Journal of Computer and System Sciences 155 (2026) 103697

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

journal homepage: www.elsevier.com/locate/jcss

Online knapsack with removal and recourse ☆

Hans-Joachim Böckenhauer [a], Ralf Klasing [b,1], Tobias Mömke [c,2], Peter Rossmanith [d], Moritz Stocker [a,*], David Wehner [a]

[a] Department of Computer Science, ETH Zurich, Universitätsstrasse 6, 8092 Zurich, Switzerland
[b] CNRS, LaBRI, Université de Bordeaux, 351, cours de la Libération, 33405 Talence CEDEX, France
[c] Institute of Computer Science, University of Augsburg, Universitätsstraße 6a, 86159 Augsburg, Germany
[d] Departement of Computer Science, RWTH Aachen University, Ahornstraße 55, 52056 Aachen, Germany

ARTICLE INFO

Article history:
Received 24 January 2024
Received in revised form 28 February 2025
Accepted 24 July 2025
Available online 30 July 2025

Keywords:
Online knapsack
Proportional knapsack
Removal
Recourse
Semi-online algorithm

ABSTRACT

We analyze the competitive ratio of the proportional online knapsack problem with removal and limited recourse. 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. The variant with removal only was analyzed by Iwama and Taketomi (ICALP, 2002). We show that even a single use of recourse can improve the performance of an algorithm. We give lower bounds for a constant number of $k \geq 1$ uses of recourse in total, matching upper bounds for $1 \leq k \leq 3$, and a general upper bound for any value of k. For a variant where a constant number of $k \geq 1$ uses of recourse can be used per step, we give tight bounds for all $k \geq 1$. We further look at a scenario where an algorithm is informed when the instance ends and give improved upper bounds in both variants for this case.

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, that is, 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 [2]. 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.

☆ An extended abstract of this paper appeared at IWOCA 2023 [1].

* Corresponding author.
E-mail addresses: hjb@inf.ethz.ch (H.-J. Böckenhauer), ralf.klasing@labri.fr (R. Klasing), moemke@informatik.uni-augsburg.de (T. Mömke), rossmani@cs.rwth-aachen.de (P. Rossmanith), moritz.stocker@inf.ethz.ch (M. Stocker), david.wehner@inf.ethz.ch (D. Wehner).

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

https://doi.org/10.1016/j.jcss.2025.103697

Table 1
Results on the competitive ratio of the online knapsack problem with removal and limited recourse. The function $f(k)$ is given explicitly by $f(k) = (\sqrt{k^2+6k+5}+k+1)/(2k+2)$. These results are illustrated in Fig. 1.

Uses of recourse	Upper bound		Lower bound	
$k \le 3$ in total	$1+\frac{1}{k+1}$	(Theorems 2, 3, 4)	$1+\frac{1}{k+1}$	(Theorem 1)
$4 \le k \le 12$ in total	5/4	(Theorem 4)	$1+\frac{1}{k+1}$	(Theorem 1)
$k \ge 13$ in total	$1+\frac{1}{\sqrt{2k}-1}$	(Theorem 5)	$1+\frac{1}{k+1}$	(Theorem 1)
k per step	$f(k) \le 1+\frac{1}{k+1}$	(Theorem 6)	$f(k) \ge 1+\frac{1}{k+2}$	(Theorem 7)

Table 2
Results on the competitive ratio of the online knapsack problem with removal and limited recourse, as well as information on the end of an instance. Instances are given in the form $(x_1, \ldots, x_{n-1}, (x_n, \perp))$, marking the last item. These results are illustrated in Fig. 1.

Uses of recourse	Upper bound		Lower bound	
$k \ge 1$ in total	$1+\frac{1}{k+2\sqrt{k}}$	(Theorem 9)	$1+\frac{1}{4k^2+4k+3}$	(Theorem 8)
$k \ge 1$ per step	$1+\frac{1}{k+2\sqrt{k}+0.546}$	(Theorem 10)	$1+\frac{1}{4k^2+4k+3}$	(Theorem 8)

In the literature, several ways of relaxing the online requirements have been considered for various online problems, leading to so-called *semi-online* problems [3,4]. Such semi-online problems enable us 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* [5], 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 [6]. 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 [7].

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, minimal spanning tree problem, and matchings [8–14]. 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.

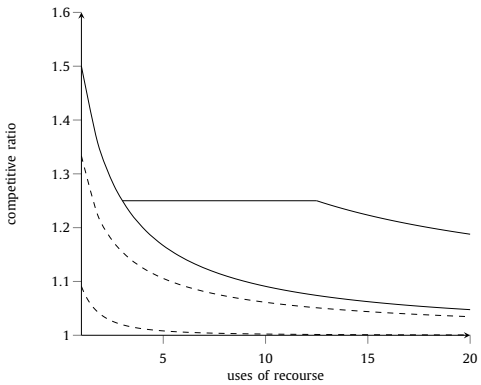
The first type of recourse is known as removal and has been studied extensively by Iwama and Taketomi [6] for the proportional model, who showed that the competitive ratio of the problem is exactly the golden ratio $\varphi \approx 1.618$. Han et al. [15] showed that this can be lowered to no more than 10/7 using randomization, but their bounds are not tight. In this paper, we only consider deterministic algorithms. Other researchers have considered the problem going beyond the proportional knapsack [16–18].

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

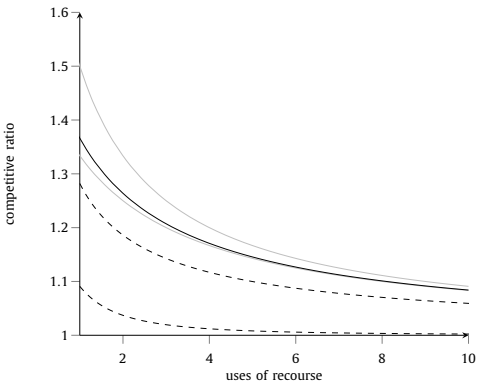
In this paper, we reserve the term "recourse" for the second type of recourse, and call the first type of recourse "removal", following the terminology introduced by Iwama and Taketomi [6]. We combine the option of unlimited removal with limited recourse, that is, a limited number of re-packings of discarded items. The resulting upper and lower bounds on the competitive ratio are shown in Table 1 and illustrated in Fig. 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 two different 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 and illustrated in Fig. 1.

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 $1 \le i \le n$, 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.



(a) Use of recourse bounded per instance.



(b) Use of recourse bounded per step. For comparison, the functions $1+1/(k+1)$ and $1+1/(k+2)$ are shown in gray.

Fig. 1. Competitivity of the Proportional Knapsack Problem with Removal and Recourse. Results from Table 1 are shown in solid lines, while results from Table 2 are shown in dashed lines.

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. The number of times an algorithm can use recourse will vary between various scenarios. 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. After this process, the condition $\sum_{x_i \in S} x_i \leq 1$ must hold, that is, the selected items respect the capacity of the knapsack.

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 $\rho_{\text{ALG}}(I) = \text{gain}_{\text{OPT}}(I)/\text{gain}_{\text{ALG}}(I)$, where $\text{gain}_{\text{OPT}}(I)$ is the gain of an optimal offline algorithm, that is, an algorithm that knows the entire instance in advance. 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, any algorithm would have a competitive ratio of 1 if this were allowed. Therefore, this relaxation does not add any benefit to the analysis of this problem. We will only consider the strict version and refer to it simply as *competitive ratio*.

1.2. General behavior of algorithms

All algorithms presented in this paper will follow a similar framework. We will usually classify 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.

We say that an algorithm *stops* if it maintains its current configuration and discards all future items. The focus of our algorithms will always be on medium items. All algorithms will follow the same rules regarding small and large items in the following order of priority:

1. Pack any large item, removing any previously packed items in the knapsack, and stop.
2. Pack any small item. If a small item cannot be packed without exceeding the capacity of the knapsack, discard it and stop.
3. When packing medium items, ignore any small items in the knapsack. If this causes the combined size of the items in the knapsack to exceed its capacity, remove small items one by one in any order until all items fit, then stop.

