$y_4 = \frac{1}{k+2} - \varepsilon$	
I_5	x_1	x_2	x_3	$x_4 = \frac{k+1}{k+2} + \varepsilon$	$x_5 = \frac{1}{k+2}$	$y_5 = \frac{k+1}{k+2} - \varepsilon$
I_6	x_1	x_2	x_3	$x_4 = \frac{k+1}{k+2} + \varepsilon$	$x_5 = \frac{1}{k+2}$	

Table 3. A family of instances showing that no algorithm that uses at most k recourses in total can achieve a competitive ratio better than $(k+2)/(k+1)$. All instances start with k items of size x_1, where $x_1 = x_3 = \frac{1}{k+2} + \varepsilon$ and $x_2 = \frac{k+1}{k+2} + (k+2)\varepsilon$.

2. The algorithm must then pack the item x_3, remove x_2 and use its entire recourse to retrieve the k copies of x_1 in instances I_2 to I_5:
   - If it packs item x_3 but only uses its recourse to retrieve $m < k$ copies of x_1, its competitive ratio on instance I_2 is at least
   $$\rho_{\text{ALG}} \geq \frac{\frac{k+1}{k+2} + (k+2)\varepsilon}{(m+1)(\frac{1}{k+2} + \varepsilon)} \geq \frac{\frac{k+1}{k+2} + (k+2)\varepsilon}{k(\frac{1}{k+2} + \varepsilon)} \to \frac{k+1}{k} > \frac{k+2}{k+1} \text{ as } \varepsilon \to 0\,.$$
   - If it does not pack item x_3 and keeps x_2, its competitive ratio on instance I_3 is at least
   $$\rho_{\text{ALG}} \geq \frac{1}{\frac{k+1}{k+2} + (k+2)\varepsilon} \to \frac{k+2}{k+1} \text{ as } \varepsilon \to 0\,.$$

   So, from here on, the algorithm cannot use any further recourse.
3. The algorithm must then pack item x_4 in instances I_4 to I_6, removing x_3 and all copies of x_1: otherwise, its competitive ratio on instance I_4 is at least
$$\rho_{\text{ALG}} \geq \frac{1}{\frac{k+1}{k+2} + (k+2)\varepsilon} \to \frac{k+2}{k+1} \text{ as } \varepsilon \to 0\,.$$
4. The algorithm must then pack item x_5 and remove x_4 in instances I_5 and I_6: otherwise, its competitive ratio on instance I_5 is at least
$$\rho_{\text{ALG}} \geq \frac{1}{\frac{k+1}{k+2} + \varepsilon} \to \frac{k+2}{k+1} \text{ as } \varepsilon \to 0\,.$$
5. However, in this situation, its competitive ratio on instance I_6 is at least
$$\rho_{\text{ALG}} \geq \frac{\frac{k+1}{k+2} + (k+2)\varepsilon}{\frac{1}{k+2}} \to k+1 > \frac{k+2}{k+1} \text{ as } \varepsilon \to 0\,.$$

Hence, ALG has a competitive ratio of at least $(k+2)/(k+1) = 1 + 1/(k+1)$. □

2.2 Upper Bound

Upper Bound for $k = 1$. We present an algorithm ALG_1 that uses recourse at most once and that achieves a competitive ratio of 3/2. We set $b = 2/3$ and distinguish between small, medium, and large items with respect to b as described in Subsection 1.1. The algorithm ALG_1

- packs any large item immediately, to this end removes some items from the knapsack if necessary, and discards all other items from there on;
- packs any small item immediately. If a small item does not fit or has to be discarded to fit a medium item, it discards all other items from there on;
- always keeps the largest medium item. If it encounters a medium item x_i that fits together with a previously encountered medium item x_j, it removes the currently packed one, uses its recourse to retrieve x_j and discards all other items from there on.

Theorem 2. *The algorithm* ALG_1 *has a competitive ratio of at most* 3/2.

Proof. We prove that ALG_1 is either optimal or achieves a gain of at least 2/3 and thus a competitive ratio of at most 3/2. If there is a large item in the instance, the algorithm packs it, leading to a gain of at least 2/3. We can therefore assume that the instance contains no large items.

If the algorithm discards a small item at any point due to size limits, its gain at that point must be at least $1 - 1/3 = 2/3$. We can therefore assume that the algorithm never discards a small item.

Now, consider the number of medium items in the optimal solution on the instance, which cannot be more than two. If it contains exactly two, the algorithm can pack two medium items, leading to a gain of at least $1/3 + 1/3 = 2/3$. If the optimal solution contains zero or one medium item, the algorithm packs all small items and (if there is one) the largest medium item. Since all small items in the instance are packed by assumption, the algorithm is optimal in this case. □

Upper Bound for $k > 1$. We define b as the unique positive solution of the equation $b^2 + 2b - 2 = 0$, so $b = \sqrt{3} - 1 \approx 0.73$. We present an algorithm ALG_2 with a competitive ratio of at most $1/b \leq 1.367$ that uses recourse at most twice. The algorithm again distinguishes between small, medium, and large items with respect to b.

