RWTH AACHEN UNIVERSITY

Going offline — Delays, reservations and predictions in online computation

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation vorgelegt von

Henri Lotze, M.Sc. aus Düsseldorf

Berichter: Universitätsprofessor Dr. Peter Rossmanith
Universitätsprofessorin Joan Boyar, Ph.D.

Tag der mündlichen Prüfung: 13.11.2023
Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.

Contents

Acknowledgements

My thanks go out to many people. First of all, I want to thank my supervisor Peter for all the lessons in typesetting, the discussions about any social and political topics that came up over many cups of tea and of course the — albeit sometimes slightly chaotic — very productive and helpful research sessions. Thank you also to my colleagues Daniel, Elisabet, Jan and Matthias for the research sessions and the shared juggling of all the teaching over the years.

It would be an injustice not to mention the support of our secretary Birgit Willms, without whom I would have gray hair by now. Thank you for helping me navigate all the insanity that our university administration tends to throw at us.

Having a big network of friends and a supportive family helped a lot — especially during the pandemic—, and I am grateful for all the support that I have received over these past years from all of them. Thank you for believing in me when I did not. I would like to give a special thank you to my friends Janosch and Martin for always lending an ear and giving helpful advice whenever I needed it.

Research is rarely done alone, and while I have to officially claim the results of some papers in this thesis, they would have never seen the light of day were it not for my many co-authors (and co-authors to be for yet unpublished work) Niklas Berndt, Hans-Joachim Böckenhauer, Elisabet Burjons, Li-Hsuan Chen, Jan Dreier, Janosch Fuchs, Fabian Frei, Matthias Gehnen, Juraj Hromkovic, Ling-Ju Hung, Ralf Klasing, Daniel Mock, Peter Rossmanith and Moritz Stocker. Thank you all for an always friendly and respectful research environment. I would also like to thank Prof. Boyar for her helpful feedback.

Lastly, I want to thank Johanna. Without your support, I would have not finished this thesis and given up many times on my way. Thank you for listening to all my insecurities, complaints and worries over all these years.

Abstract

The study of classical online computation builds upon a rather simple model. The input to a given problem is not known in advance and is revealed element by element, to which an algorithm is supposed to output an immediate and final decision, e.g. whether to add an element to its solution set or not.

While this model is very well suitable to compute worst case bounds to a broad range of problems, there are some problems which are not well suited for this analysis. Naturally, problems that do not admit a constant approximation ratio in their offline formulation do not admit a constant competitive ratio in the worst case in the online formulation. However, some problems that are approximable still do not admit a constant competitive ratio when analyzed in the online case. For a subset of these problems, such as the Simple Knapsack or Minimum Vertex Cover problem — and their generalizations —, the reason for this seems to be that a single, specific bad decision by an algorithm can be arbitrarily punished by an adversary.

There has been extensive research for such problems on how to modify either the set of problem instances or the online model itself and to study the performance of algorithms under such modified models and problems. An issue with many of these modifications is that they are very specific to a given problem. For example, popular modifications to the Knapsack problem are adding additional bins, adjusting the size of the knapsack of the algorithm by resource augmentation or to allow for the removal of items. Arguably, only the last modification is easily transferable to other problems.

In this thesis, we explore modifications of the classical online model with the aim to find natural models that cover a large range of problems and on the other hand work against pathological kinds of instances that restrict an algorithm from obtaining a constant competitive ratio. To this end, we study three modifications of the classical online model.

The first modification that we call *delayed decision* was first described by Boyar et al. as the *late accept* model. It can be applied to problems in which a newly presented element of the instance does not invalidate the current intermediate solution of the online algorithm or otherwise warrants an immediate decision. The family of problems that we analyze under this model, mostly with advice, is that of node- and edge-deletion problems under some finite obstruction set $\mathcal{F}$. We find that, depending on the obstruction set, the advice complexity for this model varies between advice linear in the size of the optimal solution, to no advice being needed to optimally solve such a deletion problem.

The second modification is that of *reservation costs*, in which a cost in regards to the measure of quality of an item for an algorithm can be paid when a new element is revealed. The decision on what to do with this item — commonly whether to add it to the solution set or to discard it — may then be delayed up to the end of the instance. We analyze the Simple Knapsack problem where the cost of reservation is a fixed fraction of the size of the reserved item. We find that for such a proportional cost factor α between 0 and 1, the competitive ratio is described by one of four function segments, depending on the concrete value of α under which the problem is analyzed. This modification is naturally applicable to problems where each item is associated with a gain, but generalizes quite broadly if one defines a sensible measure of cost.

Finally, we analyze a modification of a relative error on an instance, which we name *bounded predictions*. In this setting, the complete instance is announced beforehand. However, any element that is revealed may deviate from its announcement by a factor, which is known to the algorithm. Again, we analyze the Simple Knapsack problem under this modification, finding that a constant competitive ratio can be achieved for a relative error smaller than one. The resulting function of the competitive ratio depending on the allowed margin of error behaves curiously, with alternating strategies dominating in a periodic fashion.

Zusammenfassung

Die Forschung zu Online-Algorithmen basiert auf einem recht simplen Modell. Die Eingabe für ein gegebenes Problem ist zu Anfang nicht bekannt und wird erst Stück für Stück aufgedeckt, woraufhin ein Algorithmus unmittelbar und unwiderruflich Entscheidungen treffen muss, z.B. ob ein Element der Eingabe in seine Lösungsmenge aufgenommen werden soll oder nicht.

Während dieses Modell sehr gut dazu geeignet ist um worst-case-Schranken für eine große Breite von Problemen zu berechnen, gibt es einige Probleme, die sich nicht gut für diese Art von Analyse eignen. Selbstverständlich sind Probleme, die bereits in ihrer Offline-Formulierung keine konstante Approximationsrate erlauben auch in ihrer Online-Formulierung nicht durch eine konstante Kompetitivitätsrate abschätzbar. Es gibt jedoch einige Probleme, welche zwar approximierbar sind, jedoch keine konstante Kompetitivitätsrate in ihrer Online-Formulierung erlauben. Für eine Teilmenge dieser Probleme, wie das Simple Knapsack- und das Minimum Vertex Cover-Problem — und deren Generalisierungen —, scheint der Grund für dieses Verhalten in einzelnen, sehr spezifischen, schlechten Entscheidungen eines Algorithmuses zu liegen, welche ein böser Gegenspieler willkürlich ausnutzen und bestrafen kann.

Es gibt bereits eine große Menge an Forschung dazu, wie man entweder die Probleme selber oder das Online-Modell als Ganzes modifizieren kann um die Leistungsfähigkeit von Algorithmen unter diesen zu untersuchen. Ein Problem mit vielen dieser Formulierungen ist, dass diese sehr problemspezifisch formuliert sind. Für das Knapsack-Problem etwa sind beliebte Modifikationen das hinzufügen von zusätzlichen Behältern, die Adjustierung der Rucksackgröße durch das Verfahren der resource augmentation oder die Möglichkeit, Elements aus einem Rucksack wieder zu entfernen. Alleine die letzte Modifikation lässt sich einfach auf andere Probleme transferieren.

In dieser Dissertation erkunden wir Modifikationen des klassischen Online-Modells mit dem Ziel, dabei möglichst natürliche Modelle zu finden, welche einerseits auf eine breite Masse an Problemen angewandt werden können und sich andererseits gegen pathologische Arten von Instanzen wenden, welche einen Algorithmus davon abhalten, eine konstante Schranke auf seine Kompetitivitätsrate zu erhalten. Zu diesem Zwecke untersuchen wir drei Modifikationen des klassischen Online-Modells.

Die erste dieser Modifikationen ist die der *delayed decisions*, welche zuerst von Boyar et al. als das *late accept*-Modell eingeführt wurden. Dieses kann auf Probleme angewandt werden, in welchen ein neu präsentiertes Element der Eingabe

nicht die bisherige Zwischenlösung eines Algorithmuses invalidiert oder auf anderem Wege eine unmittelbare Entscheidung verlangt. Die Familie von Problemen, die wir nach diesem Modell analysieren, hauptsächlich mit advice, ist die der Knoten- und Kanten-Löschungsprobleme bedingt durch eine beschränkte Menge $\mathcal{F}$ von verbotenen Graphen. Wir zeigen, dass abhängig von der verbotenen Menge, die advice-Komplexität für dieses Modell zwischen linearem und überhaupt keinem advice in der Größe der optimalen Lösung nötig ist, um das Problem optimal zu lösen.

Die zweite Modifikation ist die der *reservation costs*, bei der ein Preis in Abhängigkeit von dem Maß an Qualität gezahlt werden kann, die ein Element für einen Algorithmus hat, wenn dieses aufgedeckt wird. Die Entscheidung, was dann mit diesem Element geschehen soll — üblicherweise ob es zur Lösungsmenge hinzugefügt werden soll oder abgelehnt wird — darf in diesem Fall bis zum Ende der gesamten Instanz hinausgezögert werden. Wir untersuchen das Simple Knapsack-Problem, in dem die Kosten einer Reservierung ein festgelegter Anteil der Größe eines Elements ist. Wir zeigen, dass die Kompetitivitätsrate für solch einen proportionalen Kostenfaktor α zwischen 0 und 1 durch eine von insgesamt vier Funktionssegmenten beschrieben werden kann, abhängig vom konkreten Wert von α. Diese Modifikation lässt sich sehr natürlich auf Probleme anwenden, in denen jedem Element der Eingabe ein Gewinn zugeordnet ist, lässt sich jedoch recht einfach verallgemeinern, wenn ein Kostenmaß bedacht gewählt wird.

Zuletzt analysieren wir eine Modifikation von relativen Fehlern auf einer Instanz, welche wir *bounded predictions* nennen. In diesem Modell ist die gesamte Instanz im Vorhinein bekannt. Jedoch kann jedes anschließend aufgedeckte Element der Instanz um einen Faktor, welche dem Algorithmus bekannt ist, von seiner angekündigten Größe abweichen. Auch unter dieser Modifikation untersuchen wir das Simple Knapsack-Problem, mit dem Ergebnis, dass eine konstant beschränkte Kompetitivitätsrate bis zu einem relativen Fehlerfaktor von 1 erreicht werden kann. Die Funktion der Kompetitivitätsrate abhängig von dem erlaubten Grad des Fehlers verhält sich eigenartig, indem drei alternierende Strategien periodisch dominieren.

Chapter 1

Introduction

When having to make decisions, one commonly would like to optimize their choices in the face of an uncertain future. Should one buy shares in a company, buy or rent a vehicle or make a down payment for it? Where should a city build a new facility if only the past demands are known and the future demands may differ arbitrarily? If all that is known is the past and the current time, how can we make decisions that minimize risk? Answering these questions means analyzing the space of possible future events and finding strategies that provably bound some cost measure for the decision maker.

1.1 Online Computations in the Classical Model

When bounding costs of such so-called *online problems*, there are two natural ways to measure the quality of any given strategy. One is to ask which strategy minimizes the *average* costs for the decision maker over some probability distribution on all possible events. The decision maker thus obtains knowledge about the expected costs (or expected gain) of a decision, within a bounded uncertainty. The other is to bound the *maximum possible* costs, which is more pessimistic, but gives a guarantee on the worst-case risk of decisions. While both schools of thought are reasonable, the clear focus of the research community has been on the latter case, with which we deal in this thesis. This study of *competitive analysis* was started by Sleator and Tarjan [65] in 1984 in which they studied the performance of list update and paging rules. We first formally define an *online problem*, *online algorithm* and the framework of competitive analysis very close to the definitions as given in the standard books by Borodin and El-Yaniv [18] and Komm [51].

Definition 1.1.1 (Online Problem [51])**.** An online problem $\mathcal{P}$ consists of a set of instances $\mathcal{I}$, a set of solutions $\mathcal{O}$. Every instance $I \in \mathcal{I}$ is a finite sequence of requests $I = (x_1, \ldots, x_n)$ and every output is a sequence of answers $O = (o_1, \ldots, o_n)$.

Every set of answers O is associated with some quality measurement function that we call either *cost* or *gain*, depending on whether $\mathcal{P}$ is a minimization or maximization problem. An optimal solution for an instance I is then the output O that minimizes $cost(I, O)$ or maximizes $gain(I, O)$.

An online algorithm is a special kind of greedy approximation algorithm. Informally, an online algorithm is oblivious to the nature of a given instance and reads it element by element. Whenever the next element is given to the algorithm, it irrevocably applies some function to this element. Most commonly, this function determines whether to add or leave out an element of a solution set. While this function is tied to the solution that an online algorithm computes, it can be more complex than a simple inclusion/exclusion decision. An example is the Coloring problem in which the function determines the (legal) coloring of a given node. The number of colors used is then the size of the solution.

Definition 1.1.2 (Online Algorithm [51])**.** Let $\mathcal{P}$ be an online problem and $I = (x_1, \ldots, x_n)$ be an instance of $\mathcal{P}$. An online algorithm $\texttt{ALG}$ for $\mathcal{P}$ computes the output $\texttt{ALG}(I) = (o_1, \ldots, o_n)$, where o_j only depends on $x_1, \ldots, x_j$ and $o_1, \ldots, o_{j-1}$. $\texttt{ALG}(I)$ has to be a feasible solution for I.

Measuring the performance of online algorithms is commonly done through the already mentioned framework of *competitive analysis*, which is a type of worst-case analysis. To determine the performance of a specific online algorithm, its solution size is compared relative to the solution size of an optimal algorithm on the same instance. The worst such ratio over all instances is then called the *competitive ratio* of this algorithm.

Definition 1.1.3 (Competitive ratio [51])**.** Given an optimization problem $\mathcal{P}$ and the set of all instances of this problem $\mathcal{I}_\mathcal{P}$. Let $\texttt{ALG}$ be an online algorithm and $\texttt{OPT}$ be an optimal algorithm for $\mathcal{P}$. For an $I \in \mathcal{I}$, let $\texttt{ALG}(I)$ and $\texttt{OPT}(I)$ be the respective solution sizes on I. We call an algorithm c-competitive for a $c \geq 1$ and a non-negative constant α if for every instance $I \in \mathcal{I}_\mathcal{P}$ it holds that

$$\texttt{ALG}(I) \leq c \cdot \texttt{OPT}(I) + \alpha,$$

assuming that $\mathcal{P}$ is a minimization problem. If it is a maximization problem, we call $\texttt{ALG}$ c-competitive if

$$\texttt{OPT}(I) \leq c \cdot \texttt{ALG}(I) + \alpha.$$

For a given online algorithm $\texttt{ALG}$ the *competitive ratio* is then the smallest c for which this inequality holds. If $\alpha = 0$, we call $\texttt{ALG}$ *strictly c-competitive*.

We also say that an algorithm is *constantly competitive* if its competitive ratio is bounded by a constant. If there is no algorithm with a constantly bounded competitive ratio for a problem, we say that the problem is *non-competitive*. Finally,

we sometimes speak of an *adversary* that "prepares instances". This is a way of seeing online algorithms as an interactive game: If an online algorithm is fully deterministic, one can imagine an agent, the adversary, giving items to an algorithm, looking at its output and selecting the next item of the instance according to this behavior, with the aim to maximize the competitive ratio of the algorithm. This is also a common view in the worst-case analysis of offline algorithms.

1.2 Ails of the Classical Online Model

The core of this work is to study modifications of the definition of online algorithms. While very sensible for most online problems, there are problems that do not admit any constant competitive ratio when analyzed under this framework. This is not surprising for problems that are not even approximable by a constant factor in their offline version, such as the Dominating Set problem, as online algorithms are a subclass of approximation algorithms. However, some problems do admit constant approximation ratios like the Knapsack problem and the Minimum Vertex Cover problem, but in their respective online setting, one can prove that their competitive ratio is indeed unbounded.

In this section, we first formally define the Knapsack and Minimum Vertex Cover problems and then look at the folklore lower bounds that show non-competitiveness in the classical online model for each.