We now show that this allows us to ignore small and large items in our proofs that these algorithms attain the required competitive ratio:

Lemma 1. *Let* ALG *be any algorithm that differentiates between small, medium and large items with respect to b. Assume that* ALG *follows Rules 1-3 regarding small and large items as a first priority. Then,* ALG *achieves a competitive ratio of at most* $1/b$ *if and only if it achieves such a ratio on all instances containing no small or large items.*

Proof. Assume that ALG obtains a competitive ratio of at most $1/b$ on all instances containing only medium items. If an instance contains any large item, ALG will pack it and achieve a gain of at least b and thus a competitive ratio of at most

$1/b$. If a small item in an instance does not fit, ALG must have already packed items of combined size $1-(1-b)=b$ and thus achieve a gain of at least b for a competitive ratio of at most $1/b$. The same holds if a small item has to be removed to make room for a medium item. The only case that remains is that of an instance containing small and medium items, for which ALG packs all small items. Consider any fixed optimal solution to this instance. Let S be the combined size of all small items in the instance and let M_{ALG} be the combined size of the medium items in the algorithm's solution. Lastly, let M_{OPT} be the size of an optimal solution for the subinstance consisting only of medium items. If there are no medium items in the instance, since ALG packs all small items, it is optimal. If there is at least one medium item in the instance, ALG achieves a gain of $M_{\text{ALG}}+S$, while the optimal solution cannot be larger than $M_{\text{OPT}}+S$. The algorithm therefore achieves a competitive ratio of at most

$$\frac{M_{\text{OPT}}+S}{M_{\text{ALG}}+S} \leq \frac{M_{\text{OPT}}}{M_{\text{ALG}}}.$$

However, since ALG ignores small items when packing medium items by Rule 3, $M_{\text{OPT}}/M_{\text{ALG}}$ is exactly the competitive ratio it would achieve if the instance consisted of the medium items only, so by assumption, $M_{\text{OPT}}/M_{\text{ALG}} \leq 1/b$. □

1.3. Related work

Online problems with recourse date back to Imase and Waxman [8] 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 [9–11]. Recently, recourse was considered for further problems, in particular online matching [12–14].

Many different kinds of semi-online problems have been considered in the literature; Boyar et al. [3] 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. [19]; see the survey by Dwibedy and Mohanty [4] 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. [20] in 2008 and shortly afterwards revised by Emek et al. [21], Böckenhauer et al. [22], and Hromkovič et al. [23]. Since then, it has been applied to many different online problems; for a survey, see the work by Boyar et al. [3] and the textbook by Komm [24].

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 [6] 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. [25]. *Delayed decisions* were also studied for other online problems by Rossmanith [5] and by Chen et al. [26]. Böckenhauer et al. [7] 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. [27] in the classical model and later in the model with removal [28].

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 [6] 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 $1 \leq k \leq 3$, we present algorithms that exactly match this lower bound. For all other values of k we give an upper bound of $1+1/(\sqrt{2k}-1)$.

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.

Table 3
Family of instances in the proof of Theorem 1.

	k 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	$x_4 = \frac{k+1}{k+2} + \varepsilon$	$x_5 = \frac{1}{k+2}$	$y_5 = \frac{k+1}{k+2}$
I_6	x_1	x_2	x_3	$x_4 = \frac{k+1}{k+2} + \varepsilon$	$x_5 = \frac{1}{k+2}$	

All instances start with k items of size x_1, where $x_1 = x_3 = \frac{1}{k+2} + \varepsilon$ and $x_2 = \frac{k+1}{k+2} + (k+2)\varepsilon$.

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
$$\rho_{\text{ALG}} \geq \frac{(k+1)/(k+2) + (k+2)\varepsilon}{k \cdot (1/(k+2) + \varepsilon)} \to \frac{k+1}{k} > \frac{k+2}{k+1} \text{ as } \varepsilon \to 0\,.$$
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 and I_3, and hence also in I_4 and 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
   $$\rho_{\text{ALG}} \geq \frac{\frac{k+1}{k+2} + (k+2)\varepsilon}{(m+1) \cdot (\frac{1}{k+2} + \varepsilon)} \geq \frac{\frac{k+1}{k+2} + (k+2)\varepsilon}{k \cdot (\frac{1}{k+2} + \varepsilon)} \to \frac{k+1}{k} \text{ as } \varepsilon \to 0\,.$$
   - 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
$$\rho_{\text{ALG}} \geq \frac{1}{\frac{k+1}{k+2} + (k+1)\varepsilon} \to \frac{k+2}{k+1} \text{ as } \varepsilon \to 0\,.$$
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, the competitive ratio ρ_{ALG} is at least $(k+2)/(k+1) = 1 + 1/(k+1)$. □

2.2. Upper bounds

We now show that for $1 \leq k \leq 3$, the lower bound of Theorem 1 can be matched by a corresponding upper bound. We start by presenting an algorithm A_1 that uses recourse at most once and that achieves a competitive ratio of 3/2. We distinguish between small, medium, and large items with respect to $b = 2/3$ as described in Subsection 1.2. Algorithm A_1

1. treats large and small items as described in Subsection 1.2.
2. 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 stops.

Theorem 2. *Algorithm A_1 has a competitive ratio of at most* $3/2$.

Proof. We prove that A_1 is either optimal or achieves a gain of at least $2/3$ and thus a competitive ratio of at most $3/2$. As shown in Lemma 1, we can assume that the instance contains only medium items. 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, A_1 can pack two items as well, leading to a gain of at least $1/3 + 1/3 = 2/3$. If the optimal solution contains only one item, the algorithm is optimal, since it keeps the largest item. □

We now present an algorithm A_2 that uses recourse at most twice and that achieves a competitive ratio of $4/3$, matching the lower bound of Theorem 1. We refer to small, medium and large items with respect to $b = 3/4$ as described in Subsection 1.2. The algorithm acts as follows:

1. It treats large and small items as described in Subsection 1.2.
2. It keeps the largest medium item encountered so far. If possible, it also keeps the smallest medium item.
3. If a new medium item x_i fits with a previously encountered medium item x_j such that $x_i + x_j \geq 3/4$, it retrieves x_j if necessary and stops.
4. If a new medium item fits with two previously encountered medium items, it retrieves one or both of those if necessary and stops.

Theorem 3. *Algorithm A_2 uses recourse at most twice and has a competitive ratio of at most* $4/3$.

Proof. We first note that algorithm A_2 never has to use recourse to retrieve the smallest medium item in Rule 2: Let $x'_{\min}$ and $x'_{\max}$ be the smallest and largest medium item *before* the current item is revealed, and let $x_{\min}$ and $x_{\max}$ be the smallest and largest medium item *after* it is revealed. If $x'_{\min} + x'_{\max} \leq 1$, then the algorithm already packed the smallest medium item and can replace it if $x_{\min} < x'_{\min}$ without using recourse. If $x'_{\min} + x'_{\max} > 1$, then $x_{\min} + x_{\max} \leq 1$ is only possible if $x_{\min} < x'_{\min}$, that is, if the current item is $x_{\min}$, $x_{\max} \geq x'_{\max}$, so $x_{\min}$ can also be packed without using recourse. There is one exception in the special case that the second medium item in the instance is larger than the first one; in this case, $x_{\min} = x'_{\max}$ and is thus contained in the knapsack. Therefore, A_2 can only use recourse in Rule 3 or Rule 4. In both cases, it uses recourse once or twice and stops afterwards.

As shown in Lemma 1, we can assume that the instance contains only medium items. We now look at an optimal solution for the instance. It is clear that, if the algorithm stops, it achieves a gain of at least $3/4$ and thus a competitive ratio of at most $4/3$. We will therefore also assume that it does not stop.

If the optimal solution contains one (medium) item, the algorithm is optimal, since it packs the largest item. If the optimal solution contains three items, the algorithm stops due to Rule 4.

The only remaining case is that the optimal solution contains exactly two items. Call these items m_1 and m_2 with $m_1 \leq m_2$. We can assume w.l.o.g. that $M = m_1 + m_2 < 3/4$, otherwise the algorithm would stop due to Rule 3 once the second of these items is revealed.

Case 1: If the algorithm only packs the largest item $x_{\max}$ in the instance, we first note that, since $m_1 \leq m_2$, we have $m_1 \leq M/2 < 3/8$. We must then have $x_{\max} \geq 5/8$ (otherwise, the item m_1 would be packed due to Rule 2). The algorithm therefore has a competitive ratio of at most

$$\rho \leq \frac{m_1 + m_2}{x_{\max}} \leq \frac{3/4}{5/8} = \frac{6}{5} < \frac{4}{3}.$$

Case 2: If the algorithm packs both the largest item $x_{\max}$ and the smallest item $x_{\min}$, we note that $x_{\max} \geq m_2 \geq M/2$. The algorithm therefore has a competitive ratio of at most

$$\rho \leq \frac{m_1 + m_2}{x_{\min} + x_{\max}} \leq \frac{M}{1/4 + M/2}.$$

This term increases with M, so we have

$$\rho \leq \frac{M}{1/4 + M/2} < \frac{3/4}{1/4 + 3/8} = \frac{6}{5} < \frac{4}{3}. \quad \square$$

For the last of our tight bounds, we present an algorithm A_3 that uses recourse at most three times and that achieves a competitive ratio of $5/4$, matching the lower bound of Theorem 1. We refer to small, medium and large items with respect to $b = 4/5$ as described in Subsection 1.2. The algorithm acts as follows:

1. It treats large and small items as described in Subsection 1.2.
2. As a first priority, it always keeps the largest medium item encountered so far. If possible, it also keeps the smallest. If possible after that, it also keeps the second smallest. Note that it will never have to use recourse to maintain this configuration.