- The algorithm ALG_2 treats any small or large item the same as ALG_1.
- The algorithm only keeps the largest medium item, until it is presented with a medium item that will fit with one that has already been encountered. If that is the case, it will spend one use of its recourse to retrieve that item if it is in the buffer. From there on, it will keep the two smallest medium items encountered so far. (i) If it encounters a third medium item that will fit with these two, it will pack it and discard everything else from there on. (ii) If, however, at any point the algorithm encounters a medium item x_i, such that there is a previously encountered medium item x_j with $b \leq x_i + x_j \leq 1$, it packs x_i, spends one use of recourse to retrieve x_j if it is in the buffer and discards everything else from there on.

– If at any point a medium item does not fit because of small items that have already been packed, the algorithm removes small items one by one until the medium item can be packed and discards everything else from there on.

Theorem 3. *The algorithm* $\textsc{Alg}_2$ *has a competitive ratio of at most* $1/(\sqrt{3}-1)$.

3 Number of Uses of Recourse Bounded per Step

In this scenario, an algorithm could use recourse a limited number of k times per step. We give sharp bounds on the competitive ratio of an optimal algorithm in this case, tending to 1 when k tends to infinity. Let b_k be the unique positive root of the quadratic equation $b_k^2 + (k+1) \cdot b_k - (k+1) = 0$. Then it is easy to check that $1 + 1/(k+1) \leq 1/b_k \leq 1 + 1/(k+2)$.

3.1 Upper Bound

Let $k \in \mathbb{N}$. We define $b_k = (\sqrt{k^2+6k+5} - k - 1)/2$ as the unique positive solution of the quadratic equation $b_k^2 + (k+1) \cdot b_k - (k+1) = 0$. We construct an algorithm $\textsc{Alg}_k$ with a competitive ratio of at most $1/b_k$ that uses recourse at most k times per step, again distinguishing between small, medium, and large items with respect to b_k.

– The algorithm $\textsc{Alg}_k$ treats any small or large item the same as $\textsc{Alg}_1$ in Subsection 2.2.
– As long as at most k medium items fit, $\textsc{Alg}_k$ packs them optimally, using its recourse to do so. As soon as it is presented with a medium item that will fit with k previous ones, it will use its recourse to retrieve any of these that are in the buffer. From there on, it will keep the $k+1$ smallest medium items encountered so far. (i) If it encounters an additional medium item that will fit with these $k+1$, it will pack it and discard everything else from there on. (ii) If however at any point the algorithm encounters a medium item that fits with at most k previously encountered ones, such that their sum is at least b_k, it packs it, uses its recourse to retrieve any of the others that might be in the buffer and discards everything else from there on.
– If at any point a medium item does not fit because of small items that have already been packed, $\textsc{Alg}_k$ removes small items one by one until the medium item can be packed and discards everything else from there on.

In the case $k = 1$, this is the algorithm $\textsc{Alg}_2$ that is used in Subsection 2.2 for $k > 1$ uses of recourse in total. The algorithm not only uses recourse at most twice but never spends both uses in the same step.

Theorem 4. *The algorithm* $\textsc{Alg}_k$ *has a competitive ratio of at most* $1/b_k \leq 1 + \frac{1}{k+1}$.

3.2 Lower Bound

We now prove that the algorithm provided in Subsection 3.1 is the best possible. Let b_k be defined as in Subsection 3.1.

Theorem 5. *Any algorithm that uses recourse at most k times per step cannot have a competitive ratio of less than $1/b_k \geq 1 + \frac{1}{k+2}$.*

Proof (Sketch). We define $a_k = (1 - b_k) + \varepsilon$ and present two instances $I_1 = (a_k, \ldots, a_k, b_k + (k+2)\varepsilon)$ and $I_2 = (a_k, \ldots, a_k, b_k + (k+2)\varepsilon, 1 - (k+1)a_k)$. Both instances start with $k+1$ items of size a_k. The competitive ratio of any algorithm must be at least $1/b_k$ on at least one of these instances. □

4 Information on the End of the Instance

Previously, all problems were defined in a way where the algorithm had no information on whether a certain item was the last item of the instance or not. It might be natural to allow an algorithm access to this information. In the situation where no recourse is allowed, this distinction does not matter: Any instance could be followed by a final item of size 0, in which case removing any items would not lead to a better solution. With recourse however, this might change.