Definition 1.2.1 (The Knapsack Problem)**.** Given a set of items $I = (x_1, \ldots, x_n)$ and let $w : I \to [0,1] \subseteq \mathbf{Q}$ be a size function and $g : I \to Q$ be a gain function. The Knapsack problem is to find a subset $S \subseteq I$, such that $\sum_{i \in S} w(x_i) \leq 1$, that maximizes the sum $\sum_{i \in S} g(x_i)$.

If $w(x_i) = g(x_i)$ for each item of I, we call this problem the Simple Knapsack problem. In the corresponding online version of the same problem, the item sequence is in advance not known to the algorithm.

Definition 1.2.2 (The Online Knapsack Problem)**.** Given a set of items $I = (x_1, \ldots, x_n)$ as a request sequence of items that arrive sequentially. Let $w : I \to [0,1] \subseteq \mathbf{Q}$ be a size function and $g : I \to Q$ be a gain function. Let K be an initially empty set called the *knapsack*. At each step $i \in \{1, \ldots, n\}$, the item x_i of the request sequence is given. An algorithm then has one of the following options:

- **Pack** If $\sum_{x_k \in K} w(x_k) + w(x_i) \leq 1$, set $K := K \cup x_i$.
- **Reject** Do nothing.

The Online Knapsack problem is then for an algorithm ALG to maximize the sum of gains of items in K, i.e.

$$gain_{\texttt{ALG}}(I) := \sum_{x_k \in K} g(x_k)\,.$$

For the Online Knapsack problem, a family of two instances is sufficient to show the non-competitiveness of any deterministic online algorithm. This example was introduced by Marchetti-Spaccamela and Vercellis [57].

Example 1.2.1. Let $\varepsilon > 0$. We define a set of two instances of the Online Knapsack problem: $\{((x_1, x_2), (x_1))\}$, where $g(x_1) = w(x_1) = \varepsilon$ and $g(x_2) = w(x_2) = 1$. As both instances share the same prefix, they cannot be distinguished upon the arrival of the first item. Any algorithm that will deterministically pack the first item is presented the first instance. Any algorithm that deterministically discards the first item is given the second instance. In both cases, the competitive ratio of the respective algorithm is unbounded.

Next, we take a look at the Minimum Vertex Cover problem, where a similar difficulty arises.

Definition 1.2.3 (The Minimum Vertex Cover Problem)**.** Given a graph $G = (V, E)$, the Minimum Vertex Cover Problem is to find a minimum subset $S \subseteq V$ such that $G[V \setminus S]$ contains no copy of $\circ\!\!-\!\!\circ$.

The Minimum Vertex Cover problem is a member of the family of $\mathcal{F}$-Node-Deletion problems, which generalize the forbidden graph $\mathcal{F} = \{\circ\!\!-\!\!\circ\}$ of the Minimum Vertex Cover problem to an arbitrary, but fixed and finite family $\mathcal{F}$ of forbidden graphs.

In the online variant of the problem, the graph is revealed node by node, with decisions on the membership of a node in the vertex cover having to be made immediately.

Definition 1.2.4 (The Online Minimum Vertex Cover Problem)**.** Let $\mathcal{F} = \{\circ\!\!-\!\!\circ\}$ be a fixed family of graphs. Given an online graph G induced by its nodes $V(G) = \{v_1, \ldots, v_n\}$, ordered by their occurrence in an online instance. Whenever a new node is given, this node either has to be added to the solution set S or rejected. The Online Minimum Vertex Cover problem is then to find a minimum S such that $G[V \setminus S]$ contains no copy of $\circ\!\!-\!\!\circ$.

Again, a family of two instances is sufficient to show that no deterministic online algorithm can be constantly competitive.

Example 1.2.2. We define a set of two instances for the Online Minimum Vertex Cover problem: {  ,  }, where the second graph is a star graph of degree k for some $k \in \mathbf{N}$.

In both instances, the node marked in gray is presented first. As both instances share the same prefix, they cannot be distinguished upon the arrival of this first node. Any algorithm that will deterministically include the gray node in its solution will be given the first instance, while any algorithm that will not do so is given the second instance. In the first case, the solution set of the algorithm is of size one, while the optimal solution is of size zero. In the second case, the algorithm is forced to take every node apart from the central gray node into its solution of total size k, while the optimal solution is of size one.

Thus, no deterministic online algorithm can be constantly competitive.

One may rightfully argue that these counterexamples are of a rather pathological nature. In the Online Knapsack problem, an online algorithm suffers from having no choice but to pack the very first item, as it otherwise risks not being competitive at all. At the same time, the sum of both items in the two-item instance is arbitrarily close to 1, but by definition they may not fit together. In the case of the Online Minimum Vertex Cover problem, a valid criticism of strictly applying the online model is that a decision on an isolated node has to be made, without it being clear if there even is a single edge in the final graph.

Maybe not unexpectedly, it turns out that even slightly loosening the restrains of the online model or of the problems themselves helps to greatly improve the performance of an online algorithm.

1.3 Structure of this Work

In this thesis, our aim is to explore variations of the classical online model by using new combinations of existing model variations as well as introducing new alternatives to analyze online problems. In this chapter, we start with an overview of the concrete models under which we analyze problems, giving formal definitions and connections to related models. We end this chapter with a bibliographical note that details on which publications the following chapters are based on.

The main chapters of this thesis then revolves around applying the models of the following sections to the Online Knapsack problem and a generalization of the Online Minimum Vertex Cover problem, namely general node-deletion problems as well as the closely related edge-deletion problems.

We start with the latter of the two in Chapter 2, which revolves around analyzing node- and edge-deletion problems in the classical online model, the late accept model and, as the main focus, the late accept model with advice, which

are all defined in the following sections. This last type of analysis turns out to be quite involved, which means that the picture regarding advice complexity is still incomplete and tight bounds are only shown for restricted subsets of problems of this kind.

The second problem we look at is the simple Online Knapsack problem from two different perspectives, which we properly introduce and define in the following sections. The first one is the model of *reservations*. In it, on the arrival of a new item a fee can be paid in order to postpone the decision on this item, essentially making the specific decision *offline*. This fee can be defined as a proportional or relative cost on the given item in regards to the total gain (or cost, in a minimization problem) of an online algorithm. In Chapter 3, we analyze the competitive ratio of the simple Online Knapsack problem dependent on relative costs $\alpha \in [0, 1]$ on each item. We also analyze the case in which reservations are *reimbursed* when a reserved item is packed by the algorithm.

The second perspective on the simple Online Knapsack problem is a model of *bounded predictions*, in which an online algorithm is given the complete instance of a simple knapsack problem in advance. As a caveat, each of these announcements is subject to some bounded noise, either proportional to the item size or absolute. In contrast to the model of *smoothed* analysis, this noise is not assumed to be random, but controlled by an adversary. An algorithm thus has a guarantee in which size range each item of the instance has to lie in, as well as precise knowledge of the length of the instance. We analyze the competitive ratio of the simple Online Knapsack problem dependent on the so-called *distortion* $\delta \in [0, 1]$, which determines by how much the actual size of an item may deviate from its announced size.

1.4 Advice Complexity in Online Computation

When analyzing the competitive ratio of an online problem, the assumption of the model is that nothing about the upcoming instance is known and that an online algorithm is completely deterministic. An interesting question to ask is then how additional information that is given to an online algorithm can be used to improve its competitive ratio. For example, an algorithm could have access to one bit of information that is truthfully provided to it at the start of its run. It could use this bit to decide which of two deterministic strategies it should execute on the concrete input.

This concept, with the central question of *how much* information an algorithm is missing in order to improve its competitive ratio was first published by Dobrev et al. [33]. Hromkovič et al. [46] later revised this model, which was further refined by Böckenhauer et al. [15] into a formalization that has been the established standard

in the community. The previous iterations of the model contained defects that allowed for side-channeling additional information by the oracle implicitly being able to denote the end of the advice string. The amount of information measured in bits that is necessary to improve the competitive ratio of an algorithm has been coined *advice complexity*.

We next give formal definitions for an online algorithm with advice and competitive ratios with advice, taken from the standard book by Komm [51].

Definition 1.4.1 (Advice Complexity [51])**.** Let Π be an online problem and let $I = (x_1, \ldots, x_n)$ be an instance of Π. An *online algorithm* $\texttt{ALG}$ *with advice* for Π computes the output $\texttt{ALG}^{\phi}(I) = (y_1, \ldots, y_n)$, where y_i depends on $\phi, x_1, x_2, \ldots, x_i$ and $y_1, \ldots, y_{i-1}$; ϕ denotes a *binary advice string*.

Definition 1.4.2 (Competitive Ratio with Advice [51])**.** Let Π be an online problem, let $\texttt{ALG}$ be a consistent online algorithm with advice for Π, and let $\texttt{OPT}$ be an optimal offline algorithm for Π. For $c \geq 1$, $\texttt{ALG}$ is *c-competitive with advice complexity* $b(n)$ for Π if there is a non-negative constant α such that, for every instance $I \in \mathcal{I}$, there is an advice string ϕ such that

$$gain(\texttt{OPT}(I)) \leq c \cdot gain(\texttt{ALG}^{\phi}(I)) + \alpha$$

if Π is a maximization problem, or

$$cost(\texttt{ALG}^{\phi}(I)) \leq c \cdot cost(\texttt{OPT}(I)) + \alpha$$

if Π is a minimization problem, and $\texttt{ALG}$ uses at most the first $b(n)$ bits of ϕ. If the above inequality holds for $\alpha = 0$, $\texttt{ALG}$ is called *strictly c-competitive with advice complexity* $b(n)$; $\texttt{ALG}$ is called *optimal with advice complexity* $b(n)$ if it is strictly 1-competitive with advice complexity $b(n)$. The *competitive ratio* of an online algorithm $\texttt{ALG}$ with advice complexity $b(n)$ is defined as

$$c_{\texttt{ALG}} := inf\{c \geq 1 \mid \texttt{ALG} \text{ is } c\text{-competitive for } \Pi \text{ with advice complexity } b(n)\}.$$

The field of advice complexity has been very active since its inception. A good overview over problems up recent years is given in the survey paper by Boyar et al. [22] and in the book by Komm [51]. There are many papers that study the advice complexity of singular problems, such as the coloring problem [64], metrical task systems [36] and scheduling [62, 34]. Some papers also give a deeper insight into the model of advice and its connection to other variations of the online model, such as randomization. Böckenhauer et al. [13] studied so-called *advice-preserving reductions* that allow, similar to the well-known reductions of complexity theory, to lower bound the amount of advice necessary to solve a given problem. The study of the Online Knapsack problem by Böckenhauer et al. [17] points out interesting limits of randomization, where one bit of randomization is as good as a bit of advice, but further random bits to not help to improve the expected competitive ratio of a randomized algorithm.

1.5 The Late Accept Model

There is a very subtle caveat in the definition of an online algorithm as we have given it. The output o_j may only depend on the previously seen input elements $x_1, \ldots, x_j$ and the output $o_1, \ldots, o_{j-1}$ that was previously computed by the algorithm, where we expect an output o_i for each element x_i of the instance.

This means that any newly presented input element of an instance demands an *immediate* and *irrevocable* output about what to do with an element at the time it arrives. This definition is however not always reasonable for all online problems. If we look at minimization problems, such as the aforementioned Minimum Vertex Cover problem, an algorithm would — strictly following the definition — immediately have to decide whether a currently isolated node is part of the computed vertex cover or not. One may argue that asking the question "Should a node be taken into a set to cover all edges of a graph?" is only reasonable to ask if it is clear whether the node *is actually part of an edge* in the graph.

If we thus try to formalize this idea, we could demand that an online algorithm still has to make immediate and irrevocable decision — however only if its current output, i.e. solution set, is no longer a feasible solution for the current prefix of the instance. This essentially means that any *addition* to the solution set of an algorithm is then the only irrevocable action of such an online algorithm.

This model, where only an incremental valid partial solution is to be upheld, was first studied by Boyar et al. [19] and coined *Late Accept* by Boyar et al. [21] in the following year. We next give a formal definition of such a model for online algorithms, which we call *delayed* for brevity.

Definition 1.5.1 (Delayed Online Algorithm)**.** Let $\mathcal{P}$ be an online minimization problem, $I = (x_1, \ldots, x_n)$ be an instance of $\mathcal{P}$ and $I_k = (x_1, \ldots, x_k)$ be the prefix of length $k \in \{1, \ldots, n\}$ of I. A *delayed online algorithm* $\texttt{ALG}$ for $\mathcal{P}$ computes the (partial) outputs $\texttt{ALG}(I_1), \ldots, \texttt{ALG}(I_n)$, with $\texttt{ALG}(I_j) \subseteq \{o_1, \ldots, o_j\}$. $\texttt{ALG}(I_j)$ only depends on $x_1, \ldots, x_j$ and $\texttt{ALG}(I_1), \ldots, \texttt{ALG}(I_{j-1})$. Furthermore, the output $\texttt{ALG}(I_j)$ has to be a feasible solution for I_j, $\texttt{ALG}(I_1) \subseteq \ldots \subseteq \texttt{ALG}(I_n)$ and $\texttt{ALG}(I_n) = \{o_1, \ldots, o_n\}$ has to hold.

Note that this definition is identical to that of a classical online algorithm if every new element invalidates the current partial solution, such as in metrical task system problems, scheduling problems and many other online minimization problems.

For online maximization problems, this model is not as reasonable, as a partial solution of an algorithm is commonly always also a valid partial solution for the complete instance. However, a marginally related model that is closest to that of a "dual" to late accepts is called *preemption*. In it, decisions of an algorithm, such as including items in a knapsack, can be irrevocably reversed [48, 44, 41].

While the Minimum Vertex Cover does not allow for competitive algorithms in the classical online setting [30], a competitive ratio of 2 can be proven in the late accept model: The upper bound is given by always taking both nodes of an uncovered edge into the solution (this is the classical 2-approximation offline algorithm). The lower bound can be achieved by presenting an edge v_iv_j and adding another edge to either v_i or v_j, depending on which node is not taken into the solution by a deterministic online algorithm. If both nodes are taken into the solution then no additional edge is introduced. This gadget can be repeated and forces a deterministic algorithm to take two nodes into the vertex cover where one suffices.

1.6 Reservation Costs

The previously introduced model of *late accepts* modifies the online model quite significantly, as the problems become "one-sided offline", i.e. only the solution set is irrevocable. Additionally, the immediacy is defused quite significantly: Decisions only need to be made once the current intermediate solution is no longer valid for the current prefix of the input.

We now look at a more fine-grained approach that retains the irrevocabilty, but relaxes the immediacy of a decision for a price. The classical model states that whenever a new element of the instance is presented, a decision has to be made right away. Commonly, this decision is either whether to include or reject the element regarding the solution set or which function to apply to it, such as assigning a cache slot in the Paging problem.

For the first kind of problem, i.e. inclusion or exclusion, we introduce a third option of *reserving* an item. Reserving an item means that the decision on the specific input element may be postponed until the end of the input. The algorithm may then in the end decide what to do with the element, assuming its choice entails a valid solution. This is of course not without drawback, as otherwise an online algorithm could simply reserve every single element of the instance and then compute an optimal offline solution. For a maximization (minimization) problem, a relative or absolute cost is deducted from the gain (added to the cost) of the algorithm.

Definition 1.6.1 (Online Algorithm with Reservation Costs)**.** Let $\mathcal{P}$ be an online problem and $I = (x_1, \ldots, x_n)$ be an instance of $\mathcal{P}$ and $I_m = (x_1, \ldots, x_m)$ be the prefix of length $k \in \{1, \ldots, n\}$ of I. An *online algorithm with reservation costs* $\texttt{ALG}$ for $\mathcal{P}$ computes the output $\texttt{ALG}(I) = (o_1, \ldots, o_n)$, where o_j only depends on $x_1, \ldots, x_j$ and $o_1, \ldots, o_{j-1}$. In any step $k \in \{1, \ldots, n\}$ the algorithm may decide to withhold the computation of o_k (i.e. *reserve* it) and output o_k only after step n. Let $J_k \subseteq I$ be the set of all withheld elements after step k. For every k,

$\texttt{ALG}(I_k - J_k)$ has to be a feasible solution for $I_k - J_k$. $\texttt{ALG}(I)$ has to be a feasible solution for I.