3. If either of the following two conditions are met, the algorithm no longer follows the requirements of Rule 2.
   a) If there is a combination of exactly two medium items with combined size at least 10/13, one of which is already in the knapsack, the algorithm packs both without using recourse, and removes any other medium item currently in the knapsack.
   b) If there is a combination of exactly two medium items with combined size at least 10/13 such that both items have size at least 9/25, the algorithm packs both, expending at most one use of recourse, and removes any other medium item currently in the knapsack.

   After this, it no longer follows the requirements of Rules 2 or 3 and only changes the contents of the knapsack due to Rules 4 and 5.
4. If at any point there is a combination of one to three previously encountered medium items with total size at least 4/5, it replaces the items in the knapsack with these, expending at most two uses of recourse (since one of these items must be the current one), removes any other medium item currently in the knapsack, and stops.
5. If any four medium items fit and the algorithm can retrieve all of those that are in the buffer without using recourse more than three times in total, it does so, removes any other medium item currently in the knapsack, and stops.

Theorem 4. *Algorithm A_3 uses recourse at most three times and has a competitive ratio of at most* 5/4.

Proof. The algorithm can use recourse due to Rules 3, 4 or 5. It uses recourse at most once in Rule 3 and at most twice in Rule 4. Since it can only invoke each of these rules once, it uses recourse at most three times in those two steps combined. Rule 5 by definition never exceeds three uses of recourse.

As shown in Lemma 1, we can assume that the instance contains only medium items. We first state the following simple fact:

Observation 1. *As long as the algorithm follows Rule 2, for any item of size x in the buffer, the knapsack will contain items of total size at least $1 - x$.*

Proof. It is clear that this holds in the step where x was moved to the buffer. In fact, the knapsack must contain items of total size *greater* than $1 - x$, since the algorithm would have kept x otherwise. We now show that this continues to hold after each further step. If the contents of the knapsack are not changed, the claim clearly continues to hold. If an additional item is added to the knapsack without removing others or if only the largest item in the knapsack is replaced, then it continues to hold, since the combined size of items in the knapsack only grows larger. If one of the two smallest items y is removed, then the knapsack must now contain items of combined size at least $1 - y \geq 1 - x$, since by definition $y \leq x$ (note that the second smallest medium item would be removed before the smallest). □

We now consider a fixed optimal solution and distinguish two cases.

Case 1: We first assume that this optimal solution has size less than 4/5. If Rule 3 was ever invoked, the algorithm reaches a competitive ratio of at most

$$\frac{4/5}{10/13} = \frac{26}{25} < \frac{5}{4},$$

so we will further assume that this is not the case and the algorithm always follows Rule 2.

Case 1a: The optimal solution contains exactly one item. In this case, the algorithm is optimal, since it keeps the largest item.

Case 1b: The optimal solution contains exactly two items $m_1 \leq m_2$. In this case, we know that $m_1 + m_2 < 10/13$ or $m_1 < 9/25$, since otherwise the conditions of Rule 3 would be satisfied, which we assumed is not the case. If both m_1 and m_2 are part of the algorithm's solution, the algorithm is optimal, so assume that one of them is not. If m_2 is not part of the algorithm's solution, m_1 can also not be part of it: Since that solution also contains the largest item in the instance, it would otherwise beat the optimal solution. By Observation 1, the algorithm achieves a gain of at least $1 - m_1$. Now recall that $m_1 + m_2 < 10/13$ or $m_1 < 9/25$. If $m_1 + m_2 < 10/13$, then $m_1 < 5/13$, so the algorithm achieves a gain of at least 8/13 and a competitive ratio of at most

$$\frac{10/13}{8/13} = \frac{5}{4}.$$

If $m_1 < 9/25$, the algorithm achieves a competitive ratio of at most

$$\frac{4/5}{16/25} = \frac{5}{4}.$$

Case 1c: The optimal solution contains exactly three items $m_1 \leq m_2 \leq m_3$. We then know that $m_2 + m_3 < 4/5 - 1/5 = 3/5$, and so $m_2 < 3/10$. If the algorithm's solution contains all three of these items, it is optimal, so assume that this is not the

case. If one of the two smaller items m_1 or m_2 are not in the knapsack (or if $m_2 = m_3$ and m_3 is not in the knapsack), by Observation 1, the algorithm achieves a gain of at least $1 - m_2 > 7/10$ and a competitive ratio of at least

$$\frac{4/5}{7/10} = \frac{8}{7} < \frac{5}{4}.$$

We therefore know that the algorithm packs m_1 and m_2, but not m_3, and that $m_2 < m_3$. This however means that the algorithm is optimal, since it must also pack the largest medium item, which is at least m_3.

Case 2: We now cover the case where the optimal solution is at least 4/5.

Case 2a: The optimal solution contains fewer than four items. In this case, the algorithm can achieve a gain of at least 4/5 due to Rule 4.

Case 2b: The optimal solution contains exactly four items and Rule 3b) was never invoked. In this case, when the fourth of these items arrives, the algorithm has three uses of recourse left, so it achieves a gain of at least 4/5 due to Rule 5.

Case 2c: The optimal solution contains exactly four items and Rule 3b) was invoked. In this case, the algorithm has two more uses of recourse left. We will see that this is enough. Let $m_1 \leq m_2 \leq m_3 \leq m_4$ be the items in the optimal solution and let $x_1 \leq x_2$ be the items packed by Rule 3b). We can assume that these items are distinct, otherwise the algorithm could easily pack four items. By definition, $x_1 + x_2 \geq 10/13$ and $x_1 \geq 9/25$. We can also assume that $x_1 + x_2 < 4/5$, so $x_1 < 2/5$, and that $m_1 + m_2 + m_3 + m_4 > 5/4 \cdot (x_1 + x_2) \geq 25/26$, so $m_3 + m_4 > 25/52$. Since $m_1 + m_2 + m_3 + m_4 \leq 1$, we have $m_3 + m_4 < 3/5$. This means that $x_1 + m_3 + m_4 < 1$. On the other hand, $x_1 + m_3 + m_4 \geq 9/25 + 25/52 = 1093/1300 > 4/5$, so the algorithm is able to achieve a gain of at least 4/5 due to Rule 4. □

Finally, we present a rough upper bound for general values of k. For this, we define a family of c-competitive algorithms A'_c for any $1 < c < 2$ and bound the amount of recourse they use. Let $1 < c < 2$. We refer to small, medium and large items with respect to $1/c$ as described in Subsection 1.2. Then the algorithm A'_c acts as follows:

1. It treats large and small items as described in Subsection 1.2.
2. It packs medium items in two phases:
   Phase 1. The algorithm packs any medium item it can. The first time a medium item no longer fits, it removes medium items such that it retains an optimal solution on the subinstance consisting only of medium items, and enters Phase 2.
   Phase 2. The algorithm keeps track of the size M_{OPT} of an optimal solution on the subinstance consisting only of medium items, and the combined size M_{ALG} of the medium items in its current solution. If $\lfloor \log_c(2M_{\text{OPT}}) \rfloor > \lfloor \log_c(2M_{\text{ALG}}) \rfloor$, it removes all medium items currently in the knapsack and uses recourse to pack medium items of combined size M_{OPT}.

Lemma 2. *For any $1 < c < 2$, algorithm A'_c is c-competitive and uses recourse no more than*

$$\frac{c^2 \cdot \left(c^{\lfloor \log_c(2) \rfloor} - 1\right)}{2 \cdot (c-1)^2}$$

times.

Proof. We first show that the algorithm achieves a competitive ratio of c. As shown in Lemma 1, we can assume that the instance contains only medium items. If the algorithm never enters Phase 2, it is clearly optimal. Note that by definition of Phase 1, the first time a medium item does not fit, all previous medium items must be contained in the knapsack. Thus, A'_c can in fact reconstruct an optimal solution by removal only. Since the instance contains only medium items, M_{OPT} and M_{ALG} as defined in Phase 2 are in fact the current optimal solution, and the algorithm's solution, respectively. Finally, in Phase 2, as long as $\lfloor \log_c(2M_{\text{OPT}}) \rfloor = \lfloor \log_c(2M_{\text{ALG}}) \rfloor$, we know that $\log_c(2M_{\text{OPT}}) - \log_c(2M_{\text{ALG}}) \leq 1$, so $\log_c(M_{\text{OPT}}/M_{\text{ALG}}) \leq 1$, and $M_{\text{OPT}}/M_{\text{ALG}} \leq c$. Thus, A'_c is in fact c-competitive.