4.1 Two Different Ways for Handling End of Instance

There appear to be two different ways in which the information that the instance ends might be encoded. One one hand, the instance might be given in the form $(x_1, \ldots, x_n, \perp)$ where $\perp$ informs the algorithm that the previous item was the last one, allowing it to perform one last round of removal and recourse. On the other hand, the instance could be given in the form $(x_1, \ldots, x_{n-1}, (x_n, \perp))$, where the algorithm is informed that an item is the last of the instance in the same step that the item is given. It can be shown that an algorithm will always perform at least as well if it receives the information in the former variant than in the latter. However, in the scenario where the size of the recourse is bounded by k uses in total, the chosen variant does not matter.

4.2 Upper Bound for k Uses of Recourse in Total Given Information on the End of the Instance

When information about the end of an instance is available, k uses of recourse in total are at least as useful as k uses of recourse per step without that information. Since the way that information is received does not matter as mentioned in Subsection 4.1 we will assume that the instance is given in the form $(x_1, \ldots, x_n, \perp)$.

We can now adapt the algorithm $\textsc{Alg}_k$ from Subsection 3.1 to this situation, where b_k is again defined as the unique positive solution of the quadratic equation $b_k^2 + (k+1) \cdot b_k - (k+1) = 0$ and define small, medium and large items accordingly. The obtained algorithm $\textsc{Alg}_{(k,\perp)}$ works as follows.

– It treats any small or large item the same as $\textsc{Alg}_1$ in Subsection 2.2.
– It only keeps the smallest medium item unless it can achieve a gain of at least b_k using only medium items. In this case, as in the algorithm in Subsection 3.1, it retrieves at most k items and discards everything from there on.
– If the algorithm receives the item $\perp$ and has not yet decided to discard everything else, it computes the optimal solution on the entire previous instance, retrieves all medium items in that solution using its recourse and discards all small items not contained in that solution.

Theorem 6. *The algorithm* $\textsc{Alg}_{(k,\perp)}$ *has a competitive ratio of at most* $1/b_k$.

4.3 Lower Bound for One Use of Recourse per Step Given Information on the End of the Instance

We will assume that the algorithm handles instances of the form $(x_1, \ldots, (x_n, \perp))$.

Theorem 7. *No algorithm that uses recourse at most once per step and that recognizes the last item in an instance can have a competitive ratio of less than* $18/(5+\sqrt{133}) \geq 1.088$.

Proof (Sketch). We define b as the unique positive root of the equation $27b^2 - 5b - 1 = 0$, so $b = (5+\sqrt{133})/54 \approx 0.3062$. We further define the additional item $a = (1-b)/3+\varepsilon \approx 0.2313+\varepsilon$ and present two instances $I_1 = (a, a, a, b, b, b, 1-3a)$ and $I_2 = (a, a, a, b, b, b, \varepsilon)$. The competitive ratio of any algorithm must be at least $1/(3b)$ on at least one of these instances in the limit $\varepsilon \to 0$. □

4.4 Upper Bound for One Use of Recourse per Step Given Information on the End of the Instance

We assume that the instance is given in the form $(x_1, \ldots, (x_n, \perp))$ and present an algorithm $\textsc{Alg}_\perp$ with a competitive ratio of $4/3 \approx 1.33$, which is strictly better than the optimal algorithm without that information in Subsection 3.1. We define $b = 3/4$ and distinguish between small, medium and large items, depending on b as before. The algorithm $\textsc{Alg}_\perp$ then works as follows.

– It treats any small or large item the same as $\textsc{Alg}_1$ in Subsection 2.2.
– It packs medium items as follows. (i) As long as only one medium item fits, it keeps the smallest one of these. (ii) As soon as two medium items fit, it computes the optimal sum of two medium items and keeps the larger of these two, as well as the smallest medium item encountered so far. (iii) When it encounters a medium item that fits with two previously encountered ones (one being w.l.o.g. the smallest medium item, currently packed), it packs it, retrieves the third of these three and discards everything from there on.
– If it encounters the item $(x_n, \perp)$ and has not yet decided to discard everything else, it computes the optimal solution on the entire instance. (i) If this solution contains only one medium item, it retrieves it and returns the optimal solution. (ii) If this solution contains two medium items, one of these must be either

x_n or already packed. The algorithm retrieves the other one and returns the optimal solution. (iii) If this solution contains three medium items, x_n must be a medium item and the algorithm proceeds as if it was not the last one.

Theorem 8. *The algorithm* $\textsc{Alg}_\perp$ *has a competitive ratio of at most* $4/3$.

5 Conclusion

Besides closing the gap between the upper and lower bounds for $k \geq 2$ uses of recourse in total without knowing the end of the instance and proving a lower bound in the case of knowing the end, it is an interesting open problem to consider a model in which the use of removal or recourse is not granted for free, but incurs some cost for the algorithm.