We assume in this definition that an online algorithm is explicitly notified about the end of the instance at the time that it expects to read a next element. It may at this point compute the output for all its reserved items. Naturally, we need to adjust the definition of a competitive ratio for such algorithms.

Definition 1.6.2 (Competitive Ratio under Reservation Costs)**.** Given an optimization problem $\mathcal{P}$ and the set of all instances of this problem $\mathcal{I}_\mathcal{P}$. Let $\texttt{ALG}$ be an online algorithm with reservation costs and $\texttt{OPT}$ be an optimal algorithm for $\mathcal{P}$. For an $I \in \mathcal{I}$, let $\texttt{ALG}(I)$ and $\texttt{OPT}(I)$ be the respective solution sizes on I and $J \subseteq I$ be the set of elements reserved by $\texttt{ALG}(I)$. Let $f : I \to \mathbf{R}^+$ be a function representing the reservation costs of an item. We define $f(\{x_1, \ldots, x_k\}) := f(x_1) + \ldots + f(x_k)$. We call an online algorithm with reservation costs c-competitive for a $c \geq 1$ and a non-negative constant α, if for every instance $I \in \mathcal{I}_\mathcal{P}$ it holds that

$$\texttt{ALG}(I) + f(J) \leq c \cdot \texttt{OPT}(I) + \alpha,$$

assuming that $\mathcal{P}$ is a minimization problem. If it is a maximization problem, we call $\texttt{ALG}$ c-competitive if

$$\texttt{OPT}(I) \leq c \cdot (\texttt{ALG}(I) - f(J)) + \alpha.$$

For a given online algorithm $\texttt{ALG}$ the *competitive ratio* is then the smallest c for which this inequality holds. If $\alpha = 0$, we call $\texttt{ALG}$ *strictly c-competitive*.

As the model of reservations is still very young, only few contributions were based on it so far. In the secretary problem [25], some improvements to the classical expected competitive ratio were possible, up to reservation costs of $\frac{1}{e}$, where e is Euler's number. In a model where reservation costs are to be paid for each round that an item is reserved, i.e. whenever the next element of the instance is presented, the advantage of reservations diminishes even quicker, with no additional benefit to reservations once the reservation costs exceed $\frac{1}{n}$.

When combining reservations with removals in the Online Simple Knapsack problem [26], the bound of the golden ratio ϕ by Iwama and Taketomi [48] can be improved, but only for reservation costs up to approximately 0.254. For higher costs of reservation, the best tactic is not to reserve any items at all.

1.7 Bounded Predictions

When acting in an online setting, the assumption that the future is completely unknown is often unrealistic. When buying boxes for moving to a different house,

one usually has a pretty good idea of what one owns and can then buy a number of boxes on this estimate. When planning to drive a group of friends home, one can already pre-plan a route with the rough knowledge on where these persons actually live.

A common theme is thus that the knowledge of the world and the future is actually not unknown, on the contrary: it is often known but the details are very uncertain. If we want to model this kind of behavior, we could thus assume that an instance is already given in advance: all elements are revealed beforehand. When the elements arrive, i.e. we have to actually pack the moving boxes or ask our friend where they live, we learn the *actual* nature of the element. For the sake of a simple model, let us assume that the actual number of elements is correctly given. We also assume that our estimates do not deviate beyond a certain bound, i.e. that the maximum uncertainty is bounded and this bound is known in advance.

Our definition deals with relative errors, i.e. errors bounded by a multiplicative constant. Note that for this definition, we normalize the prediction: Instead of an item being able to be made smaller or larger, we transform a given prediction. In this transformed prediction, we give the lowest possible value that the item can take as the prediction. This of course then modifies the possible variation of this item, resulting in a factor of $\frac{1+\delta}{1-\delta}$ in the multiplicative case.

Definition 1.7.1 (Online Algorithm with Bounded Predictions)**.** Given a constant $\delta \in [0, 1]$, called *distortion*. Let $\mathcal{P}$ be an online problem and $I = (x_1, \ldots, x_n)$ be an instance of $\mathcal{P}$. Let $P = (x_1^-, \ldots, x_n^-)$ be a sequence of elements called the *prediction* such that $x_i \in [x_i^-, x_i^+]$, where $x_i^+ := \frac{1+\delta}{1-\delta} x_i^-$. An online algorithm $\texttt{ALG}$ for $\mathcal{P}$ computes the output $\texttt{ALG}(I) = (o_1, \ldots, o_n)$, where o_j only depends on $x_1, \ldots, x_j$, on $o_1, \ldots, o_{j-1}$, on δ and on P. $\texttt{ALG}(I)$ has to be a feasible solution for I.

It is important to note the difference between the model of bounded predictions and the one of *machine learned advice*, which has recently been called *untrusted predictions* or just *predictions*. The latter has been introduced by Lykouris and Vassilvitskii [56] and works as follows. An algorithm is given a prediction on the input, just as in our model. However, no bound is given on the error, which is rather treated as a variable and algorithms are designed with the aim of meeting three characteristics. The first is that the algorithms output an optimal solution when the given prediction is correct (*consistency*), the second is that they perform as well as a regular online algorithm on the problem when the predictions become arbitrarily bad (*robustness*) and the last is that they degrade with increasing unreliability of the prediction (*smoothness*).

The essential difference that is present in the model of bounded predictions is that the algorithm is given knowledge about the maximum error that it can expect, namely δ, as part of its input in the model of bounded predictions. Classically, this error term is unknown and an algorithm is to perform as well as possible. We

believe bounded predictions to be equally reasonable as that without bounds on the predictions, as it is not unnatural for a person to be sure that their prediction will not be arbitrarily far off from the truth.

The "classical" model of predictions has seen a big influx of results in the past few years, with the model being applied to several different online problems, such as scheduling [61, 53, 20, 7, 58], metric algorithms [2, 3], matching problems [54, 32, 50], spanning tree problems [38, 10] and many more.

The model of bounded predictions is, however, not our original modification. Azar et al. [5] recently discussed scheduling problems based on this model, where they developed algorithms that can learn about the maximum distortion and make decisions based upon this value. In a more recent work, they extended their analysis to the scheduling on multiple machines [6], in which they also analyzed the case in which an algorithm is oblivious to the actual distortion of the instance.

In order to avoid confusion, as Azar et al. still speak of *problems with predictions* in their works, we took the liberty to name this specific model *bounded predictions* in this work. We believe that using this slightly modified version of the original prediction model can yield an even more fine-grained way of worst-case analysis, when one can assume that an oracle can only be wrong up to a known, bounded degree.

Where we assume that an adversary is able to control both the predicted instance and the actual distortion of the items, there is a related model of *smoothed analysis*, in which an adversary can fix an instance, which is then subject to some random (commonly Gaussian) distortion, or *noise*. This model of an adversary not having complete control over its prepared instance was first made popular when showing that the simplex algorithm runs in expected polynomial time when its input is subjected to such random noise [66]. Since then, there was a large influx of results in this area for a wide range of problems, such as multi-level feedback algorithms [8], analyzing the k-means method [4] for clustering, and the (offline) 0/1 knapsack problem [9]. The model of smoothed analysis thus gives evidence that the worst-case running time or worst-case approximation ratios often seem to suffer from very specific and limited adversarial inputs which break down if even only a very slight perturbation of the instance is given.

1.8 Bibliographical Note

The results presented in Chapter 2 stem from two publications. The first of these publication was accepted at the 31st International Workshop on Combinatorial Algorithms (IWOCA 2020) under the title

1. Li-Hsuan Chen, Ling-Ju Hung, Henri Lotze and Peter Rossmanith, "Further Results on Online-Node- and Edge-Deletion Problems with Advice," *31st*

International Workshop on Combinatorial Algorithms, (June 2020): https://doi.org/10.1007/978-3-030-48966-3_11 [28]

A longer version was subsequently published in a peer-reviewed journal:

2. Li-Hsuan Chen, Ling-Ju Hung, Henri Lotze and Peter Rossmanith, "Online Node- and Edge-Deletion Problems with Advice," *Algorithmica 83*, (2021): https://doi.org/10.1007/s00453-021-00840-9 [29]

This journal version contains all missing proofs and further results on the competitive analysis of the underlying problems. Additional results that were previously published by the co-author Peter Rossmanith in [63] have also been added to the journal version. The author claims no authorship of the additional results from [63], which are only used as a reference and are clearly cited in this work, where used.

The second publication that is the basis for the results of Chapter 2 was accepted at the 34th International Workshop on Combinatorial Algorithms (IWOCA 2023) under the title

3. Niklas Berndt and Henri Lotze, "Advice Complexity Bounds for Online Delayed $\mathcal{F}$-Node-, H-Node- and H-Edge-Deletion Problems," *34th International Workshop on Combinatorial Algorithms*, (June 2023): https://doi.org/10.1007/978-3-031-34347-6_6 [11]

Parts of the results of Section 2.4 were researched in the scope of a bachelor thesis together with Sijia Guo and have not yet undergone peer review.

Chapter 3 is based on the results of a publication that has been accepted at the 38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021) under the title

4. Hans-Joachim Böckenhauer, Elisabet Burjons, Juraj Hromkovic, Henri Lotze and Peter Rossmanith, "Online Simple Knapsack with Reservation Costs," *38th International Symposium on Theoretical Aspects of Computer Science*, (March 2021): https://doi.org/10.4230/LIPIcs.STACS.2021.16 [12]

A journal version that closes the only remaining gap in the above publication is currently under peer-review for publication in the *Journal of Computer and System Sciences*, with Fabian Frei as an additional co-author. The results of Section 3.6 were researched in the scope of a master thesis together with Rasmus Piorr and have not yet undergone peer review.

The results of Chapter 4 are based on yet unpublished work that has not yet undergone peer review. It was submitted to the 41st Symposium on Theoretical Aspects of Computer Science (STACS 2024). The authors for this work are

Matthias Gehnen, Henri Lotze and Peter Rossmanith, and the title for the publication is *Online Simple Knapsack with Bounded Predictions*.

During his studies as a PhD student, the author also contributed to the following publications that are not covered in this thesis:

5. Elisabet Burjons, Fabian Frei, Matthias Gehnen, Henri Lotze, Daniel Mock, Peter Rossmanith, "Delaying Decisions and Reservation Costs," *29th International Conference on Computing and Combinatorics*, (December 2023)

6. Elisabet Burjons, Matthias Gehnen, Henri Lotze, Daniel Mock, Peter Rossmanith, "The Online Simple Knapsack Problem with Reservation and Removability," *48th International Symposium on Mathematical Foundations of Computer Science*, (August 2023): `https://doi.org/10.4230/LIPIcs.MFCS.2023.29` [26]

7. Elisabet Burjons, Janosch Fuchs, Henri Lotze, "The Slotted Online One-Sided Crossing Minimization Problem on 2-Regular Graphs," *33th International Workshop on Combinatorial Algorithms*, (June 2022): `https://doi.org/10.1007/978-3-031-06678-8_18` [24]

8. Elisabet Burjons, Matthias Gehnen, Henri Lotze, Daniel Mock, Peter Rossmanith, "The Secretary Problem with Reservation Costs," *27th International Conference on Computing and Combinatorics*, (October 2021): `https://doi.org/10.1007/978-3-030-89543-3_46` [25]

9. Jan Dreier, Henri Lotze, Peter Rossmanith, "Hard Problems on Random Graphs," *47th International Colloquium on Automata, Languages, and Programming*, (July 2020): `https://doi.org/10.4230/LIPIcs.ICALP.2020.40` [35]

Chapter 2

Online Delayed Node- and Edge-Deletion Problems

In Chapter 1 we have defined the Minimum Vertex Cover problem and discussed the difficulties of analyzing this problem in the classical model for online computation. As a way to allow for a reasonable analysis of this problem, we have introduced the model of *late accepts* and seen in Section 1.5 that the Minimum Vertex Cover problem admits a 2-competitive online algorithm with a matching lower bound under this model.

In this chapter, we look at generalizations of the Minimum Vertex Cover problem as well as a related class of problems. In Definition 1.2.4, the problem is defined via an *obstruction set* of a single graph, namely $\mathcal{F} = \{\circ\!-\!\circ\}$. If in a graph such a subgraph, i.e. two nodes connected by an edge, can be found and neither of the nodes are covered, at least one of the two nodes of it need to be added to the solution set of the algorithm and thus be covered.

We are, however, not interested in deletion problems for specific obstruction sets, such as the Minimum Vertex Cover problem, but rather in showing upper and lower bounds for *sets of problems*. Specifically, we want to answer the question if we can find lower and upper bounds for whole classes of problems that are defined via obstruction sets. So instead of looking at a $\circ\!-\!\circ$-Node-Deletion problem, we want to look at Delayed $\mathcal{F}$-Node-Deletion problems, where $\mathcal{F}$ is an arbitrary but finite collection of finite graphs. This would mean that any graph $H \in \mathcal{F}$ that can be found in the input graph needs to be covered. Of course, this begs the question whether we are interested in keeping an online graph free of these graphs in the sense that no *induced* or *proper* subgraph of the input graph is isomorphic to any $H \in \mathcal{F}$. In this chapter, we only discuss this problem regarding induced subgraphs. We leave the case of proper subgraphs open.

To give a simple example to start off, we look at the problem of keeping a graph *trivially perfect*. One characterization of this graph class is via an obstruction set,

forbidding any induced path and cycle over four nodes, i.e. $\mathcal{F} = \{$  , $\}$. In the online problem a graph is then revealed, node by node, and an algorithm has to check whether either or can be found in the graph induced by the nodes revealed so far. If one such graph can be found, the algorithm has to cover nodes until this is no longer the case. An online algorithm is then correct if it manages to cover every induced copy of and .

A closely related problem to that of node-deletions is that of edge-deletions. The premise is the same: A fixed obstruction set is given and whenever a copy of some forbidden graph can be found in the graph, an edge of it has to be covered. We will later see that this type of problem turns out to be much harder to analyze, as the deletion of edges from one $H \in \mathcal{F}$ can result in an induced copy of some other $H' \in \mathcal{F}$. In the rest of this chapter, instead of speaking of covering nodes or edges, we speak of *deleting* nodes or edges.

While an analysis of the online problem itself is warranted, we will only briefly look at the competitive analysis of node- and edge-deletion problems and instead focus on the advice complexity of these kinds of problems, building on the work of Rossmanith [63]. Not entirely surprising, we find out that neither the competitive ratios, nor the advice complexity bounds hold tightly for *all* obstruction sets. Remarkably, we are however still able to show that problems over a wide range of obstruction sets share similar bounds.

2.1 Problem Definitions and Notation

We use standard graph notation. Given an undirected graph $G = (V, E)$, $G[V']$ with $V' \subseteq V$ denotes the graph induced by the node set V'. We use $|G|$ to denote $|V(G)|$ and $||G||$ to denote $|E(G)|$. We use the notation $\overline{G}$ for the complement graph of G. We write $G - U$ for $G[V(G) - U]$ and $G - u$ for $G - \{u\}$ and also use $G - E$ similarly for an edge set E. K_n and $\overline{K_n}$ denote the n-clique and the n-independent set respectively. P_n denotes the path on n nodes. The neighborhood of a vertex v in a graph G consists of all vertices adjacent to v and is denoted by $N^G(v)$. A vertex v is called universal, if $N^G(v) = V \setminus \{v\}$. A node v is called a *twin* of a node w if $N^G(v) \cup \{v\} = N^G(w) \cup \{w\}$. If only $N^G(v) = N^G(w)$, then v is called a *false twin* of w.

We use the symbol $\trianglelefteq$ to denote an induced subgraph relation, i.e. $A \trianglelefteq B$ iff A is an induced subgraph of B. We use $H \trianglelefteq_\varphi G$ for graphs H and G iff there exists an isomorphism φ such that $\varphi(H)$ is an induced subgraph of G. We denote by H a finite graph and by $\mathcal{F}$ a finite set of finite graphs, if not stated otherwise. For a given $\mathcal{F}$-deletion problem, we denote by *opt* the size of the optimal solution, i.e. the minimum amount of deletions — of either nodes or edges — on a given input graph G that makes it $\mathcal{F}$-free. The problem and input graph should always

be clear from the context. If we want to make clear that we are referring to a specific family $\mathcal{F}+$ or regarding a particular graph G that is to be made $\mathcal{F}$-free, we sometimes use the notation $opt_{\mathcal{F}}(G)$.