We now argue on the number of uses of recourse that A'_c expends. At the end of Phase 1, and thus at the beginning of Phase 2, $M_{\text{ALG}} > 1/2$: If a medium item does not fit, then the combined size of medium items previously in the knapsack or the new medium item must be greater than 1/2. This means that $\log_c(2M_{\text{ALG}})$ is always positive.

Now assume that a medium item arrives which causes $\lfloor \log_c(2M_{\text{OPT}}) \rfloor > \lfloor \log_c(2M_{\text{ALG}}) \rfloor$.

Since M_{OPT} increased, the current medium item must be part of any optimal solution at this point and A'_c can pack it without using recourse. Let M'_{OPT} be the combined size of all other medium items in such an optimal solution. Since all those items were available before, we know that $\lfloor \log_c(2M'_{\text{OPT}}) \rfloor \leq \lfloor \log_c(2M_{\text{ALG}}) \rfloor$, and thus $M'_{\text{OPT}} \leq c^{\lfloor \log_c(2M_{\text{ALG}}) \rfloor + 1} \cdot 1/2$.

Since any medium item has size greater than $1 - 1/c$, the algorithm has to expend at most $c^{\lfloor \log_c(2M_{\text{ALG}}) \rfloor + 1}/(2 \cdot (1 - 1/c))$ uses of recourse to reconstruct an optimal solution.

Each time A'_c uses recourse, $\lfloor \log_c(2M_{\text{ALG}}) \rfloor$ increases by at least 1. The largest possible value that $\lfloor \log_c(2M_{\text{OPT}}) \rfloor$ can achieve is $\lfloor \log_c(2) \rfloor$, so whenever the algorithm uses recourse, $\lfloor \log_c(2M_{\text{ALG}}) \rfloor < \lfloor \log_c(2) \rfloor$, so $\lfloor \log_c(2M_{\text{ALG}}) \rfloor \leq \lfloor \log_c(2) \rfloor - 1$. Therefore, iterating through the possible values of $\lfloor \log_c(2M_{\text{ALG}}) \rfloor$, the number of uses of recourse expended by A'_c is limited by

$$\begin{aligned}
&\sum_{i=0}^{\lfloor \log_c(2) \rfloor - 1} \frac{c^{i+1}}{2 \cdot (1 - 1/c)} \\
&= \frac{c^2}{2 \cdot (c-1)} \cdot \sum_{i=0}^{\lfloor \log_c(2) \rfloor - 1} c^i \\
&= \frac{c^2}{2 \cdot (c-1)} \cdot \frac{c^{\lfloor \log_c(2) \rfloor} - 1}{c-1} \\
&= \frac{c^2 \cdot \left(c^{\lfloor \log_c(2) \rfloor} - 1\right)}{2 \cdot (c-1)^2} . \quad \square
\end{aligned}$$

Theorem 5. *For any $k \geq 4$, there exists an algorithm A_k that uses recourse at most k times in total and has a competitive ratio of at most*

$$1 + \frac{1}{\sqrt{2k} - 1} .$$

Proof. For smaller values of k, we can use the algorithm from Theorem 4 that uses recourse at most three times. For larger values, we use the algorithm A'_c for as small a value of c as possible. By Lemma 2, A'_c uses recourse at most

$$\frac{c^2 \cdot \left(c^{\lfloor \log_c(2) \rfloor} - 1\right)}{2 \cdot (c-1)^2} \leq \frac{c^2 \cdot \left(c^{\log_c(2)} - 1\right)}{2 \cdot (c-1)^2} = \frac{c^2}{2 \cdot (c-1)^2}$$

times. It therefore suffices to choose c such that

$$\frac{c^2}{2 \cdot (c-1)^2} \leq k .$$

This is a quadratic inequality and is equivalent to

$$c \geq \frac{2k + \sqrt{2k}}{2k - 1} = 1 + \frac{\sqrt{2k} + 1}{2k - 1} = 1 + \frac{1}{\sqrt{2k} - 1} . \quad \square$$

3. Number of uses of recourse bounded per step

In this section, we consider algorithms that can use recourse a limited number of $k \geq 1$ 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 $b_k = (\sqrt{k^2 + 6k + 5} - k - 1)/2$. Then b_k satisfies the equation

$$b_k^2 + (k+1) \cdot b_k - (k+1) = 0 . \tag{1}$$

It is easy to check that $1/b_k = (\sqrt{k^2 + 6k + 5} + k + 1)/(2k + 2)$. Alternatively, we can define b_k as the unique positive root of the quadratic function $\varphi(x) = x^2 + (k+1) \cdot x - (k+1)$. Since $\varphi((k+1)/(k+2)) = -(k+1)/(k+2)^2 < 0$ and $\varphi((k+2)/(k+3)) = 1/(k+3)^2 > 0$, we can check that $(k+1)/(k+2) < b_k < (k+2)/(k+3)$, and thus $1 + 1/(k+2) < 1/b_k < 1 + 1/(k+1)$.

3.1. Upper bounds

We first give a family of algorithms B_k that use recourse at most k times per step and achieve a competitive ratio of $1/b_k$. We distinguish between small, medium and large items with respect to b_k as described in Subsection 1.2.

1. Algorithm B_k treats small and large items as described in Subsection 1.2.
2. As long as at most k medium items fit, B_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. The algorithm then abandons Rule 2 and continues with Rule 3.
3. The algorithm maintains the $k+1$ smallest medium items encountered so far. Note that it can do so without using recourse.
   (a) If it encounters an additional medium item that fits with these $k+1$, it packs it and stops.
   (b) 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 stops.

Theorem 6. *For any $k \geq 1$, algorithm B_k has a competitive ratio of at most $1/b_k < 1 + \frac{1}{k+1}$.*

Proof. As shown in Lemma 1, we can assume that the instance contains only medium items.

If there are $k+2$ items in the output of B_k, then the algorithm's gain is at least $(k+2) \cdot (1-b_k)$, which is greater than b_k, since $b_k < (k+2)/(k+3)$, so its competitive ratio is smaller than $1/b_k$.

If there are at most k items in the output of B_k, either at most k items can be packed together, in which case the algorithm is optimal, or they were packed because their sum is at least b_k, in which case the algorithm achieves a gain of at least b_k and a competitive ratio of at most $1/b_k$.

We can therefore assume there are exactly $k+1$ items in the output of B_k. If their sum is at least b_k, then B_k has a competitive ratio of at least $1/b_k$. Otherwise, let M_{ALG} be their sum and let M_{OPT} be the value of an optimal solution. There are at most $k+1$ items in the optimal solution, otherwise the algorithm would have packed $k+2$ items by Rule 3*a*). Also, $M_{\text{OPT}} < b_k$, otherwise B_k would have packed items of combined size at least b_k by Rule 3*b*). Now, the competitive ratio of B_k on the instance is at most

$$\frac{M_{\text{OPT}}}{M_{\text{ALG}}} \leq \frac{b_k}{(k+1) \cdot (1-b_k)} = \frac{1}{b_k}$$

by choice of b_k. □

An interesting thing to note is that the definition of B_k and the proof of Theorem 6 also hold for $k = 0$. In this case, $1/b_k = (\sqrt{5}+1)/2$, which is the competitive ratio of the algorithm for online knapsack with removal (but without recourse) in [6]. The algorithms B_k can thus be seen as an extension of that algorithm.

3.2. Lower bounds

We now prove that the algorithms B_k are in fact the best possible.

Theorem 7. *Any algorithm* ALG *that uses recourse at most k times per step cannot have a competitive ratio of less than $1/b_k > 1 + \frac{1}{k+2}$.*

Proof. We define $a_k = 1 - b_k$ and present two instances $I_1 = (a_k, \ldots, a_k, b_k + \varepsilon)$ and $I_2 = (a_k, \ldots, a_k, b_k + \varepsilon, 1 - (k+1) \cdot a_k)$, both of which start with $k+1$ copies of a_k. Since I_1 is a prefix of I_2, the algorithm must act the same on both instances up to the item $b_k + \varepsilon$. We claim that the algorithm cannot have a competitive ratio of less than $1/b_k$ on both of these instances.

By design, the algorithm will not be able to pack both an item of size a_k and the item of size $b_k + \varepsilon$. Since $b_k > (k+1)/(k+2)$, it also cannot pack the two items $b_k + \varepsilon$ and $1 - (k+1) \cdot a_k$ simultaneously.