In the *general online knapsack problem*, the items have both a size and a value, and the goal is to maximize the value of the packed items while obeying the knapsack bound regarding their sizes. If we consider the general online knapsack problem with removal and one use of recourse per step, we can easily see that the competitive ratio is unbounded: Suppose there is an online algorithm with competitive ratio c. The adversary then presents first an item of size and value 1, if the algorithm does not take this item, the instance ends. Otherwise, it presents up to $(c+1)/\varepsilon$ many items of size $\varepsilon/(c+1)$ and value ε. If the online algorithm decides at some point to take such an item of value ε and uses its recourse to fetch another of these from the buffer, the instance ends and the competitive ratio is $1/(2\varepsilon)$. A similar argument shows that a recourse of size k per step cannot avoid an unbounded competitive ratio either. But, again, these arguments heavily depend on the algorithm's unawareness of the end of the instance. It remains as an open problem to extend the results for known instance lengths to the general case.

References

1. S. Angelopoulos, C. Dürr, and S. Jin. Online maximum matching with recourse. *J. Comb. Optim.*, 40(4):974–1007, 2020.
2. H.-J. Böckenhauer, E. Burjons, J. Hromkovič, H. Lotze, and P. Rossmanith. Online simple knapsack with reservation costs. In M. Bläser and B. 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.
3. H.-J. Böckenhauer, J. Dreier, F. Frei, and P. Rossmanith. Advice for online knapsack with removable items. *CoRR*, abs/2005.01867, 2020.
4. H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, and T. Mömke. On the advice complexity of online problems. In *20th International Symposium on Algorithms and Computation, ISAAC 2009*, volume 5878 of *Lecture Notes in Computer Science*, pages 331–340, 2009.
5. H.-J. Böckenhauer, D. Komm, R. Královič, and P. Rossmanith. The online knapsack problem: Advice and randomization. *Theor. Comput. Sci.*, 527:61–72, 2014.

6. J. Boyar, L. M. Favrholdt, M. Kotrbcík, and K. S. Larsen. Relaxing the irrevocability requirement for online graph algorithms. *Algorithmica*, 84(7):1916–1951, 2022.
7. J. Boyar, L. M. Favrholdt, C. Kudahl, K. S. Larsen, and J. W. Mikkelsen. Online algorithms with advice: A survey. *ACM Comput. Surv.*, 50(2):19:1–19:34, 2017.
8. L. Chen, L. Hung, H. Lotze, and P. Rossmanith. Online node- and edge-deletion problems with advice. *Algorithmica*, 83(9):2719–2753, 2021.
9. S. Dobrev, R. Královič, and D. Pardubská. How much information about the future is needed? In V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, and M. Bieliková, editors, *SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 19-25, 2008, Proceedings*, volume 4910 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 2008.
10. D. Dwibedy and R. Mohanty. Semi-online scheduling: A survey. *Comput. Oper. Res.*, 139:105646, 2022.
11. Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. *Theor. Comput. Sci.*, 412(24):2642–2656, 2011.
12. A. Gu, A. Gupta, and A. Kumar. The power of deferral: Maintaining a constant-competitive steiner tree online. *SIAM J. Comput.*, 45(1):1–28, 2016.
13. A. Gupta and A. Kumar. Online steiner tree with deletions. In *SODA*, pages 455–467. SIAM, 2014.
14. X. Han, Y. Kawase, and K. Makino. Online unweighted knapsack problem with removal cost. *Algorithmica*, 70(1):76–91, 2014.
15. J. Hromkovič, R. Královič, and R. Královič. Information complexity of online problems. In P. Hlinený and A. 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.
16. M. Imase and B. M. Waxman. Dynamic steiner tree problem. *SIAM J. Discret. Math.*, 4(3):369–384, 1991.
17. K. Iwama and S. Taketomi. Removable online knapsack problems. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. J. Eidenbenz, and R. 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.
18. H. Kellerer, V. Kotov, M. G. Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Oper. Res. Lett.*, 21(5):235–242, 1997.
19. D. Komm. *An Introduction to Online Computation - Determinism, Randomization, Advice*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.
20. A. Marchetti-Spaccamela and C. Vercellis. Stochastic on-line knapsack problems. *Math. Program.*, 68:73–104, 1995.
21. N. Megow and L. Nölke. Online minimum cost matching with recourse on the line. In *APPROX-RANDOM*, volume 176 of *LIPIcs*, pages 37:1–37:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
22. N. Megow, M. Skutella, J. Verschae, and A. Wiese. The power of recourse for online MST and TSP. *SIAM J. Comput.*, 45(3):859–880, 2016.
23. P. Rossmanith. On the advice complexity of online edge- and node-deletion problems. In H.-J. Böckenhauer, D. Komm, and W. Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, volume 11011 of *Lecture Notes in Computer Science*, pages 449–462. Springer, 2018.