A graph G is called *$\mathcal{F}$-free* if there is no $H \trianglelefteq_{\varphi} G$ for any $H \in \mathcal{F}$. Furthermore, a *gluing* operation works as follows: Given two graphs G and G', identify a single node from G and a single node from G'. For example, if we glue  together with  at the gray nodes, the resulting graph is .

Definition 2.1.1 (The Delayed $\mathcal{F}$-Node-Deletion Problem)**.** Let $\mathcal{F}$ be a fixed family of graphs. Given an online graph G induced by its nodes $V(G) = \{v_1, \ldots, v_n\}$, which are ordered by their occurrence in an online instance. The Delayed $\mathcal{F}$-Node-Deletion problem is for every i to select a set $S_i \subseteq \{v_1, \ldots, v_i\}$ such that $G[\{v_1, \ldots, v_i\}] - S_i$ is $\mathcal{F}$-free. Furthermore, it has to hold that $S_1 \subseteq \ldots \subseteq S_n$, where $|S_n|$ is to be minimized.

The definition of the Delayed $\mathcal{F}$-Edge-Deletion problem is identical, with S being a set of edges of G instead of nodes. If $\mathcal{F} = \{H\}$ we speak of an H-Deletion Problem instead of an $\{H\}$-Deletion Problem. For the sake of brevity of notation, we shorten the name of the problems outside the statements of theorems, lemmata and corollaries: for Delayed $\mathcal{F}$-Node-Deletion, we write D$\mathcal{F}$ND, for Delayed $\mathcal{F}$-Edge-Deletion we write D$\mathcal{F}$ED.

For an obstruction set $\mathcal{F}$ we assume that there exist no distinct $H_1, H_2 \in \mathcal{F}$ with $H_1 \trianglelefteq_{\varphi} H_2$ for some isomorphism φ, as each online graph containing H_2 also contains H_1, making H_2 redundant. Furthermore, we assume in the case of the D$\mathcal{F}$ED that $\mathcal{F}$ contains no $\overline{K_n}$ for any n. This assumption is arguably reasonable as no algorithm that can only delete edges is able to remove a set of isolated nodes from a graph. By $\log(n)$ we always denote the logarithm to base 2.

Lastly, we call an advice complexity bound *essentially tight*, if its lower and upper bound match up to encoding details, where the gap is the size of an optimal solution that is explicitly communicated via additional advice. This results in an overhead of advice bits logarithmic in the size of the solution.

2.2 Related Work and Our Contributions

$\mathcal{F}$-Node- and $\mathcal{F}$-Edge-Deletion problems were given attention by the research community for a longer time now. Yannakakis [68] first gave NP-completeness proofs of these problem classes for a number of concrete graph families $\mathcal{F}$ in 1978 and showed together with Lewis in 1980 [55] that node-deletion problems for hereditary problems are NP-complete. In 1979, Krishnamoorthy and Deo [52] gave NP-hardness proofs for 17 further node-deletion problems. The study was subsequently

Table 2.1: Advice complexity bounds of delayed node- and edge-deletion problems. All entries are to be read as Θ of the respective term. "?" denotes an open problem. A trivial upper bound of $\mathcal{O}(opt \log |H|)$ exists for all stated deletion problems, where H is a maximum order graph of $\mathcal{F}$, as shown in Theorem 2.7.1, where the same argumentation works for edge-deletion problems.

	Node-Deletion		Edge-Deletion	
Forbidden family	connected	arbitrary	connected	arbitrary
$\mathcal{F} = \{H\}$	$opt \log \lvert H\rvert^a$	$opt \log \lvert H\rvert^b$	$opt \log \lvert\lvert H\rvert\rvert^a$	$opt \log \lvert\lvert H\rvert\rvert^c$
$\mathcal{F} = \{H_1, \ldots, H_{max}\}$	$opt \log \lvert H_{max}\rvert^a$	$0^d / \log opt^e / opt \log \lvert H_{max}\rvert^a / ?$	opt^f	?

[a]Rossmanith [63] [b]Thm. 2.7.1 [c]Cor. 2.5.1, Thm. 2.5.2 [d]Thm. 2.7.3 [e]Thms. 2.7.4, 2.7.5, 2.7.6 [f]Thm. 2.6.3

extended for further graph classes[69, 60]. Cai [27] studied the parameterized complexity of general graph modification problems, which are a more generalized class of these problems. Enright and Meeks [37] recently studied edge-deletion problems parameterized by the treewidth of graphs showed that the problem is in FPT under this parameter.

The research of this thesis was motivated by the paper of Rossmanith [63], which gave first results on the advice complexity of the D$\mathcal{F}$ND and D$\mathcal{F}$ED problems for restricted classes of $\mathcal{F}$. In this work, he gives essentially tight advice complexity bounds for D$\mathcal{F}$ND problems for all classes $\mathcal{F}$ that consist only of connected graphs. For D$\mathcal{F}$ED problems, the restrictions are more severe: An essentially tight bound on the advice complexity of such problems is given only for families of graphs $\mathcal{F}$ in which $\mathcal{F}$ consists of a single, connected graph H.

Our work iterates on the problems left open by Rossmanith. Table 2.1 gives an overview on which problems have been solved or partially solved in the realm of advice complexity.

Our contributions that we present in this chapter are the following. While we are mainly interested in the advice complexity of the aforementioned problems, we give some limited insights on the competitive ratio of these problems in the classical online model and the late accept model. In Section 2.3 we show that in the classical online model — perhaps to little surprise — any H-Node-Deletion problem is not competitive, assuming H is a connected graph and not a single node. Noticeably, this does not generalize, as there are families that can be optimally solved even in this very restricted models, such as families of all graphs over k nodes, for some $k \in \mathbf{N}$.

We then shortly discuss the nature of the online problem in the late accept model in Section 2.4. To this end, we give a trivial upper bound algorithm that always deletes a complete H that it finds in an input graph. We then show that

for a number of families of graphs, the resulting competitive ratio is indeed the best possible.

Turning to the analysis of advice complexity of these problems, starting in Section 2.5, we give essentially tight advice complexity bounds for D$\mathcal{F}$ND and D$\mathcal{F}$ED problems in which $\mathcal{F}$ consists of a single, but disconnected graph H. On the basis of the results of Rossmanith, we show via a simple advice preserving reduction that if $\mathcal{F}$ consists of only disconnected graphs, we can give essentially tight bounds on the advice complexity of such problems.

Rossmanith gave a tight advice complexity bound of $\Theta(opt)$ for DHED problems in which H was a connected graph. We extend these results in Section 2.5 by showing that the same holds true if H is disconnected. For the general D$\mathcal{F}$ED problem, we are able to give a general upper bound of $\Theta(opt)$, but are only able to provide a matching lower bound for families $\mathcal{F}$ of connected graphs, both in Section 2.6. We leave the analysis for arbitrary families $\mathcal{F}$ open.

We also provide results giving insight on the advice complexity of D$\mathcal{F}$ND problems for arbitrary families $\mathcal{F}$ in Section 2.7. To this end, we show that depending on $\mathcal{F}$, advice constant, logarithmic or linear in the size of the *opt* can be *necessary and sufficient* to optimally solve these problems. We are not able to fully characterize all online node-deletion problems, but strongly suspect that advice linear in *opt* is both necessary and sufficient for the remaining D$\mathcal{F}$ND problems.

2.3 The Classical Online Problem

Since the Minimum Vertex Cover problem is not competitive in the classical online model with an obstruction set containing only a single edge, it should be no big surprise that the same holds for general node-deletion problems when looking at many other families $\mathcal{F}$. The details are however not as trivial as one may think at first.

To look at said details more closely, let us first formally define these deletion problems in the classical online model.

Definition 2.3.1 (The $\mathcal{F}$-Node-Deletion Problem)**.** Let $\mathcal{F}$ be a fixed family of graphs. Given an online graph G induced by its nodes $V(G) = \{v_1, \ldots, v_n\}$, an algorithm has to finally either include or exclude a newly revealed node from its solution set S. The algorithm is correct if $G[V \setminus S]$ is $\mathcal{F}$-free before any next node is revealed. The goal is to minimize $|S|$.

Using a simple proof, let us first show that for an obstruction set of a single connected graph that is not an isolated node, no algorithm can be competitive in the classical online model.

Lemma 2.3.1. *Let H be a connected graph with $|H| > 1$. Then there is no online algorithm for the H-Node-Deletion problem that is c-competitive for any constant c.*

Proof. Let G be the online graph and $k = |H|$. The adversary starts constructing a copy H_c of H by iteratively revealing its nodes, such that a node with the smallest degree in H is revealed last. If an online algorithm deterministically deletes any node before this copy is fully revealed, the instance instead ends after exactly this node deletion. Such an algorithm would then have a solution of size 1, while the optimal solution is of size 0.

Assuming that H is completed and an algorithm has not deleted any node until this point, any correct online algorithm is now forced to delete the very last node v_k^0 of H that was revealed. The rest of the instance then consists of nodes v_k^i with $N(v_k^i) := N(v_k)$ and $i \in \mathbf{N}$. As these nodes all share the same open neighborhood, the algorithm is now confronted with another copy of H, since $\varphi(H_c) = G[V(H_c \cup v_k^i \setminus v_k^{i-1})]$ for some isomorphism φ. Its only possible option is to also delete every newly presented node v_k^i in order to maintain a correct solution.

An optimal solution for this graph deletes a node $v_j \in N(v_k^0)$. Since v_k^0 (and each of its copies) had the smallest degree in H by construction, the degree of each v_k^i is decreased by one. H has no node of degree $d(v_k^i) - 1$ and every subset of k nodes in the remaining graph includes at least one v_k^i. Since k copies of v_k^i themselves cannot form an H, as their degree is each too small, the optimal solution is indeed of size 1, while any online algorithm is forced to delete an arbitrary number of nodes. □

An important insight to be had is that the non-competitiveness of such graphs H as in Lemma 2.3.1 give no insight into the advice complexity of an $\mathcal{F}$-Node-Deletion problem if $H \subset \mathcal{F}$. To this end, we proof that two classes of families that share an H as given in Lemma 2.3.1 can admit either an optimal online algorithm or no competitive online algorithm.

Lemma 2.3.2. *Let $\mathcal{F}_k$ be the family of all graphs over k nodes for a fixed $k \in \mathbf{N}$. There exists an online algorithm optimally solving the $\mathcal{F}_k$-Node-Deletion problem.*

Proof. The algorithm simply waits for the kth node to be revealed and deletes it as well as any subsequent nodes. Any optimal algorithm must delete all but $k-1$ nodes from the graph. □

The constant k in the previous and next lemma can be chosen such that both families share the graph K_k, which should make the mentioned insight clear.

Lemma 2.3.3. *Let $\mathcal{F} = \{K_k, P_k\}$ for a fixed $k \in \mathbf{N}_{>3}$. Then there is no online algorithm for the* Delayed H-Node-Deletion *problem that is c-competitive for any constant c.*

Proof. We use the same proof as in Lemma 2.3.1 by applying the construction of the proof to a copy of K_k. Let G be the online graph and v_k^n be the copies of the node v_k^0 introduced in the proof and $i \in \mathbf{N}$. We only have to show that the graph $G - v_j$ for $v_j \in N(v_k^0)$ is $\mathcal{F}$-free, i.e. that it contains no induced K_k, nor a P_k, independent of the number of new nodes that are introduced in the construction of the proof.

The fact that the graph is K_k-free follows from the proof of Lemma 2.3.1. Since K_k is a complete graph, any potentially induced P_k either consists exclusively of newly introduced nodes v_k^i or of a mixture of the nodes of K_k and the newly introduced nodes. The former is not possible as there is no edge between any pair of nodes v_k^l and v_k^m for some $l, m \in \mathbf{N}$. Any set of three nodes taken from the set $V(K_k)$ induces a triangle. If only a single node of $V(K_k)$ is used, this node is the center of a star graph. Lastly, if two nodes of $V(K_k)$ are used, they are the shared neighbor of all nodes v_k^i, inducing a C_4 on any possible path.

Thus, the graph $G - v_j$ is $\mathcal{F}$-free. □

2.4 The Delayed Online Problem

The restriction of an algorithm to be forced to wait for a copy of a forbidden graph H to be revealed, and at the same time to be forced to delete the very last node of such an H is arguably too strong.

In this section, we want to contrast the findings of the previous section by looking at the possible competitive ratios of algorithms in the *late accept* model.

We start by showing that independent of the obstruction set $\mathcal{F}$, there always exists a simple algorithm with a constantly bounded competitive ratio for both node-deletion and edge-deletion problems.

Theorem 2.4.1. *The* Delayed $\mathcal{F}$-Node-Deletion *admits an algorithm that is k-competitive for $k = \max_{H \in \mathcal{F}}\{|H|\}$ and the* Delayed $\mathcal{F}$-Edge-Deletion *problem admits an online algorithm that is k-competitive for $k = \max_{H \in \mathcal{F}}\{||H||\}$.*

Proof. Whenever an algorithm finds an induced H, it deletes all of its nodes, respectively edges. □

This may seem like a trivial and imprecise bound at first. Yet, as we will see next, it turns out that this algorithm is actually the best possible for a number of families $\mathcal{F}$. To give the following proofs, we need a small combinatorial lemma.

Lemma 2.4.1. *Let $S = \{s_1, \ldots, s_k\}$, $T = \{t_1, \ldots, t_k\}$ be two ordered sets with $|S| = |T| = k \in \mathbf{N}$. For every subset $R \subset T \cup S$ with $|R| = k + 1$ there exists an i such that both $s_i \in R$ and $t_i \in R$.*

Proof. Assume this was not the case and that there was a subset R violating this condition. Then for every index $j \in \{1, \ldots, k\}$ only either $s_j \in R$ *or* $t_j \in R$. Since there are only k different indices to choose from, by the pigeonhole principle the $k + 1$st element then needs to be a "duplicate" as stated. □

We continue by providing matching lower bounds for Theorem 2.4.1 for two types of graphs H. The basic idea of these lower bounds is the same. If an algorithm deletes a node, we can "repair" the intermediate graph with a twin or false twin of the deleted node, depending on the concrete $\mathcal{F}$. We repeat this until the $k = |H|$th node is deleted and then argue, using Lemma 2.4.1, that only a single deletion is optimal for the same graph.

Theorem 2.4.2. *Let H be a fixed connected, triangle-free graph with $k = |H|$ and $|H| > 2$. Then any online algorithm solving the* Delayed H-Node-Deletion *problem cannot achieve a competitive ratio better than k.*

Proof. Let G be the online graph. The adversary first reveals a copy of H_c of H, forcing a deletion from any algorithm. If an algorithm deletes node v_i from the copy of H, the algorithm next reveals a twin node v_i' of v_i, i.e. $N^G(v_i') := N^G(v_i) \cup \{v_i\}$. Since v_i' shares its neighborhood with v_i and the edge $v_i v_i'$ is not part of the remaining graph after the deletion of v_i, there is again a copy of H in the graph: $\varphi(H) = G[(V(H_c) \cup v_i') \setminus v_i]$ for some isomorphism φ. This then forces another deletion of an algorithm.

For a total of $k-1$ deletions of an algorithm, this scheme is repeated. Should an algorithm delete multiple nodes at once, they are treated as if they were sequential deletions in any order. After the kth deletion of an algorithm, the instance ends. Clearly, any algorithm can be forced to delete k nodes, and there is an adversarial strategy independent of the deletion order.

We claim that deleting exactly one of the remaining nodes of H_c, called v_k, is the unique optimal solution. We thus have to show that $G - v_k$ is H-free.