If no items are packed, the competitive ratio is unbounded on I_1. If the algorithm packs up to $k+1$ items a_k, its competitive ratio on I_1 is at least

$$\rho_{\text{ALG}} \geq \frac{b_k + \varepsilon}{(k+1) \cdot a_k} \geq \frac{b_k}{(k+1) \cdot (1-b_k)} = \frac{1}{b_k},$$

by definition of b_k in (1).

If the item $b_k + \varepsilon$ is packed, then the algorithm cannot retrieve all $k+1$ items a_k when the item $1 - (k+1) \cdot a_k$ is presented in I_2. Since $1 - (k+1) \cdot a_k + k \cdot a_k = 1 - (1 - b_k) = b_k$, the competitive ratio on this instance is at least

$$\rho_{\text{ALG}} \geq \frac{1}{b_k + \varepsilon},$$

which goes to $1/b_k$ as $\varepsilon \to 0$. □

4. Information on the end of an 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, there might be an advantage in recognizing the end of an instance.

4.1. Two different ways to communicate the end of an instance

There appear to be two different ways in which the information that the instance ends might be encoded. On the 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 and can use removal and recourse as usual. We will show in Lemma 3 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, as will be shown in Lemma 4.

Lemma 3. *For any algorithm* ALG *operating on instances of the form*

$$(x_1, \ldots, x_{n-1}, (x_n, \perp))$$

that has a competitive ratio of $c \geq 1$, there is an algorithm ALG′ *with the same limits on its recourse as* ALG *operating on instances of the form*

$$(x_1, \ldots, x_n, \perp)$$

with a competitive ratio of at most c.

Proof. Algorithm ALG′ copies the behavior of ALG up to the last item. Given the item $\perp$, it behaves as ALG would on the item $(0, \perp)$. This change does not affect the optimal solution. Since ALG has a competitive ratio of at most c on that instance by assumption, ALG′ has a competitive ratio of at most c on the original instance and thus on any instance. □

Lemma 4. *For any algorithm* ALG *operating on instances of the form*

$$(x_1, \ldots, x_n, \perp)$$

that uses recourse at most k times in total and that has a competitive ratio of $c \geq 1$, there is an algorithm ALG′ *operating on instances of the form*

$$(x_1, \ldots, x_{n-1}, (x_n, \perp))$$

that uses recourse the same number of times as ALG *with a competitive ratio of at most c.*

Proof. Given an instance $(x_1, \ldots, x_{n-1}, (x_n, \perp))$, ALG′ simply simulates ALG on the instance $(x_1, \ldots, x_{n-1}, x_n, \perp)$, leading to the same competitive ratio that ALG achieves. □

Note that the same proof does not work (without adaptation) in the scenario where algorithms may use their recourse k times per step, since ALG would be able to use its recourse an additional k times compared to ALG′.

In this section, we will focus on instances of the form $(x_1, \ldots, x_{n-1}, (x_n, \perp))$ even in the case where the amount of recourse is limited by step, since it feels natural to tie the use of recourse to the reveal of an item.

4.2. Lower bounds

We first give a lower bound on the competitive ratio of any algorithm for the knapsack problem with removal and limited recourse. The bound is given for $k \geq 1$ uses of recourse per step, and thus clearly also holds for $k \geq 1$ uses of recourse in total.

Theorem 8. *No algorithm that receives instances of the form $(x_1, \ldots, x_{n-1}, (x_n, \perp))$, and that uses recourse at most $k \geq 1$ times per step can have a competitive ratio better than*

$$c_k = \frac{2k+1}{\sqrt{4k^4 + 8k^3 + 8k^2 + 2k} - 2k^2},$$

where the asymptotic behavior of c_k is given by

$$1 + \frac{1}{4k^2 + 4k + 3} \leq c_k \leq 1 + \frac{1}{4k^2 + 2k - 1}.$$

Proof. We choose $a_k > 0$ to satisfy the following condition:

$$\frac{1}{(2k+1) \cdot a_k} = \frac{(2k+1) \cdot a_k}{2k \cdot (1 - 2k \cdot a_k)}. \tag{2}$$

Solving this equation leads to

$$a_k = \frac{\sqrt{4k^4 + 8k^3 + 8k^2 + 2k} - 2k^2}{(2k+1)^2}\,, \tag{3}$$

so

$$a_k < \frac{\sqrt{4k^4 + 8k^3 + 8k^2 + 4k + 1} - 2k^2}{(2k+1)^2} = \frac{1}{2k+1}\,. \tag{4}$$

We then set $b_k = 1 - 2k \cdot a_k + \varepsilon$ where $\varepsilon > 0$ is chosen sufficiently small, and, since $(2k+1) \cdot a_k < 1$ by (4), we have $b_k > a_k$. We present $2k+1$ copies of a_k and $2k$ copies of b_k. An important thing to note is that any combination of $2k$ of these items will fit, but the only way to pack $2k+1$ of them is to pack all copies of a_k.

If the algorithm packs fewer than k copies of b_k, we present the final item $(1 - 2k \cdot b_k, \perp)$. Since it can retrieve at most k copies of b_k, the algorithm's best possible gain is $a_k + (2k-1) \cdot b_k + (1 - 2k \cdot b_k) = 1 + a_k - b_k = (2k+1) \cdot a_k - \varepsilon$ if it decides to pack at most $2k$ of the original items. If the algorithm instead packs all $(2k+1)$ copies of a_k, it can no longer pack the final item, since by Equations (2) and (4), $2k \cdot b_k < (2k+1) \cdot a_k$ for small enough values of ε. The algorithm's competitive ratio is therefore at least $1/((2k+1) \cdot a_k)$.

If the algorithm packs at least k copies of b_k, it must have packed at most k copies of a_k. We then present the final item $(\varepsilon, \perp)$. Since $2k \cdot a_k + b_k > 1$ and $b_k > a_k$, and at most k items a_k can be retrieved, the algorithm can pack at most $2k$ of the original items for a gain of at most $2k \cdot b_k + \varepsilon$. As $\varepsilon \to 0$, its competitive ratio is therefore at least $(2k+1) \cdot a_k / ((2k \cdot (1 - 2k \cdot a_k))$, which is $1/((2k+1) \cdot a_k)$ by Equation (2).

Using the value of a_k in (3), the competitive ratio the algorithm achieves for a fixed k is at least

$$\frac{1}{(2k+1) \cdot a_k} = \frac{2k+1}{\sqrt{4k^4 + 8k^3 + 8k^2 + 2k} - 2k^2}\,.$$

For the bounds on c_k, note that

$$\left(2k^2 + 2k + 1 - \frac{1}{2k} + \frac{1}{4k^2}\right)^2 \geq 4k^4 + 8k^3 + 8k^2 + 2k\,,$$

hence

$$\begin{aligned}
&\frac{2k+1}{\sqrt{4k^4 + 8k^3 + 8k^2 + 2k} - 2k^2} \geq \frac{2k+1}{\left(2k^2 + 2k + 1 - \frac{1}{2k} + \frac{1}{4k^2}\right) - 2k^2} \\
&= \frac{8k^3 + 4k^2}{8k^3 + 4k^2 - 2k + 1} = 1 + \frac{2k-1}{8k^3 + 4k^2 - 2k + 1} \\
&\geq 1 + \frac{2k-1}{8k^3 + 4k^2 + 2k - 3} = 1 + \frac{1}{4k^2 + 4k + 3}\,.
\end{aligned}$$

On the other hand, note that

$$\left(2k^2 + 2k + 1 - \frac{1}{2k}\right)^2 \leq 4k^4 + 8k^3 + 8k^2 + 2k\,,$$

hence

$$\begin{aligned}
&\frac{2k+1}{\sqrt{4k^4 + 8k^3 + 8k^2 + 2k} - 2k^2} \leq \frac{2k+1}{\left(2k^2 + 2k + 1 - \frac{1}{2k}\right) - 2k^2} \\
&= \frac{2k+1}{2k + 1 - \frac{1}{2k}} = \frac{4k^2 + 2k}{4k^2 + 2k - 1} = 1 + \frac{1}{4k^2 + 2k - 1}\,. \quad \square
\end{aligned}$$

4.3. Upper bounds

Lastly, we give concrete algorithms that use the fact that the end of the instance is communicated to them. We again assume that the instance is given in the form $(x_1, \ldots, x_{n-1}, (x_n, \perp))$.