As $|H| = k$, H_c missing the node v_k, and as we have introduced $k-1$ new nodes by this strategy, any remaining potential copy of H must consist of a mixture of nodes of $H_c - v_k$ and newly introduced nodes. By Lemma 2.4.1 we know that any such H must include a node v_i of H_c and its newly introduced twin v_i'. Since H_c is connected, there must be a neighbor of v_i and/or a neighbor of v_i' in the node set of this copy of H, as $|H| > 2$. However, then v_i, v_i' and this neighbor close a triangle by construction. Since H is by definition triangle-free, this leads to a contradiction. Thus, $G - v_k$ is H-free. □

The second lower bound deals with a "complementary" graph class of cliques, which are fully triangulated. We use a very similar proof technique, with the essential difference being that the upcoming contradiction stems from the lack of triangulation, in contrast to the existence of a triangle in the previous lemma.

Theorem 2.4.3. *Let $H = K_k$ be a clique over k nodes with $|H| > 2$. Then any online algorithm solving the* Delayed $\mathcal{F}$-Node-Deletion *problem cannot achieve a competitive ratio better than k.*

Proof. Let G be the online graph. The adversary first reveals a copy of H_c of H, forcing a deletion from any algorithm. If an algorithm deletes node v_i from the copy of H, the algorithm next reveals a false twin node v_i' of v_i, i.e. $N^G(v_i') := N^G(v_i)$. Since v_i' shares its neighborhood with v_i, there is again a copy of H in the graph: $\varphi(H) = G[(V(H_c) \cup v_i') \setminus v_i]$ for some isomorphism φ. This then forces another deletion of an algorithm.

For a total of $k-1$ deletions of an algorithm, this scheme is repeated. Should an algorithm delete multiple nodes at once, they are treated as if they were sequential deletions in any order. After the kth deletion the instance ends. Clearly, any algorithm can be forced to delete k nodes, and there is an adversarial strategy independent of the deletion order.

We claim that deleting exactly one of the remaining nodes of H_c, called v_k, is the unique optimal solution of the same instance. We thus have to show that $G - v_k$ is H-free.

As $|H| = k$, H_c missing the node v_k, and as we have introduced $k-1$ new nodes by this strategy, any remaining potential copy of H must consist of a mixture of nodes of $H_c - v_k$ and newly introduced nodes. By Lemma 2.4.1 we know that any such H must include a node v_i of H_c and its newly introduced twin v_i'. However, since there is no edge between v_i and v_i', they cannot be part of the same clique, leading to a contradiction. Thus, $G - v_k$ is H-free. □

We leave open the competitive analysis of general families for node-deletion problems and for edge-deletion problems.

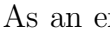
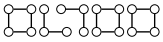
2.5 Online Delayed *H*-Node-/*H*-Edge-Deletion Problems with Advice

In this section, we want to focus on the question which amount of advice bits is necessary and sufficient to the solve both D$\mathcal{F}$ND and D$\mathcal{F}$ED problems when assuming that $\mathcal{F}$ consists of a single, disconnected graph H.

Rossmanith [63] answered this question for single, *connected* graphs H in the setting of node-deletions, giving an essentially tight lower bound of $opt \cdot$

$\log(\max_{H \in \mathcal{F}} |H|)$. His lower bound construction first presents a copy of an H and then glues another copy of the same H together at an isomorphic node. It is then easy to show that deleting this shared node is exactly the optimal solution. This gadget is then repeated an arbitrary number of times — with the shared node of the two copies of H chosen arbitrarily in each iteration — to ensure that constant advice is not sufficient. The advice complexity follows naturally: An algorithm has to distinguish every node of the largest graph H, and every completed gadget can be made H-free with a single deletion.

For disconnected graphs, the same strategy does not allow us to so cleanly separate iterations of a gadget and thus to calculate the advice complexity as a union of nicely separated gadgets.

As an example, consider $H = \{$  $\}$, i.e. a disconnected graph consisting of two square graphs. As soon as the first iteration of the lower bound is completed, the online graph G has the form  . Following the lower bound proof of Rossmanith, in the second iteration, another copy of H is then introduced, resulting in the graph  . Clearly, *two* squares now have to be deleted to ensure that the graph is H-free, but the remaining square of the *first* gadget now also has to be taken into consideration. The size of the optimal solution thus increased involuntarily after two gadgets, making it hard to count advice bits against the amount of deletions. This becomes even harder to handle when there are additional graphs in $\mathcal{F}$ that can only be found after a number of gadgets have been repeated, such as $\mathcal{F} = \{$  $,$ $\}$. While we only talked about the proof for node-deletion problems, Rossmanith gives a similar construction for edge-deletion problems, which fails for the same reasons on disconnected graphs.

This should be sufficient motivation to look a for a different approach in order to show the advice complexity of these problems. In the remainder of this section, we are concerned only with the case of edge-deletions. While the lower bound construction we use can be adapted for node-deletion-problems without much effort, we will later see that for these problems there is a much simpler proof to obtain essentially tight advice complexity bounds.

2.5.1 Lower Bound for H-Edge-Deletion Problems

While we have just seen that the construction by Rossmanith cannot be simply adapted for the case of disconnected DHED problems, we introduce a construction that is "dual" to that of Rossmanith; Where Rossmanith's construction ensures in every iteration that the remainder of the gadget is disconnected and each component smaller than the given H, we want to ensure that after each iteration, all gadgets from *all iterations* are connected in such a way that no disconnected H can be induced in the remaining fragment. This construction as well as the following proofs of this subsection can also be found in [11].

For this gadget construction, we use the *join* of graphs.

Definition 2.5.1. Given two graphs G_1, G_2. The *join graph* $G = G_1 \nabla G_2$ is constructed by connecting each vertex of G_1 with each vertex of G_2 with an edge, i.e. $V(G) = V(G_1) \cup V(G_2)$, $E(G) = E(G_1) \cup E(G_2) \cup \{ v_1v_2 \mid v_1 \in V(G_1), v_2 \in V(G_2) \}$.

It is insightful to understand why exactly the join operation behaves nicely with disconnected graphs. The most important observation is summarized in the following lemma.

Lemma 2.5.1. *Let H be a disconnected graph and let G_1, G_2 be two other graphs. Then $G_1 \nabla G_2$ is H-free iff G_1 and G_2 are H-free.*

Proof. Obviously, if $G_1 \nabla G_2$ is H-free the same must hold for all its induced subgraphs, in particular G_1 and G_2. For the converse, we assume that G_1 and G_2 are H-free but $W \subseteq V(G_1 \nabla G_2)$ induces an H in $G_1 \nabla G_2$. Since G_1 and G_2 are H-free, W cannot contain only vertices from one of the two graphs. Hence, there exists a $v_1 \in V(G_1) \cap W$ and a $v_2 \in V(G_2) \cap W$. These two vertices are connected by an edge in $G_1 \nabla G_2$. Then any other vertex $w \in W$ is either connected to v_1 or v_2 by an edge. Thus, $(G_1 \nabla G_2)[W]$ must be connected. However, H is a disconnected graph, therefore, W cannot induce an H in $G_1 \nabla G_2$. □

We introduce the notion of an e-extension of a graph H. Intuitively, an e-extension is a graph $U_H(e)$ that extends H in such a way that the unique optimal way to make $U_H(e)$ H-free is to delete the edge e.

Definition 2.5.2. For a disconnected graph H and an edge $e \in E(H)$ we call a graph $U_H(e)$ an *e-extension* of H if it satisfies

(E.1) $H \trianglelefteq U_H(e)$,

(E.2) $H \not\trianglelefteq_\varphi U_H(e) - e$, and

(E.3) $H \trianglelefteq_\varphi U_H(e) - f$ for all $f \in E(U_H(e)) \setminus \{e\}$.

We call H *edge-extendable* if for every $e \in E(H)$ there is such an e-extension.

It turns out that extendability is a sufficient condition for the desired lower bound to hold.

Theorem 2.5.1. *Let H be an edge-extendable graph. Then any optimal online algorithm solving the* Delayed H-Edge-Deletion *problem needs $opt \cdot \log ||H||$ bits of advice.*

Proof. Let $m \in \mathbf{N}$ be arbitrary. We construct a family of instances with optimal solution size m such that any optimal online algorithm needs advice to distinguish these instances. Take m disjoint copies $H^{(1)}, ..., H^{(m)}$ of H. We denote the vertices of $H^{(i)}$ by $v^{(i)}$. Furthermore, let $e_1, ..., e_m$ be arbitrary edges such that $e_i \in E(H^{(i)})$. We construct the instance $G(e_1, ..., e_m)$ in m phases. In the ith phase we reveal $H^{(i)}$ and join it with the already revealed graph from previous phases. Then we extend $H^{(i)}$ to $U_{H^{(i)}}(e_i)$ and again join the newly added vertices with the already revealed graph from the previous phases. If $G(e_1, ..., e_{i-1})$ is the graph after phase $i-1$, after phase i we have revealed a graph isomorphic to $G(e_1, ..., e_{i-1})\nabla U_{H^{(i)}}(e_i)$. Thus, $G := G(e_1, ..., e_m) \simeq U_{H^{(1)}}(e_1)\nabla ... \nabla U_{H^{(m)}}(e_m)$. We claim that $X := \{e_1, ..., e_m\}$ is the unique optimal solution for the H-Edge Deletion problem on G. Deleting all e_i from G yields a graph isomorphic to $(U_{H^{(1)}}(e_1) - e_1)\nabla ... \nabla (U_{H^{(m)}}(e_m) - e_m)$. By definition of an e-extension, and Lemma 2.5.1 this graph is H-free. Thus X is indeed a solution. It is furthermore optimal because G contains m edge-disjoint copies of H. Finally, if in one of the $U_{H^{(i)}}(e_i)$ we delete any other edge than e_i, by definition we need to delete at least one more edge to make $U_{H^{(i)}}(e_i)$ H-free. Hence, X is the unique optimal solution.

We can construct $||H||^m$ such instances that pairwise only differ in the choice of the edges $e_1, ..., e_m$. Any online algorithm needs advice to distinguish these instances, and therefore requires $m \cdot \log ||H||$ bits to be optimal on all of them. Since $m = opt$, the claim is proven. □

Next, we prove constructively that each disconnected graph H without isolated vertices is edge-extendable. Later we deal with the case that H has isolated vertices.

Lemma 2.5.2. *Let H be a disconnected graph without isolated vertices. Then H is edge-extendable.*

Proof. Let $xy = e \in E(H)$ be an arbitrary edge. We construct an e-extension $U_H(e)$ of H as follows. Let H' be a disjoint copy of H with vertex set $V(H') = \{ v' \mid v \in V(H) \}$. Now we identify x with x' and y with y' and join H with H' everywhere else. We call the resulting graph U. An example can be seen in Figure 2.1. We verify (E.1)-(E.3) for U. (E.1) is trivially fulfilled. Furthermore, U satisfies (E.3) since U consists of two copies of H that only share the edge e. Deleting any other edge f will leave one of the two copies unchanged, and thus $H \trianglelefteq_\varphi U - f$. Finally, we prove (E.2) by contradiction. Suppose that there is a set $W \subseteq V(U)$ that induces a graph isomorphic to H in $U^* := U - e$. If W contains neither x nor y, we have $H \trianglelefteq_\varphi U^* - \{x, y\} = (H - \{x, y\})\nabla(H' - \{x', y'\})$. However, the latter is H-free by Lemma 2.5.1. Thus, W must contain x or y. W.l.o.g. we assume that $x \in W$. Since H has no isolated vertices, W contains a neighbor of x in U^*. We have $N^{U^*}(x) = (N^H(x) \setminus \{y\}) \cup (N^{H'}(x') \setminus \{y'\})$, thus w.l.o.g.

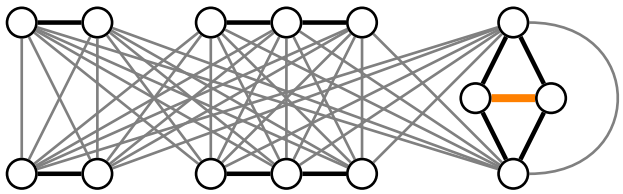


Figure 2.1: Example for the e-extension of the graph $P_2 \cup P_3 \cup K_3$ as constructed in Lemma 2.5.2. The edge e is depicted in orange.

we may assume that there is a $v \in W \cap N^H(x) \setminus \{y\}$. However, there must also exist a $w' \in W \cap (V(H') \setminus \{x', y'\})$: Otherwise $W \subseteq V(H)$, yet after the edge deletion $H - e$ is H-free. This implies $vw' \in E(U^*)$, and every vertex except for y is either connected to v or w'. However, if $y \in W$, also some neighbor of y must be in W, and therefore we conclude that W must induce a connected graph. This contradicts the assumption, and we have proven that U is indeed an e-extension of H. □

The results from Rossmanith together with Lemma 2.5.2 and Theorem 2.5.1 yield the following corollary.

Corollary 2.5.1. *Let H be a graph without isolated vertices. Then any optimal online algorithm solving the* Delayed H-Edge-Deletion *problem needs $opt \cdot \log ||H||$ bits of advice.*

We now turn finally to the case where H has isolated nodes. We prove via a simple advice-preserving reduction from the known case to the case where H has isolated vertices that the same lower bound holds.

Theorem 2.5.2. *Let H be a graph with $k > 0$ isolated vertices. Then any optimal online algorithm solving the* Delayed H-Edge-Deletion *problem needs at least $opt \cdot \log ||H||$ bits of advice.*

Proof. Let H' be the graph we obtain from H by deleting all isolated vertices. Of course, $||H|| = ||H'||$. For any graph G' we have: G' is H'-free iff $\overline{K_k} \cup G'$ is H-free. Let G' be an online instance for the Delayed H'-Edge-Deletion. We construct an online instance G for the Delayed H-Edge-Deletion that presents $\overline{K_k}$ in the first k time steps and then continues to present G' node by node. Note that $|G| = |G'| + k$. The deletions in G' can be translated in to deletions in G by shifting k time steps. The optimal solutions of G and G' coincide up to this shifting by k time steps, and of course $opt_H(G) = opt_{H'}(G') = opt$. Thus, the advice complexity for the Delayed H-Edge-Deletion is at least the advice complexity for the Delayed H'-Edge-Deletion. For this problem, however, we already have a lower bound from Corollary 2.5.1. Thus, the same lower bound applies to the case where H has isolated vertices. □

We will later see in Section 2.7.1 that a much simpler argument lets us obtain the same lower bound for the DHND for disconnected H.

2.6 Online Delayed $\mathcal{F}$-Edge-Deletion Problems with Advice

In Section 2.5 and in the work of Rossmanith one can see that determining the advice complexity for a single forbidden graph is manageable in the case of edge-deletion. The constructions for both disconnected and connected H are arguably quite simple, while proving that they are indeed correct is more involved. The situation gets much worse once we allow for an obstruction set to include more than a single graph, even if we demand them all to be connected.

Intuitively, the reason for this is quite simple. Assume that in the node-deletion case an algorithm identifies some induced forbidden graph H in the input only after a node has been revealed. Once it deletes any node of this H, it can then be certain that no further copy of *any* H is induced in the graph. This is the case since we assume that no pair of graphs in a family $\mathcal{F}$ induce one another, as discussed in Section 2.1.

Once we turn to edge-deletion problems, this certainty is gone. As a simple example, consider $\mathcal{F} = \{$$,$$\}$. Obviously, $\not\trianglelefteq_\varphi$, but deleting any single edge from the produces a . The consequence is that when designing both algorithms and lower bounds for D$\mathcal{F}$ED problems, we have to be very careful when counting deletions, as a single one may not always be sufficient to make a graph $\mathcal{F}$-free.

Most of the results of this section can also be found in [29]. Except for the final lower bound proof, they are valid for arbitrary families of graphs $\mathcal{F}$. We can however only proof a lower bound matching the upcoming upper bound for connected families $\mathcal{F}$, as we point out later. We start by defining some vocabulary that makes it easier to give our statements and proofs.

Definition 2.6.1. Let $\mathcal{F}$ be a fixed family of graphs and $H \in \mathcal{F}$. Let $S \subseteq 2^{E(H)}$.

1. A set $D \subseteq E(H)$ is *H-optimal* for a graph G if $H \trianglelefteq G$ and $G - D$ is $\mathcal{F}$-free and $opt = |D|$.

2. A set $D \subseteq E(H)$ is *H-good* for a graph G if $H \trianglelefteq G$ and D is a non-empty subset of some $\bar{D} \subseteq E(G)$ where $\bar{D}$ is H-optimal for G.

3. S is *H-sound* if $H - D$ is $\mathcal{F}$-free for every $D \in S$.

4. S is *H-sufficient* if for every graph G with $H \trianglelefteq G$ there is a $D \in S$ such that D is H-good for G.

5. S is *H-minimal* if for every $D \in S$, there is a graph G such that D is H-good for G, but every $D' \in S$, $D' \neq D$ is not.

Before we use any of these new terms, we try to give an intuition on some of them. The terms H-optimality and H-goodness for a graph G are the easiest to grasp: We will be interested in calculating and communicating an optimal solution, but this solution may not be necessarily unique. Thus, any solution set for an H that is of the same size as an optimal one is H-optimal. As in our algorithm we will sometimes be interested in communicating partial solutions that are of course part of a bigger, complete optimal solution, we sometimes use the term H-good.

Lemma 2.6.1. *Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be a family of graphs, G a graph and $D \subseteq E(H_i)$ that is H_i-good for G. Then there is a subgraph $G' \subseteq G$ such that D is H_i-optimal for G'.*

Proof. As D is H_i-good for G there must be some $\bar{D} \supseteq D$ that is optimal for G by the definition of goodness. Let us construct the graph $G' = G - (\bar{D} - D)$, i.e., we get G' from G by removing all edges that are in $\bar{D}$, but not in D. Let us assume that D is not optimal for G'. Then there must be an optimal D' for G' with $|D'| < |D|$. Then, however, $G - ((\bar{D} - D) \cup D') = G' - D'$ is also $\mathcal{F}$-free by construction, which is impossible because $(\bar{D} - D) \cup D'$ is smaller than the already optimal $\bar{D}$. Hence, D is optimal for G. □

H-soundness and H-sufficiency as well as H-minimality are all properties of a concrete family of edges S that is a subset of the powerset of the edge set $E(H)$. Such a set S is the very core of both the upper and lower bound proofs of this section. For the upper bound, an algorithm will be tasked with deleting edges of a copy of some H that it can identify in the input graph. It can then calculate a set S with the H-soundness, H-sufficiency and H-minimality properties on the fly for the concrete H. It then reads advice which concrete edge subset from S it should delete. The H-soundness then ensures that due to the deletions, no other $H' \in \mathcal{F}$ are induced by the remainder of H in the graph, while the H-sufficiency ensures that any edge set that could be part of an optimal solution is available for selection to the algorithm. H-minimality is then to ensure that no redundant edge sets are part of S. If e.g. for $H = \{$  $\}$ all of the three edges are separately put into S, then minimality demands that no superset consisting of these three sets is also in S.

Let us look at an example for a concrete set $\mathcal{F}$ and see how one can construct these sets S by hand. The obstruction set we use is that for *trivially perfect* graphs. We will come back to this example later and determine the amount of advice necessary to solve the D$\mathcal{F}$ED problem optimally on this $\mathcal{F}$, using these sets.

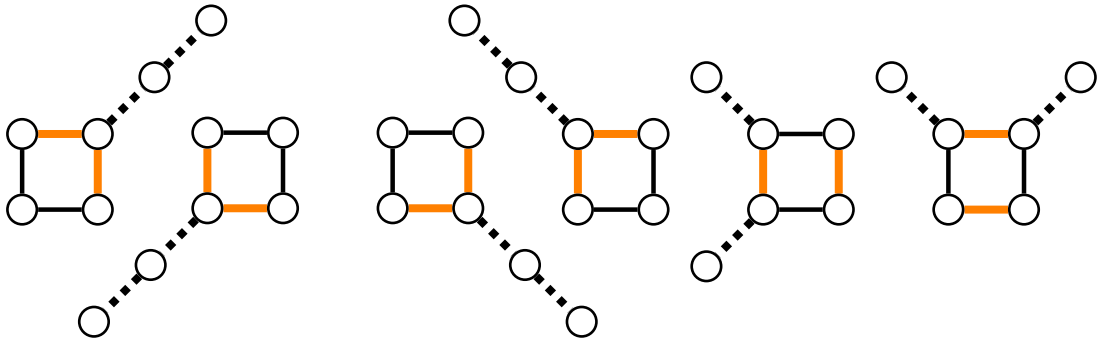


Figure 2.2: In Example 2.6.1 the set S_1 contains six edge sets shown here as orange lines. The corresponding supergraphs of H_1 are depicted demonstrating that all six sets in S_1 are necessary.

Example 2.6.1. Given $\mathcal{F} = \{$  ,  $\}$. We compute sets $S_i \subseteq 2^{E(H_i)}$ that are H_i-sound and H_i-sufficient and H_i-minimal for each $H_i \in \mathcal{F}$. We start with $H_1 =$  .

A single edge cannot be in S_1, as this would violate the demanded H_1-soundness by leaving a  in the graph. S_1 must not only consist of edge subsets of size three or larger, as this would violate the H_1-sufficiency for the graph  if the input graph was H_1 itself. Thus, S_1 contains at least one two-element subset of $E(H_1)$. Indeed, every subset including exactly two edges is part of S_1, as the following construction shows, which is also visualized in Figure 2.2. By constructing an input graph that extends one of the nodes of the  to a  , each pair of neighboring edges is exactly the optimal solution. On the other hand, if we attach an edge to each of two neighboring nodes of the  , we need to delete exactly two opposing edges of . As we cover each two-element edge subset of , we do not need to look at edge subsets bigger than size two, if we also want S_1 to be H_1-minimal.

We continue by computing S_2 for $H_2 =$  , which is a bit simpler. S_2 must not only consist of edge subsets of size two or larger, as this would violate the H_2-sufficiency for the graph  . On the other hand, every one-edge subset of $E(H_2)$ is part of S_2. The outer edges of the path are optimal for deletion if we attach a P_2 to either end of the original  . The middle edge is optimal for deletion if we attach an edge to both ends of the path. Again, as we cover every single-edge subset, we do not need to look at bigger subsets for S_2.

Thus, $S_1 =$ {  ,  ,  ,  ,  ,  ,  } and $S_2 =$ {  ,  ,  } are each H_i-sound, H_i-sufficient and H_i-minimal regarding their respective H_i.

We do not have to determine such edge sets by hand, of course. One can simply enumerate all possible elements of $2^{E(H)}$ for an $H \in \mathcal{F}$ and check whether such a subset of edges is H-sound and H-sufficient. The check for H-soundness is clearly decidable, but it may not be clear whether the check for H-sufficiency is decidable or only semidecidable.

To show that H-sufficiency is indeed decidable, we give the following lemma.

Lemma 2.6.2. *Let $\mathcal{F}$ be a family of forbidden graphs, $H \in \mathcal{F}$, and $S \subseteq 2^{E(H)}$. There is an algorithm that can decide whether S is H-sufficient.*

Proof. It is sufficient to verify for all connected graphs G with $H \trianglelefteq G$ that some $D \in S$ is H-good for G, i.e., there is an optimal solution for G that contains D. By Lemma 2.6.1 we can restrict our search to all such G's that have an optimal solution that is a subset of $E(H)$. There are infinitely many graphs G to check. To overcome this we define the *unfolding* of G, written $\Upsilon(G)$, as the set of the following graphs: Remember that $H \trianglelefteq G$. If there is some $H' \in \mathcal{F}$ with $H' \trianglelefteq_\varphi G$ then $G[E(H) \cup E(\phi(H'))] \in \Upsilon(G)$ (for every possible ϕ). If, however, $\Upsilon(G)$ contains two graphs G' and G'' that are isomorphic via an isomorphism that is the identity on $V(H)$, then only the lexicographically smaller one is retained.

This means that the unfolding of G contains all induced subgraphs that consist of H and one other copy of some forbidden induced subgraph from $\mathcal{F}$ that must overlap with H in some way (because we assumed that G has an optimal solution that consists solely of edges from H). Here is a small example: Let $\mathcal{F} = \{$  ,  $\}$, $H = \{$  $\}$, $G =$. Then $\Upsilon(G) = \{$,  ,  ,  $\}$.

It is easy to see that deleting some $D \subseteq E(H)$ from G makes it $\mathcal{F}$-free iff deleting the same D from all graphs $G' \in \Upsilon(G)$ makes all of them $\mathcal{F}$-free. Hence, there is an optimal solution for G that is a subset of $E(H)$ iff there is such a subset that is "optimal" for $\Upsilon(G)$ (i.e., the deletion of no smaller edge set can make all graphs in $\Upsilon(G)$ $\mathcal{F}$-free).

There are only finitely many possibilities for $\Upsilon(G)$, i.e. possibilities to glue H with another $H' \in \mathcal{F}$, and we can enumerate all of them. Let us say this enumeration is $\Upsilon_1, \ldots, \Upsilon_t$. For each Υ_i we first find out whether there is a G with $\Upsilon(G) = \Upsilon_i$. We can do this by enumerating all graphs G up to a size that does not exceed the sum of the sizes of all graphs in Υ_i and computing $\Upsilon(G)$ for them. If indeed $\Upsilon(G) = \Upsilon_i$ then we test whether some $D \in S$ is H-good for G. Iff these tests pass for all i then S is indeed H-sufficient. □

We show a way to modify a given set that is already H-sound and H-sufficient for some H, such that it also become H-minimal, later. H-minimality is only necessary for the lower bound proof, as the upper bound algorithm only requires the first two characteristics in order to be correct and optimal.

An important tool that we use in the analysis of the number of advice bits is the solution of a special recurrence relation: Let $(d_1, \ldots, d_k) \in \mathbf{N}^k$. Let $m(n)$ be the solution to the recurrence relation with

$$m(n) = \begin{cases} \sum_{i=1}^{k} m(n - d_i) & \text{if } n \geq \max\{d_1, \ldots, d_k\} \\ c_n & \text{otherwise} \end{cases} \tag{2.1}$$

where $c_n \geq 0$ and some $c_i > 0$ for $0 \leq i < \max\{d_1, \ldots, d_k\}$. Let $\beta(d_1, \ldots, d_k) = \inf_\tau\{\, \tau \mid m(n) = O(\tau^n) \,\}$. Note that β does not depend on the c_i's and that $m(n)$ does depend on the d_i's.

If $S = \{D_1, \ldots, D_k\}$ is a family of sets, we define $\beta(S) = \beta(|D_1|, \ldots, |D_k|)$.

A homogeneous linear recurrence relation with constant coefficients usually has a solution of the form $\Theta(n^{k-1}\tau^n)$ if τ is the dominant root of the characteristic polynomial with multiplicity k [39]. However, in (2.1) the coefficients of the characteristic polynomial are real numbers and there is exactly one sign change. By Descartes' rule of signs there is exactly one positive real root and therefore its multiplicity has to be one [31, 45]. Therefore $m(n) = \Theta(\beta(S)^n)$.

This function β occurs in the advice complexity bounds of both the upper and lower bound of this section, as it enables us to capture the varying number of edges that have to be deleted in every step: While in the node-deletion setting, it was always possible to assume that every $H \in \mathcal{F}$ that was presented could be removed such that the graph was $\mathcal{F}$-free again afterwards with exactly one deletion, this is not the case here. Example 2.6.1 shows us that for each $H \in \mathcal{F}$ a different number of edges may have to be deleted in order to ensure that a graph is $\mathcal{F}$-free afterwards. This recursion relation is thus used to help us determine (up to a constant error term) how many different unique solutions we have to distinguish over all possible concatenations of different $H \in \mathcal{F}$ in a worst case scenario. This is essentially what gives us an advice bound in the upcoming upper bound analysis.

This machinery does however make it tricky to then get an intuitive grasp on the actual amount of advice that is necessary — and as we will later see, sufficient — to solve an D$\mathcal{F}$ED problem. We thus continue Example 2.6.1 at the end of this section and calculate the amount of advice needed for optimally solving the corresponding edge-deletion problem.

2.6.1 Upper Bound

We first give an upper bound algorithm, following the intuition given in the previous paragraphs.

Theorem 2.6.1. *Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be a family of graphs and let S_i be H_i-sound and H_i-sufficient for all $i \in \{1, \ldots, k\}$. Then there is an $m \in \mathbf{R}$ and an online algorithm that solves the* Delayed $\mathcal{F}$-Edge-Deletion *problem for every graph G with $m \cdot opt + O(1)$ many advice bits where $2^m \leq \max_{i\in\{1,\ldots,k\}} \beta(S_i)$.*

Proof. The algorithm receives $opt \cdot \log(\max_i\{\beta(S_i)\}) + O(1)$ many advice bits and then a graph G as a sequence of growing induced subgraphs $G_1, \ldots G_n$ with $G_i := G[\{v_1, \ldots, v_n\}]$ and $G_n = G$. Let C be the solution set of the algorithm, initialized with $C := \emptyset$. The algorithm deletes in total exactly *opt* edges. We analyze the total number of different advice strings the algorithm might use when

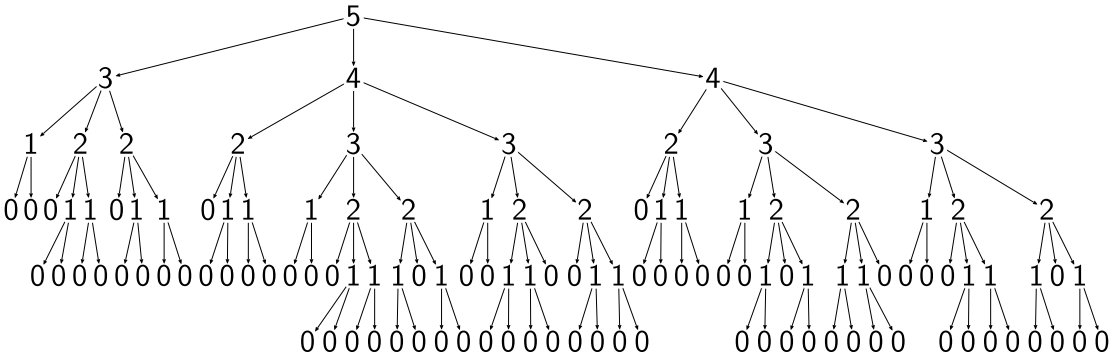


Figure 2.3: Possible ways to delete H_1's with $S_1 = \{D_1, D_2, D_3\}$ and $|D_1| = 2$, $|D_2| = |D_3| = 1$, assuming the optimal number of deletions is 5.

deleting *opt* edges. The algorithm first calculates for each $H_i \in \mathcal{F}$ a set S_i that is H_i-sound and H_i-sufficient.

When the next node v_j is revealed, the algorithm checks if the graph $G_j - C$ is $\mathcal{F}$-free. If this is not the case, it deterministically chooses some $H_i \in \mathcal{F}$ for which $H_i \trianglelefteq_\varphi G_j$. It then reads from the advice tape which $D \in S_i$ it should select to delete edges from H_i, which was chosen to be $\varphi(H_i)$-good by the advisor. It then deletes the edges corresponding to the set D and sets $C := C \cup \varphi(D)$. If $G_j - C$ is still not $\mathcal{F}$-free, it repeats this deletion step until it is.