We first describe a family of algorithms $A_k^{\perp}(b)$ that use recourse at most k times in total. Let $1/2 < b < 1$. We distinguish between small, medium, and large items with respect to b as described in Subsection 1.2 and present an algorithm that expends at most $k \geq 1$ uses of recourse in total. The algorithm acts as follows:

1. It treats small and large items as described in Subsection 1.2.
2. It keeps as many of the smallest medium items as possible, prioritizing smaller items.
3. Once the final item $(x_n, \perp)$ arrives, it computes an arbitrary optimal offline solution (note that the algorithm is not required to be efficient).

	Optimal solution		
number of items	r	$i = j - r$	k
items	$m_1, \ldots, m_r$	$m_{r+1}, \ldots, m_j$	$m_{j+1}, \ldots, m_{k+j}$
size	M_1	M_2	M_3

	Solution constructed by algorithm		
number of items	r	$i = j - r$	k
items	$m_1, \ldots, m_r$	$z_1, \ldots, z_i$	$m_{j+1}, \ldots, m_{k+j}$
size	M_1	$\geq (k+i) \cdot (1-b) \cdot M_2$	M_3

Fig. 2. Solution constructed by the algorithm in Theorem 9.

- If the optimal solution contains at most k medium items, it retrieves them.
- If the optimal solution contains exactly $k+i$ medium items for $i > 0$, it retrieves the largest k of these. It further packs all of the medium items in the optimal solution that are among the smallest ones kept in the knapsack. Finally, it packs the remaining of the smallest medium items one by one, starting with the largest, as long as they fit.

Lemma 5. *For any $k \geq 1$, algorithm $A_k^{\perp}(b)$ uses recourse at most k times in total and achieves a competitive ratio of at most $1/b$ if*

$$b \leq \frac{(k+i) \cdot i + k}{(i+1) \cdot (k+i)} \tag{5}$$

for all $i \in \mathbb{N}$ with $i \geq 1$.

Proof. As shown in Lemma 1, we can assume that the instance contains only medium items. If the optimal solution contains at most k items, the algorithm is optimal. So assume the optimal solution contains $k+j$ items for $j \geq 1$. We now fix some notation. We call the items in the optimal solution $m_1 \leq \cdots \leq m_{k+j}$. We first consider any items that are already in the knapsack when the final item arrives. Since the items are ordered by size, we can assume that those are the items $m_1, \ldots, m_r$ for $0 \leq r \leq k+j$. If $r \geq j$, the algorithm is optimal, since it can retrieve the k items $m_{j+1}, \ldots, m_{k+j}$. We will therefore assume that $r < j$ and define $i = j - r$. Since the optimal solution contains $k+j$ items, there must be at least $k+j-1$ items already in the knapsack, of which at least $k+j-1-r \geq i$ are not contained in the optimal solution. We choose the largest i such items and call these $z_1 \leq \cdots \leq z_i$. We introduce the notation $M_1 = m_1 + \cdots + m_r$, $M_2 = m_{r+1} + \cdots + m_j$ and $M_3 = m_{j+1} + \cdots + m_{k+j}$, such that the optimal solution has size $M_1 + M_2 + M_3$. The process is illustrated in Fig. 2. Since the items are ordered, we know that $M_2/i \leq M_3/k$ and thus $M_3 \geq (k/i) \cdot M_2$.

We first prove that $M_2 \leq i/(k+i)$. If not, we would have

$$M_2 + M_3 \geq M_2 \cdot (1 + k/i) > \frac{i}{k+i} \cdot \frac{k+i}{i} = 1\,.$$

This implies that

$$z_1 + \cdots + z_i \geq i \cdot (1-b) = (1-b) \cdot (k+i) \cdot \frac{i}{k+i} \geq (k+i) \cdot (1-b) \cdot M_2\,.$$

The algorithm will pack the k items $m_{j+1}, \ldots, m_{k+j}$, the r items $m_1, \ldots, m_r$ and at least the i items $z_1, \ldots, z_i$, for a competitive ratio of at most

$$\begin{aligned}
\frac{M_1 + M_2 + M_3}{M_1 + (k+i) \cdot (1-b) \cdot M_2 + M_3} &\leq \frac{M_2 + M_3}{(k+i) \cdot (1-b) \cdot M_2 + M_3} \\
&\leq \frac{M_2 + k/i \cdot M_2}{(k+i) \cdot (1-b) \cdot M_2 + k/i \cdot M_2} \\
&= \frac{k+i}{i \cdot (k+i) \cdot (1-b) + k} \\
&= \frac{1}{i \cdot (1-b) + k/(k+i)}\,,
\end{aligned}$$

which is at most $1/b$, since

$$\begin{aligned}
& i \cdot (1-b) + k/(k+i) \geq b \\
\iff & (k+i) \cdot i \cdot (1-b) + k \geq (k+i) \cdot b \\
\iff & (k+i) \cdot i + k \geq (k+i) \cdot (i+1) \cdot b \\
\iff & b \leq \frac{(k+i) \cdot i + k}{(i+1) \cdot (k+i)}
\end{aligned}$$

which is given by (5). □

Theorem 9. *For any $k \geq 1$, for the choice of $b = \left(1 + 1/(k + 2\sqrt{k})\right)^{-1}$, algorithm $A_k^{\perp}(b)$ achieves a competitive ratio of at most*

$$1 + \frac{1}{k + 2\sqrt{k}}.$$

Proof. We can see that

$$\frac{(i+1)\cdot(k+i)}{(k+i)\cdot i + k} = 1 + \frac{i}{(k+i)\cdot i + k} = 1 + \frac{1}{k + i + k/i}.$$

It can be checked using standard methods of calculus that the real function $f(x) = k + x + k/x$ takes its minimum on $\{x > 0\}$ at $x = \sqrt{k}$. Hence, by choosing $b = \left(1 + 1/(k + 2\sqrt{k})\right)^{-1}$, Condition (5) is fulfilled and by Lemma 5, $A_k^{\perp}(b)$ achieves a competitive ratio of $1/b$. □

Lastly, we give a family of algorithms $B_k^{\perp}(b)$ that use recourse at most k times per step depending on a paramter b and make use of the fact that they know the end of an instance. We again assume that the instance is given in the form

$$(x_1, \ldots, x_{n-1}, (x_n, \perp)).$$

Let $1/2 < b \leq (2k+2)/(2k+3)$. We distinguish between small, medium and large items with respect to b as described in Subsection 1.2 and present an algorithm that behaves as follows:

1. It treats large and small items as described in Subsection 1.2.
2. It keeps as many of the smallest medium items as possible, prioritizing smaller items.
3. If any $2k+2$ medium items fit, it adds the current one to the $2k+1$ smallest previous ones and stops.
4. If at any point a gain of at least b can be reached with at most k medium items, it retrieves them and stops.
5. If at any point a gain of at least b can be reached with $k < \ell \leq 2k+1$ medium items and the current item is not the last one, it retrieves the other at most $2k$ items in two steps and stops.
6. If the item $(x_n, \perp)$ arrives, it computes an arbitrary optimal offline solution (note that the algorithm is not required to be efficient). If that solution contains at most k medium items, it retrieves them. If it contains $k < \ell \leq 2k+1$ medium items, if x_n is part of the optimal solution, it adds it to the knapsack. Then it retrieves the largest k medium items in the optimal solution that are not x_n. It further packs all of the medium items in the optimal solution that are among the smallest ones kept in the knapsack. Finally, it packs the remaining of the smallest medium items one by one, starting with the largest, as long as they fit.

Lemma 6. *For any $k \geq 1$, algorithm $B_k^{\perp}(b)$ uses recourse at most k times per step and has a competitive ratio of at most $1/b$ as long as b satisfies the condition*

$$b^2 + i \cdot b - \frac{k}{k+i} \cdot b - i \leq 0 \tag{6}$$

for all $1 \leq i \leq k+1$.

Proof. We first show that $b \leq (2k+2)/(2k+3)$: Since Condition (6) is satisfied for $b = 0$ and the coefficient of b^2 is positive, we know that, if the condition is not satisfied for $b = (2k+2)/(2k+3)$, it is also not satisfied for any larger value of b. This is the case, since for $i = 1$,

$$\left(\frac{2k+2}{2k+3}\right)^2 + \frac{2k+2}{2k+3} - \frac{k}{k+1} \cdot \frac{2k+2}{2k+3} - 1 = \frac{1}{(2k+3)^2} > 0.$$

This means that $(2k+2) \geq (2k+3)\cdot b$, so $(2k+2)\cdot(1-b) \geq b$, which in turn implies that, if the algorithm can pack any $2k+2$ medium items, it achieves a gain of at least b and hence a competitive ratio of at most $1/b$. If any $2k+2$ medium items fit, the algorithm can do this: Before the last of these items was revealed, the other $2k+1$ medium items must have fit, so the algorithm would have kept at least the smallest $2k+1$ medium items, which would then fit with the newly revealed one. We will therefore assume that no $2k+2$ medium items fit. (Note that, for certain values of b for $k \geq 3$, it might not even be possible for $2k+2$ medium items to fit.) As shown in Lemma 1, we can also assume that the instance contains only medium items. We now consider any optimal solution. If this solution consists of at most k items, the algorithm can retrieve them all and is optimal. We will therefore assume that it consists of $k+j$ items $m_1 \leq \cdots \leq m_{k+j}$ for $1 \leq j \leq k+1$.