The advice string is therefore partitioned into $|S_i|$ subsets, one for each $D \in S_i$. As it is possible that $|D| > 1$, we now have to carefully count the advice relative to *opt*, i.e. how often the algorithm needs to ask for advice dependent on the number of edges it deletes in an iteration. After deleting $\varphi(D)$, the remaining number of necessary deletions — initially *opt* — is now by $|D|$ smaller. If $m(opt)$ is the total number of advice strings we get the recurrence $m(opt) = \max_i\big(\sum_{D \in S_i} m(opt - |D|)\big)$ if *opt* is at least as big as every $D \in S_i$. Standard techniques show that $m(opt) = O(\max\{\beta(S_1), \ldots, \beta(S_k)\}^{opt})$. □

In Figure 2.3 you can find the behavior of an algorithm that solves the D$\mathcal{F}$ED with $S_1 = \{D_1, D_2, D_3\}$ and $|D_1| = 2$, $|D_2| = |D_3| = 1$ when it encounters only a forbidden $H_1 \in \mathcal{F}$ as a tree of possible different computation paths. In the tree nodes you find the number of edges that will still be deleted. This corresponds to the recurrence $m(n) = m(n-2) + m(n-1) + m(n-1)$ for $n \geq 2$ and $m(0) = 1$, $m(1) = 2$. Then $m(5) = 70$ and $m(n) = \Theta((\sqrt{2}+1)^n)$ because $\beta(2,1,1) = \sqrt{2}+1$. The exact solution is $m(n) = \frac{1}{4}(\sqrt{2}+1)^n(\sqrt{2}+2) - \frac{1}{4}(1-\sqrt{2})^n(\sqrt{2}-2)$.

2.6.2 Lower Bound

Giving a lower bound that allows us to measure the advice using sets of edges as in the upper bound is more involved. There is a reason why we introduced the vocabulary of *soundess*, *sufficiency* and *minimality*: We show that any optimal online algorithm with advice needs to be able to distinguish the edge sets of any $S \subseteq 2^{E(H)}$ with these three characteristics for any $H \in \mathcal{F}$, assuming $\mathcal{F}$ is connected.

In order to start this analysis, we first introduce a technical lemma. It states that we can find a matching with special properties in every connected bipartite graph. This will later help us to find the "best set" S an algorithm could possibly choose, regarding the necessary amount of advice to optimally solve the problem. This will especially give us a way to modify a given set S that is already H-sound and H-sufficient for some H such that it becomes H-minimal.

Lemma 2.6.3. *Let $G = (U, V, E)$ be a bipartite graph where $U = \{u_1, \ldots, u_k\}$. Let $\leq$ be a reflexive and transitive relation on U such that $u_1 \leq \cdots \leq u_k$. Moreover, assume that $V \subseteq N(U)$, i.e., every node in V is connected to some node in U. Then there exist a $U' \subseteq U$ and $V' \subseteq V$ such that*

1. *$N(U') = V$,*
2. *$G[U' \cup V']$ is a matching,*
3. *$\min N(v) \in U'$ for every $v \in V'$.*

Proof. We claim that Algorithm 1 computes sets U' and V' that fulfill the three properties stated in the lemma. As $G[U' \cup V']$ is an induced matching in G, the matching can be found easily from U' and V'.

Algorithm 1 Algorithm for Finding a Matching from Lemma 2.6.3

```
1: U' ← { min N(v) | v ∈ V }
2: V' ← ∅
3: for u ∈ U' in descending order do
4:     if there is a v ∈ V such that N(v) ∩ U' = {u} then
5:         V' ← V' ∪ {v}                    ▷ pick any such v
6:     else
7:         U' ← U' \ {u}
```

We prove all three properties separately.

1. "$N(U') = V$": The precondition of the lemma states that $N(U) = V$ and therefore that $N(v) \neq \emptyset$ for every $v \in V$. In line 1 we add to U' a neighbor of each $v \in V$, which already guarantees $N(U') = V$.

We have to prove that the invariant $N(U') = V$ is maintained in the **for**-loop. Only in line 7 a node is removed from U', which could destroy the property $N(U') = V$. However, we remove u only if there is no $v \in V$ for which u is the only neighbor. Hence, every v retains at least one other neighbor in U'.

2. "$G[U' \cup V']$ is a matching": If in the end $u \in U'$, then u was *not* removed from U' in line 7, which means that line 5 was executed and a v added to V' with $N(v) \cap U' = \{u\}$. At this point of time v has only one neighbor in U'. Afterwards U' is only shrinking, so v cannot have more than one neighbor in U' at the end. Also, afterwards u cannot be deleted from U' as each u is only considered once in the body of the **for**-loop. This means that at the end each $v \in V'$ has exactly one neighbor in U'. There cannot be a $u \in U'$ with no neighbor in V' because it would have been removed in line 7. Hence, $U' \cup V'$ induces a matching.

3. "$\min N(v) \in U'$ for every $v \in V'$": Directly after line 2 the statement is obviously true.

   Let us assume that this property does not hold at any later point. Then there is a $v \in V'$ and $\min N(v) = u$, where $u \notin U'$. For this to hold, there must be a different $u' \in U', u' \in N(v)$ that is matched to v because $U' \cup V'$ induces a matching. After line 1 was executed, U' thus contained u and u'. Because $u \leq u'$ and in line 3 the vertices in U' are visited in descending order, u' was visited *before* u.

   We are now looking at the moment when u' was visited in line 3. Then U' still contained both $u, u' \in U'$ and $u, u' \in N(v)$. u' cannot be matched with v at this point, as $|N(v) \cap U'| > 1$. This means that after u' was considered in the **if**-condition in line 5, either u' was removed from U' or u' was matched with a node from $V \setminus v$, both leading to a contradiction.

□

As this result is rather abstract in itself, let us look at an example in which we determine a matching from a given bipartite graph.

Example 2.6.2. We are given the bipartite graph depicted in Figure 2.4. Let the seven numerically labeled nodes at the top be the set U and the alphabetically labeled nodes on the bottom be V. We assume that the nodes of U are ordered in an ascending manner, based on some reflexive and transitive relation, as shown in the figure. We claim that the highlighted nodes of U, i.e. the nodes $\{1, 3, 4, 6\}$ together with the highlighted nodes of V, i.e. the nodes $\{b, d, i, m\}$, form sets $U' \subseteq U$ and $V' \subseteq V$, that have the properties stated in Lemma 2.6.3.

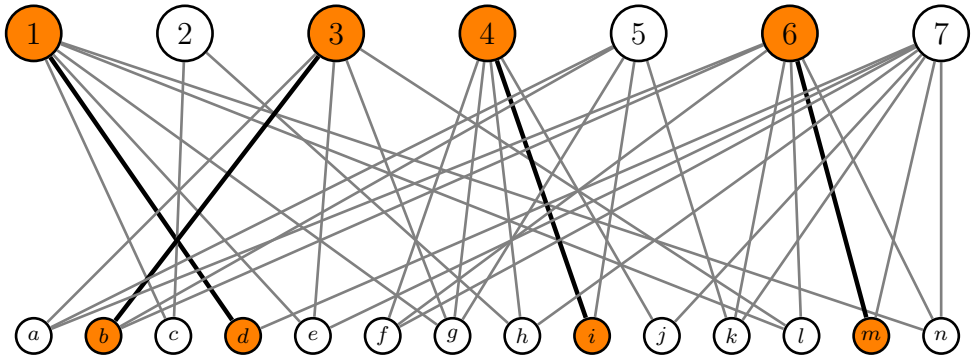


Figure 2.4: An example for the algorithm in the proof of Lemma 2.6.3. The sets U' (top, ordered from left to right), V' (bottom) and the matching are highlighted.

1. "$N(U') = V$": It is easy to check that $N(\{1, 3, 4, 6\}) = V$ by checking that there is an edge from these four nodes to every bottom node of the graph.

2. "$G[U' \cup V']$ is a matching": A way to check that this holds is to confirm that each node of U' is connected to exactly one node of V' and vice versa.

3. "$\min N(v) \in U'$ for every $v \in V'$": If $v \in V'$ then $N(v)$ contains several vertices from U, but exactly one node in U', i.e., its partner in the matching. We require that this partner is the *smallest* one in $N(v)$. You can check the third property in the figure easily: Just verify that the matching edge is the leftmost emerging edge of each $v \in V'$. For example, m has two neighbors 6 and 7. The smaller one is its partner in the matching and hence in U'.

We now use this rather abstract lemma to show how we can determine sets of edges S that are H-sound, H-sufficient and H-minimal for some $H \in \mathcal{F}$ with an additional property: As a set with these properties may not be unique, we are interested in a set S with the cardinally smallest sets D. Subsequently, we show that these sets are exactly those that any optimal algorithm will need to be able to distinguish, giving us a lower bound.

Lemma 2.6.4. *Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be a family of connected graphs and S_i be H_i-sound and H_i-sufficient for all $i \in \{1, \ldots, k\}$. Then there are $S'_i \subseteq S_i$ such that S'_i is H_i-sound, H_i-sufficient and H_i-minimal and moreover:*

For every $D' \in S'_i$ there is a graph G with $H_i \trianglelefteq G$ such that D' is H_i-good for G and for every $D \in S_i \setminus S'_i$ that is also H_i-good for G, it holds that $|D| \geq |D'|$.

Proof. We use Lemma 2.6.3. In the following we look at a fixed i and write H for H_i, S for S_i, and S' for S'_i.

If $H \trianglelefteq G_1$ and $H \trianglelefteq G_2$, then we define that $G_1 \equiv_S G_2$ iff for all $D \in S$ it holds that D is H-good for G_1 if and only if D is H-good for G_2. There are only

$2^{|S|}$ many possibilities which $D \in S$ are H-good for some G and consequently the equivalence relation $\equiv_S$ has at most $2^{|S|}$ many equivalence classes. Assume that $R = \{G_1, \ldots, G_m\}$ is a set of representatives of all equivalence classes. We build a bipartite graph (S, R, E) where there is an edge between $D \in S$ and $G_i \in R$ iff D is H-good for G_i. Furthermore we define a reflexive and transitive relation on S by defining $D \leq D'$ iff $|D| \leq |D'|$. By Lemma 2.6.3 there is a matching between S and R that fulfills all three conditions that are stated there. In particular, we can determine the set $S' \subseteq S$ that corresponds to U' in the lemma.

1. S' is H-sound because it is a subset of S, which is H-sound itself.

2. S' is H-sufficient because by Lemma 2.6.3 we know that $N(S') = R$ in the bipartite graph. That means that there is an edge from every G_i to some $D \in S'$, which means there is some $D \in S'$ such that D is H-good for G_i.

3. S' is H-minimal, because for every $D \in S'$, its matched graph is covered exactly by D, as $G[U' \cup V']$ is an induced matching.

4. Lemma 2.6.3 states that $G[S' \cup R]$ is a matching. By the H-minimality of S', for every $D' \in S'$ there is a graph G such that only D' is H-good for G and no other member of S'. This is exactly the graph $G_i \in R$ matched with D', as $G[S' \cup R]$ is an induced matching. By condition 3 of Lemma 2.6.3, the cardinally smallest neighbor of G_i is D', thus there cannot be a $D \in S$ that is smaller than D'.

Thus, S' fulfills the conditions stated in Lemma 2.6.3. □

We now have all tools ready to give the lower bound.

Theorem 2.6.2. *Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be a family of connected graphs and assume there is an algorithm A that can solve the* Delayed $\mathcal{F}$-Edge-Deletion *problem for all inputs G with at most $m \cdot opt + O(1)$ advice for some $m \in \mathbf{R}$. Then there exist S_i' that are H_i-sound, H_i-sufficient, and H_i-minimal and $\beta(S_i') \leq 2^m$ for every $i \in \{1, \ldots, k\}$.*

Proof. By Lemma 2.6.4 there is an $S_i' = \{D_1, \ldots, D_r\} \subseteq S_i$ that is H_i-sound, H_i-sufficient, and H_i-minimal. It additionally has the property that for every $D' \in S_i'$ there is a graph G with $H_i \trianglelefteq G$ such that D' is H_i-good for G and for every $D \in S_i \setminus S_i'$ that is also H_i-good for G, it holds that $|D| \geq |D'|$.

Let $l \in \mathbf{N}$. The adversary prepares $\Theta(\beta(S')^l)$ many instances by repeating the following two-step procedure in several rounds until the size of the optimum solution for the presented graph exceeds $l - \max\{|D_1|, \ldots, |D_r|\}$.

1. The adversary presents a disjoint copy of H_i. Since H_i is connected, H_i cannot induce another $H_j \in \mathcal{F}$ with $H_j \neq H_i$.

2. The adversary selects a $D_j \in S'_i$. It then computes a graph G_j of H_i with $H_i \trianglelefteq G_j$ that is stated to exist by the H_i-minimality of S'_i. I.e. D_j is H_i-good for G_j, but all other members of S'_i are not. Of course, since D_j is H_i-good for G_j, the set D_j can be extended into a set $\bar{D}_j \supseteq D_j$ that $\bar{D}_j$ is H_i-*optimal* for G_j. Let $D'_j = \bar{D}_j - D_j$ be the set of these additional edges and $G'_j = G_j - D'_j$ be the graph G_j without these additional edges. Clearly, D_j is now H_i-optimal for G'_j.

   Next, the adversary transforms the chosen H_i into one of the r possible G'_js, making an arbitrary $D_j \in S'_i$ the unique optimal solution and thus increasing the size of the optimal solution by $|D_j|$.

To guarantee the uniqueness of this solution, we show that no other set of edges of S'_i can be H_i-good for this modified graph G'_j, and thus that D_j is the unique optimal choice of edges to delete for G'_j. For a contradiction, assume that this assumption is false, i.e that there is a $D_k \in S'_i$ that is H_i-good for G'_j and $D_k \neq D_j$. If D_k is H_i-good for G'_j, we are able to extend it to a set $\bar{D}_k \supseteq D_k$ that is even H_i-optimal, just as we did with D_j before. However, $G_j - D_k - ((\bar{D}_k - D_k) \cup D'_j)$ is $\mathcal{F}$-free, implying that D_k is *also* H_i-good for G_j, which is a contradiction, as the adversary chose G_j to be the graph where *only* D_j is H_i-good for G_j.

As soon as the size of the optimal solution exceeds $l - \max\{|D_1|, \ldots, |D_r|\}$, the adversary "pads" the rest of the instance with disjoint copies of H_i, without turning them into bigger connected graphs until the size of the optimal solution reaches exactly l. The number $N(l)$ of different instances is then given by the following recurrence:

$$N(l) = \begin{cases} \sum_{j=1}^{r} N(l - |D_j|) & \text{if } l \geq \max\{|D_1|, \ldots, |D_r|\} \\ 1 & \text{otherwise} \end{cases}$$

It is easy to see that $N(l) = \Theta(\beta(S'_i)^l)$. The algorithm has to react differently on all of these instances: When the algorithm sees a new H_i to be turned into one of $G'_1, \ldots, G'_r$, it deletes different edge sets for each of the r possibilities.

The adversary constructed an instance that consists of a sequence of disjoint graphs $G'_{i_1}, \ldots, G'_{i_t}$ from the set $\{G'_1, \ldots, G'_r\}$ of which the total size is at least $\sum_{j=1}^{t} opt_{\mathcal{F}}(G_{i_j}) - \max\{|D_1|, \ldots, |D_r|\}$ and $O(1)$ many copies of H_i. If G is the whole constructed instance we have $opt = l + O(1)$ because $opt_F(H_i)_{\mathcal{F}} = O(1)$, which is the size of the "padding". Together with $N(l) = \Theta(\beta(S'_i)^l)$ this means that Algorithm A uses at least $\log N(l) = l \cdot \log \beta(S'_i) + O(1) = opt \cdot \log \beta(S'_i) + O(1)$ advice bits. The equality between the last two terms holds, as we can hide the

difference between the two terms, which is of size $(opt - l)\beta(S'_i)$ — and thus constant —, in the $O(1)$ term.

Assume Algorithm A uses at most $m \cdot opt + O(1)$ advice bits on every graph G as stated in the precondition above. Then m cannot be smaller than $\log \beta(S'_i)$ for every $i \in \{1, \ldots, k\}$ because opt can be arbitrarily big. □

Putting it all together, we obtain the wanted bound, which is matching up to a constant derivation.