Case 1: We first assume that $m_1 + \cdots + m_{k+j} < b$. We also consider the ℓ smallest items $y_1 \le \cdots \le y_\ell$ that are kept in the knapsack before the final item. Clearly, since $k+j$ items fit after the final item, we must have $\ell \ge k+j-1 \ge j$. The algorithm will retrieve the k items $m_{j+1}, \ldots, m_{k+j}$. It will also be able to use an additional $r \ge 0$ of the items $y_1, \ldots, y_\ell$ that are also in the optimal solution. Since the items are ordered, we will assume that these are the items $m_1, \ldots, m_r$. If $r = j$, the algorithm is optimal, so we will assume that $r < j$ and define $i = j - r$. In this case, since $\ell - r \ge j - r = i$, we must have at least i of the smallest items left. We will choose the i largest of these and call them $z_1 \le \cdots \le z_i$. We also use the notation $M_1 = m_1 + \cdots + m_r$, $M_2 = m_{r+1} + \cdots + m_j$ and $M_3 = m_{j+1} + \cdots + m_{k+j}$ such that the optimal solution has size $M_1 + M_2 + M_3$. Since the items are ordered, we know that $M_2/i \le M_3/k$ and thus $M_3 \ge (k/i) \cdot M_2$. This implies that $M_2 + M_3 \ge M_2 \cdot (1 + k/i)$, so $M_2 \le (M_2 + M_3) \cdot i/(k+i) \le b \cdot i/(k+i)$, which means that

$$z_1 + \cdots + z_i \ge i \cdot (1-b) = (1-b) \cdot i \cdot \frac{k+i}{b} \cdot \frac{b}{k+i} \ge (1-b) \cdot \frac{k+i}{b} \cdot M_2 \,.$$

The algorithm will construct a solution consisting (at least) of the $k+j$ items $m_1, \ldots, m_r, z_1, \ldots, z_i, m_{j+1}, \ldots, m_{k+j}$. This means, its competitive ratio is at most

$$\begin{aligned} \frac{M_1 + M_2 + M_3}{M_1 + \frac{(1-b)\cdot(k+i)}{b} \cdot M_2 + M_3} &\le \frac{M_2 + M_3}{\frac{(1-b)\cdot(k+i)}{b} \cdot M_2 + M_3} \\ &\le \frac{M_2 + (k/i) \cdot M_2}{\frac{(1-b)\cdot(k+i)}{b} \cdot M_2 + (k/i) \cdot M_2} \\ &= \frac{b \cdot (k+i)}{(1-b) \cdot (k+i) \cdot i + k \cdot b} \\ &= \frac{b}{i + b \cdot \frac{k}{k+i} - i \cdot b} \,. \end{aligned} \tag{7}$$

Condition (6) implies that $b^2 + i \cdot b - \frac{k}{k+i} \cdot b - i \le 0$ and from this we can see that the competitive ratio is at most $1/b$.

Case 2: We now deal with the case where the optimal solution has size at least b. If it does not contain the final item x_n, the optimal solution must have already had size at least b when the item x_{n-1} was revealed. In this case, the algorithm would have retrieved all of these items in two steps and is optimal. If x_n is part of the optimal solution, we can pack it. Let $m_1 \le \cdots \le m_{k+j}$ be the *other* items in the optimal solution. If $j = 0$, the algorithm will retrieve all of them and is optimal, so we can assume that $j \ge 1$. Now we define r, i, M_1, M_2 and M_3 as in Case 1. Since $x_n > 1 - b$, we have $M_2 + M_3 < b$. Since the items are ordered, we also still have $M_2/i \le M_3/k$. Using the same argument as in Case 1, we can show that the algorithm reaches a gain of at least $x_n + M_1 + \frac{(1-b)\cdot(k+i)}{b} \cdot M_2 + M_3$ for a competitive ratio of

$$\frac{x_n + M_1 + M_2 + M_3}{x_n + M_1 + \frac{(1-b)\cdot(k+i)}{b} \cdot M_2 + M_3} \le \frac{M_2 + M_3}{\frac{(1-b)\cdot(k+i)}{b} \cdot M_2 + M_3} \le \frac{1}{b} \,,$$

as we have shown in (7). □

Theorem 10. *For any $k \ge 1$, $B_k^{\perp}(b)$ achieves a competitive ratio of*

$$1 + \frac{1}{k + 2\sqrt{k} + \alpha} \,,$$

where

$$\alpha = \frac{2\sqrt{2} + \sqrt{5 + 4\sqrt{2}} - 5}{2} > 0.546$$

and

$$b = \frac{k + 2\sqrt{k} + \alpha}{k + 2\sqrt{k} + \alpha + 1} \,.$$

Proof. We will use Lemma 6. Consider a competitive ratio of $c = 1 + 1/a$. It can be checked that a competitive ratio of c can be achieved, that is, $b = 1/c = a/(a+1)$ satisfies Condition (6), if

$$a^2 \cdot i - (a+1) \cdot i \cdot (k+i) \le a \cdot k \tag{8}$$

for all $1 \leq i \leq k+1$. The left-hand side of (8) is quadratic in i. For real values of i, it takes a maximum of $(a \cdot (k-a) + k)^2/(4 \cdot (a+1))$. It therefore suffices to check that

$$\frac{(a \cdot (k-a) + k)^2}{4 \cdot (a+1)} \leq a \cdot k\,.$$

If we choose $a = k + 2\sqrt{k} + d$, for some $d > 0$, this condition can be transformed into

$$(4-4d) \cdot k^{5/2} + (11 - 14d - d^2) \cdot k^2 + (8 - 8d - 12d^2) \cdot k^{3/2} + (4d - 18d^2 - 2d^3) \cdot k - 8d^3 \cdot k^{1/2} - d^4 \geq 0\,. \tag{9}$$

Consider the real polynomial

$$p_d(x) = (4-4d) \cdot x^5 + (11 - 14d - d^2) \cdot x^4 + (8 - 8d - 12d^2) \cdot x^3 + (4d - 18d^2 - 2d^3) \cdot x^2 - 8d^3 \cdot x - d^4\,.$$

Then, the left-hand side of (9) is exactly $p_d(k^{1/2})$. We can see that

$$p_d(1) = -d^4 - 10 \cdot d^3 - 31 \cdot d^2 - 22 \cdot d + 23\,.$$

The given value for α is exactly the positive root in d of $p_d(1)$, so $p_\alpha(1) = 0$. Furthermore, the first three coefficients of $p_\alpha(x)$ are positive and the last three coefficients are negative, so, by Descartes' rule of signs, $p_\alpha(x)$ will have exactly one positive root. Since the coefficient of x^5 is positive and $p_\alpha(1) \geq 0$, this implies that $p_\alpha(x) \geq 0$ for all $x \geq 1$ and thus (9) is satisfied for all $k \geq 1$ for $d = \alpha$. □

5. Conclusion

We have analyzed a variant of the proportional online knapsack problem with removal, in which an algorithm has access to a limited amount of additional recourse. We have given tight bounds in the case where it has access to this resource a constant number of times for each new item. In the case where its access is limited over the entire instance, we have given tight bounds for $1 \leq k \leq 3$ uses of recourse. Whether the lower bound of Theorem 1 is tight for $k \geq 4$ remains as an open question. If it were shown to be tight, this would imply an interesting balance: an algorithm with access to $k+1$ uses of recourse in total could achieve a better competitive ratio than one with access to k uses in every single step.

We further showed that the competitive ratio of this type of algorithm can be strongly improved if it is informed when the instance ends. We gave an upper bound for the competitive ratio in the case where the number of uses of recourse is constant over the entire instance and slightly improved it for the case of a constant number of uses for each new item. As is, it is unclear whether being able to use recourse in more than one step is actually helpful. Further research might try to answer this question. It might also analyze what the influence of the alternative way of communicating the end of an instance mentioned in Subsection 4.1 on the competitive ratio would be.

Other variants that could be considered might restrict the immediate accessibility of the removal and/or retrieval of items. For example, removing or retrieving items might come at a certain cost, or the buffer from which items can be retrieved might be limited in the number or combined size of items it can contain.

Lastly, our focus was entirely on the proportional knapsack problem in this paper. It is not hard to see that the general knapsack problem with removal is not competitive, even with additional, limited, recourse: Consider a single item of size and value 1, followed by at most n^2 items of size $1/n^2$ and value $1/n$. Any algorithm must pack the first item. If an algorithm never packs any of the small items, it will have a competitive ratio of n. But if it ever packs one of the small items and retrieves k additional ones, the instance ends. The algorithm will achieve a gain of at most $(k+1)/n$ and a competitive ratio of at least $n/(k+1)$. As $n \to \infty$, its competitive ratio is unbounded in both cases. This example however requires the algorithm to be unaware of the end of an instance. It would be interesting to see whether the problem becomes competitive if the end of the instance is communicated, as in the framework of Section 4.

CRediT authorship contribution statement

Hans-Joachim Böckenhauer: Writing – original draft, Formal analysis, Writing – review & editing, Project administration, Investigation, Supervision, Methodology, Conceptualization. **Ralf Klasing:** Methodology, Funding acquisition, Conceptualization, Writing – original draft, Investigation, Formal analysis, Writing – review & editing, Supervision. **Tobias Mömke:** Formal analysis, Writing – original draft, Investigation, Writing – review & editing, Methodology, Conceptualization, Funding acquisition. **Peter Rossmanith:** Writing – original draft, Formal analysis, Writing – review & editing, Conceptualization, Investigation, Methodology, Supervision. **Moritz Stocker:** Writing – original draft, Writing – review & editing, Methodology, Conceptualization, Investigation, Project administration, Formal analysis. **David Wehner:** Methodology, Formal analysis, Writing – original draft, Writing – review & editing, Conceptualization, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The models analyzed in this paper were first considered during the Mountain Workshop 2022 in Val d'Aran organized by Xavier Muñoz. The authors thank the workshop participants, especially Juraj Hromkovič, for valuable discussions.

Data availability

No data was used for the research described in the article.

References

[1] H.-J. Böckenhauer, R. Klasing, T. Mömke, P. Rossmanith, M. Stocker, D. Wehner, Online knapsack with removal and recourse, in: S.-Y. Hsieh, L.-J. Hung, C.-W. Lee (Eds.), Proceedings of the 34th International Workshop on Combinatorial Algorithms, IWOCA 2023, Springer, 2023, pp. 123–135.
[2] A. Marchetti-Spaccamela, C. Vercellis, Stochastic on-line knapsack problems, Math. Program. 68 (1995) 73–104, https://doi.org/10.1007/BF01585758.
[3] J. Boyar, L.M. Favrholdt, C. Kudahl, K.S. Larsen, J.W. Mikkelsen, Online algorithms with advice: a survey, ACM Comput. Surv. 50 (2) (2017) 19:1–19:34, https://doi.org/10.1145/3056461.
[4] D. Dwibedy, R. Mohanty, Semi-online scheduling: a survey, Comput. Oper. Res. 139 (2022) 105646, https://doi.org/10.1016/j.cor.2021.105646.
[5] P. Rossmanith, On the advice complexity of online edge- and node-deletion problems, in: H.-J. Böckenhauer, D. Komm, W. Unger (Eds.), Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday, in: Lecture Notes in Computer Science, vol. 11011, Springer, 2018, pp. 449–462.
[6] K. Iwama, S. Taketomi, Removable online knapsack problems, in: P. Widmayer, F.T. Ruiz, R.M. Bueno, M. Hennessy, S.J. Eidenbenz, R. Conejo (Eds.), Proceedings of the 29th International Colloquium on Automata, Languages and Programming, ICALP 2002, in: Lecture Notes in Computer Science, vol. 2380, Springer, 2002, pp. 293–305.
[7] H.-J. Böckenhauer, E. Burjons, J. Hromkovič, H. Lotze, P. Rossmanith, Online simple knapsack with reservation costs, in: M. Bläser, B. Monmege (Eds.), Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, in: LIPIcs, vol. 187, Schloss Dagstuhl - Leibniz-Zentrum Für Informatik, 2021, pp. 16:1–16:18.
[8] M. Imase, B.M. Waxman, Dynamic Steiner tree problem, SIAM J. Discrete Math. 4 (3) (1991) 369–384.
[9] A. Gupta, A. Kumar, Online Steiner tree with deletions, in: SODA, SIAM, 2014, pp. 455–467.
[10] N. Megow, M. Skutella, J. Verschae, A. Wiese, The power of recourse for online MST and TSP, SIAM J. Comput. 45 (3) (2016) 859–880.
[11] A. Gu, A. Gupta, A. Kumar, The power of deferral: maintaining a constant-competitive Steiner tree online, SIAM J. Comput. 45 (1) (2016) 1–28.
[12] N. Megow, L. Nölke, Online minimum cost matching with recourse on the line, in: Approx-Random, in: LIPIcs, vol. 176, Schloss Dagstuhl - Leibniz-Zentrum Für Informatik, 2020, pp. 37:1–37:16.
[13] S. Angelopoulos, C. Dürr, S. Jin, Online maximum matching with recourse, J. Comb. Optim. 40 (4) (2020) 974–1007.
[14] J. Boyar, L.M. Favrholdt, M. Kotrbcík, K.S. Larsen, Relaxing the irrevocability requirement for online graph algorithms, Algorithmica 84 (7) (2022) 1916–1951.
[15] X. Han, Y. Kawase, K. Makino, Randomized algorithms for online knapsack problems, Theor. Comput. Sci. 562 (2015) 395–405, https://doi.org/10.1016/j.tcs.2014.10.017.
[16] X. Han, Y. Kawase, K. Makino, H. Guo, Online removable knapsack problem under convex function, Theor. Comput. Sci. 540 (2014) 62–69, https://doi.org/10.1016/J.TCS.2013.09.013.
[17] X. Han, Q. Chen, K. Makino, Online knapsack problem under concave functions, Theor. Comput. Sci. 786 (2019) 88–95, https://doi.org/10.1016/J.TCS.2018.03.025.
[18] L. Gourvès, A. Pagourtzis, Removable online knapsack with bounded size items, in: H. Fernau, S. Gaspers, R. Klasing (Eds.), Proceedings of the 49th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2024, in: Lecture Notes in Computer Science, vol. 14519, Springer, 2024, pp. 283–296.
[19] H. Kellerer, V. Kotov, M.G. Speranza, Z. Tuza, Semi on-line algorithms for the partition problem, Oper. Res. Lett. 21 (5) (1997) 235–242, https://doi.org/10.1016/S0167-6377(98)00005-4.
[20] S. Dobrev, R. Královič, D. Pardubská, How much information about the future is needed?, in: V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, M. Bieliková (Eds.), Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2008, in: Lecture Notes in Computer Science, vol. 4910, Springer, 2008, pp. 247–258.
[21] Y. Emek, P. Fraigniaud, A. Korman, A. Rosén, Online computation with advice, Theor. Comput. Sci. 412 (24) (2011) 2642–2656, https://doi.org/10.1016/j.tcs.2010.08.007.
[22] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, T. Mömke, On the advice complexity of online problems, in: Y. Dong, D.-Z. Du, O.H. Ibarra (Eds.), 20th International Symposium on Algorithms and Computation, ISAAC 2009, in: Lecture Notes in Computer Science, vol. 5878, 2009, pp. 331–340.
[23] J. Hromkovič, R. Královič, R. Královič, Information complexity of online problems, in: P. Hlinený, A. Kucera (Eds.), Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science, MFCS 2010, in: Lecture Notes in Computer Science, vol. 6281, Springer, 2010, pp. 24–36.
[24] D. Komm, An Introduction to Online Computation - Determinism, Randomization, Advice, Texts in Theoretical Computer Science. An EATCS Series, Springer, 2016.
[25] X. Han, Y. Kawase, K. Makino, Online unweighted knapsack problem with removal cost, Algorithmica 70 (1) (2014) 76–91, https://doi.org/10.1007/s00453-013-9822-z.
[26] L.-H. Chen, L.-J. Hung, H. Lotze, P. Rossmanith, Online node- and edge-deletion problems with advice, Algorithmica 83 (9) (2021) 2719–2753, https://doi.org/10.1007/s00453-021-00840-9.
[27] H.-J. Böckenhauer, D. Komm, R. Královič, P. Rossmanith, The online knapsack problem: advice and randomization, Theor. Comput. Sci. 527 (2014) 61–72, https://doi.org/10.1016/j.tcs.2014.01.027.
[28] H.-J. Böckenhauer, J. Dreier, F. Frei, P. Rossmanith, Advice for online knapsack with removable items, CoRR, arXiv:2005.01867, 2020.