Theorem 2.6.3. *Let $\mathcal{F} = \{H_1, \ldots, H_k\}$ be connected graphs. The advice complexity for the* Delayed $\mathcal{F}$-Edge-Deletion *problem is then $m \cdot opt + O(1)$ where $m = \max_{i \in \{1,\ldots,k\}} \min\{ \log \beta(S) \mid S \subseteq 2^{E(H_i)}$, S is H_i-sound and H_i-sufficient$\}$. There is an algorithm that can compute m from $\mathcal{F}$. More specifically, there is an algorithm that gets $\mathcal{F}$ and $o \in \mathbf{N}$ as the input and returns the oth bit of the binary representation of m.*

Proof. "$\leq$" by Theorem 2.6.1. "$\geq$" by Theorem 2.6.2. An algorithm can enumerate all possible $S \subseteq E(H)$ and then test if S is H_i-sound and H_i-sufficient (by Lemma 2.6.2). Then $\beta(S)$ is computed by finding the only real root of the characteristic polynomial of the corresponding recurrence relations [39]. □

To conclude this section, we now determine the advice complexity for our example of trivially perfect graphs. In Example 2.6.1 we have seen that S_1 consists of six sets of size two and that S_2 consists of three sets of size one. We thus obtain $\beta(2, 2, 2, 2, 2, 2) < 2.5$ and $\beta(1, 1, 1) = 3$. Thus, by Theorem 2.6.3, the advice complexity is $3 \cdot opt + O(1)$.

2.7 Online Delayed $\mathcal{F}$-Node-Deletion Problems with Advice

In the previous sections we have seen that for node-deletion problems, the advice complexity for different restricted $\mathcal{F}$ could always be proven to be in $\Theta(opt)$, independent of the concrete family. An interesting result that we present in this section is that *not all* D$\mathcal{F}$ND problems obey this bound. Indeed, we see that, depending on the forbidden family of graphs $\mathcal{F}$, different amounts of advice are necessary and sufficient. It should not be very surprising that there exist some families for which $\Theta(opt \log |H|)$ is still a tight bound on the advice complexity. But curiously, for some families, advice in $\Theta(\log opt)$ and for some even *no advice* is needed. While we are unable to fully partition the advice complexity space of all D$\mathcal{F}$ND problems, we strongly suspect that for the vast majority of families, indeed $\Theta(opt \log |H|)$ is necessary and sufficient, as we discuss at the end of this

chapter. We start with some rather simple advice complexity bounds. Most of the results of this section can also be found in [11].

We first give a simple upper bound on all node deletion problems, that simply tells an algorithm exactly which nodes to delete. This is a trivial generalization of the upper bound used for connected D$\mathcal{F}$ND by Rossmanith [63].

Theorem 2.7.1. *Let $\mathcal{F}$ be an arbitrary family of graphs, and let $H \in \mathcal{F}$ be a maximum order graph. Then there exists an optimal online algorithm for the* Delayed $\mathcal{F}$-Node-Deletion *problem with advice that reads at most $opt \cdot \log|H| + O(\log opt)$ bits of advice.*

Proof. The online algorithm reads *opt* from the tape using self-delimiting encoding and $O(\log opt)$ advice bits (see [16]). Then it reads $\lceil opt \cdot \log|H| \rceil$ bits and interprets them as *opt* numbers $d_1, ..., d_{opt} \leq |H|$. Whenever some forbidden induced graph is detected, the algorithm deletes the d_ith node. □

2.7.1 Families of Disconnected Graphs

It is easy to show that the advice complexity result for connected D$\mathcal{F}$ND problems by Rossmanith is easily translated for *disconnected* D$\mathcal{F}$ND problems.

Lemma 2.7.1. *Let $\mathcal{F}$ be an arbitrary family of graphs, and let $\overline{\mathcal{F}} := \left\{ \overline{H} \mid H \in \mathcal{F} \right\}$ be the family of complement graphs. Then the advice complexity of the* Delayed $\mathcal{F}$-Node-Deletion *problem is the same as for the* Delayed $\overline{\mathcal{F}}$-Node-Deletion *problem.*

Proof. We prove the statement by giving an advice preserving reduction from Delayed $\mathcal{F}$-Node-Deletion to Delayed $\overline{\mathcal{F}}$-Node-Deletion: If G is an online instance for the $\mathcal{F}$-problem, such that in the ith time step $G[\{v_1, ..., v_i\}]$ is revealed, then we construct $\overline{G}$ as an online instance for the $\overline{\mathcal{F}}$-problem by revealing $\overline{G[\{v_1, ..., v_i\}]}$ in the ith time step. An optimal online algorithm has to delete the same nodes in the same time steps in both instances. This proves that the advice complexity for the $\mathcal{F}$-problem is at most the advice complexity for the $\overline{\mathcal{F}}$-problem. The same reduction in the other direction yields equality. □

From this follows immediately the desired lower bound on the advice complexity of the D$\mathcal{F}$ND for disconnected $\mathcal{F}$.

Theorem 2.7.2. *Let $\mathcal{F}$ be a family of disconnected graphs, and $H \in \mathcal{F}$ a maximum order graph. Then any optimal online algorithm for the* Delayed $\mathcal{F}$-Node-Deletion *problem needs $opt \cdot \log|H|$ bits of advice.*

Proof. Since all graphs in $\mathcal{F}$ are disconnected, $\overline{\mathcal{F}}$ is a family of connected graphs. For $\overline{\mathcal{F}}$, the lower bound proven by Rossmanith of $opt \cdot \log|H|$ holds. The claim follows from Theorem 2.7.1. □

In particular, these results imply a tight lower bound on the advice complexity of the DHND for disconnected graphs H.

Corollary 2.7.1. *Let H be a disconnected graph. Then any optimal online algorithm for the* Delayed H-Node-Deletion *problem needs $opt \cdot \log |H|$ advice bits.*

2.7.2 Families Requiring no Advice for Optimality

While maybe sounding curious at first, there are very simple families of arbitrary size such that the corresponding node-deletion problems require no advice in order to be optimally solved. While the proof itself is not very interesting, we give it anyway for the sake of drawing a bigger picture of the landscape of advice complexity.

Theorem 2.7.3. *Let $\mathcal{F}_k$ be a family of all graphs of size k for some fixed $k \in \mathbf{N}$. Then there exists an online algorithm solving the* Delayed $\mathcal{F}$-Node-Deletion *problem optimally.*

Proof. Any optimal solution for any concrete instance has to delete all but an arbitrary set of $k-1$ nodes of the instance in order to be correct. An online algorithm optimally solving this problem thus waits for $k-1$ nodes to be revealed. It then deletes any further nodes of the instance. □

2.7.3 Families Requiring Logarithmic Advice

Perhaps the most interesting result in this section is that advice logarithmic in *opt* is both necessary and sufficient for some specific families $\mathcal{F}$. Specifically, if the graph of an instance cannot be arbitrarily padded with nodes in any way without eventually inducing a graph in $\mathcal{F}$, the advice complexity for this deletion problem is bounded. An example for such a family is one that includes both an independent set and a clique.

Lemma 2.7.2. *Let $\mathcal{F}$ be an arbitrary family of graphs. Then there exists a minimal $R \in \mathbf{N}$ such that all graphs of size at least R are not $\mathcal{F}$-free iff $K_n, \overline{K_m} \in \mathcal{F}$ for some $n, m \in \mathbf{N}$.*

Proof. Ramsey's Theorem guarantees the existence of R if $K_n, \overline{K_m} \in \mathcal{F}$. Conversely, if $\mathcal{F}$ contains no clique (independent set), then arbitrarily big cliques (independent sets) are $\mathcal{F}$-free. □

We can use this observation to construct an algorithm that is mainly concerned with the remaining graph after all deletions have been made. As the size of this

graph is bounded by a constant, we can simply tell an online algorithm when it should *not* delete every node of an H that it sees. Thus, if we assume the largest possible remaining graph is of some constant size $k \in \mathbf{N}$, the algorithm can simply read the length of *opt* from the advice tape and then a list of at most k *deletion number* and *node label* pairs that tell the algorithm in which of the upcoming *opt* deletions one of the nodes of this biggest remaining graph is involved, and which of the nodes it is. The following proof has to take into account a few more subtleties: The remaining graph of the instance may be smaller than k, a remaining node can be part of multiple $H \in \mathcal{F}$ in the instance and single or multiple nodes of the remaining graph could be part of a copy of a detected $H \in \mathcal{F}$.

Theorem 2.7.4. *If $K_n, \overline{K_m} \in \mathcal{F}$ for some $n, m \in \mathbf{N}$, then there is an optimal online algorithm with advice for the* Delayed $\mathcal{F}$-Node-Deletion *problem that uses $O(\log opt)$ bits of advice.*

Proof. Let R be as in Lemma 2.7.2, and k be the size of the biggest graph in $\mathcal{F}$. Algorithm 2 uses at most

$$\lceil \log(R-1) \rceil + \lceil (R-1) \cdot \log(opt \cdot k) \rceil = O(\log opt)$$

bits of advice. We assume that the algorithm is given *opt* beforehand, which can be encoded using $O(\log opt)$ bits of advice using self-delimiting encoding as in [16]. The advisor computes an optimal offline solution. After deleting the nodes from G, a graph with at most $R-1$ nodes must remain, otherwise it would not be $\mathcal{F}$-free by Lemma 2.7.2. Let $u \leq R-1$ be the number of nodes that are considered by the online algorithm below and that will *not* be deleted. The advisor writes u onto the tape. Next, for all those u nodes $(v_i)_{i \leq u}$, the advisor computes in which round the algorithm considers this node for the first time. A node is considered by the algorithm if it is part of an induced forbidden graph $H \in \mathcal{F}$ that at some point is recognized. The node (v_i) can thus be identified by a pair $(r_i, a_i) \in \{1, \ldots, opt\} \times \{1, \ldots, k\}$. Then the algorithm encodes all these pairs $((r_i, a_i))_{i \leq u}$ onto the tape.

The algorithm starts by reading u and these pairs from the tape. Then it sets its *round counter* r to 1, and the set of *fixed nodes* (i.e. the set of nodes that it encounters and which will not be deleted) F to $\emptyset$. Whenever the algorithm finds a forbidden induced subgraph it checks in the list (r_i, a_i) which of its nodes it must not delete, and adds them to F. Then it deletes any other vertex from $W \setminus F$. □

This proof implies that, given some $\mathcal{F}$, if we can always bound the size of the graph remaining after deleting an optimal number of nodes by a constant, we can construct an algorithm that solves the Delayed $\mathcal{F}$-Node-Deletion with advice logarithmic in *opt*. Under certain conditions we also get a lower bound logarithmic in *opt* as we will see in the following two theorems.

Algorithm 2 Optimal Online Algorithm with Logarithmic Advice

```
Read ⌈log(R − 1)⌉ bits of advice, interpret as number u ∈ {1, ..., R − 1}
Read u · ⌈log(k · opt)⌉ bits of advice, interpret as u pairs ((r_i, a_i))_{i≤u} ⊆ {1, ..., opt} × {1, ..., k}
r ← 1, F ← ∅
for all t = 1, ..., |G| do
    G_t ← reveal next node
    while G_t[W] ≃ H ∈ F for some W ⊆ V(G_t) do
        for all i = 1, ..., u do
            if r == r_i then
                v_i ← a_i'th vertex of W
                F ← F ∪ {v_i}
        Delete any vertex from W \ F (or all at once)
        r ← r + 1
```

Theorem 2.7.5. *Let $K_n, \overline{K_m} \in \mathcal{F}$, and let D be a graph that is $\mathcal{F}$-free, $|D| = R-1$, and $||D||$ is maximal among such graphs. If D has no universal vertex, then any optimal online algorithm for the* Delayed $\mathcal{F}$-Node-Deletion *problem needs $\Omega(\log opt)$ bits of advice.*

Proof. Let c be the size of the biggest clique in D. Of course $c < n$, otherwise $K_n \trianglelefteq D$. Let $opt > n$, and select c distinct numbers $u_1, \ldots, u_c \in \{1, \ldots, opt + c\}$. Reveal K_{opt+c}, then continue to add $|D| - c$ nodes and join them with all nodes of the K_{opt+c} except $u_1, \ldots, u_c$. Between these newly added vertices and $\{u_1, \ldots, u_c\}$ add edges such that they form a graph isomorphic to D. The entire resulting graph G is then isomorphic to $K_{opt} \nabla D$. We claim that the unique optimal way to make G $\mathcal{F}$-free is to delete $X := V(K_{opt})$.

First we observe that X is a solution, because $G - X \simeq D$. It is also optimal because if we delete fewer vertices then we are left with more than $opt + |D| - opt = |D|$ nodes, and the resulting graph is not $\mathcal{F}$-free according to Lemma 2.7.2. For the uniqueness, suppose there was another optimal solution $X' \neq X$. Let $\overline{X}, \overline{X'}$ denote the respective complement sets, i.e. the nodes that are not deleted in the respective optimal solution. Then because D has no universal vertex we have

$$||G[\overline{X}]|| < ||G[\overline{X} \cap \overline{X'}]|| + |\overline{X'} \setminus \overline{X}| \cdot (|D| - 1) = ||G[\overline{X'}]||.$$

This would contradict the assumption that D was chosen such that $\|D\|$ is maximized. Thus, we can construct $\binom{opt+c}{c}$ such instances, each of which has a unique optimal solution, and the solutions are pairwise different. The nodes must be deleted already when the K_{opt+c} is revealed, hence we get a lower bound on the advice complexity of $\log\left(\frac{(opt+c)\cdot\ldots\cdot(opt+1)}{c!}\right) = \Omega(\log opt)$. □

With a similar construction for independent sets instead of cliques we get the following sufficient condition for the necessity of logarithmic advice.

Theorem 2.7.6. *Let $K_n, \overline{K_m} \in \mathcal{F}$, and let D be a graph that is $\mathcal{F}$-free, $|D| = R-1$, and $||D||$ is minimal among such graphs. If D has no isolated vertex, then any optimal online algorithm for the* Delayed $\mathcal{F}$-Node-Deletion *problem needs $\Omega(\log opt)$ bits of advice.*

2.8 Open Problems

With this chapter, we have touched upon many aspects of node- and edge-deletion problems. As is clear from Table 2.1, the work we started is not yet complete, with the advice complexity for the most general families of graphs still being open. Of course, the online complexity, both in the late accept and classical online model have only been partially analyzed by us and still warrant more research.

As a closely related field of analysis that is still open, the analysis of randomized algorithms on the delayed problem may prove to be very interesting.

Another very naturally related field of study is to look at the same problems but forbidding any proper, nontrivial subgraph from each $\mathcal{F}$ to occur. Of course, each such problem can be reduced to that of forbidden induced graphs. This can be done by constructing an $\mathcal{F}'$ by adding each possible supergraph H' for each $H \in \mathcal{F}$ with $|H'| = |H|$.

Chapter 3

The Online Simple Knapsack Problem with Reservations

The Online Simple Knapsack problem suffers from pathological families of instances, as we have seen in Example 1.2.1 of the introduction. Due to these specialized corner cases, a worst case analysis does not make sense for this problem in the classical online setting. In this chapter, we thus study a variation of the problem where one may pay a proportion of an item's gain $\alpha \in [0, 1]$ in order to "set aside" a given item. The decision on whether to pack or reject the specific item is then postponed, essentially allowing an algorithm to make parts of the instance *offline*. We call α the *reservation costs*, and fixing a value of α defines a corresponding α-Online Simple Knapsack problem. Of course, if the reservation costs are set to 0, the complete instance may be reserved, allowing an algorithm to find an optimal solution at no extra cost. On the other hand, the higher the reservation costs, the worse the trade-off between being able to the ability to postpone decisions and the additional costs that have to be paid for this freedom. Indeed, once the reservation costs reach 1, the option of reserving items becomes useless, as the complete potential gain of a reserved item is compensated by its reservation costs.

3.1 Problem Definitions and Notation

We will abuse notation in this chapter by using x_i for both the label of an item and for the size (or gain) of the same item.

Definition 3.1.1 (The Online Simple Knapsack with Reservations Problem)**.** Given a fixed constant $\alpha \in [0, 1]$, called *reservation costs*. Given a set of items $I = (x_1, \ldots, x_n)$ as a request sequence of items that arrive sequentially. Let $w : I \to [0, 1] \subset \mathbf{Q}$ be a size function and $g : I \to [0, 1] \subset \mathbf{Q}$ with $w = g$ be

a gain function. Let K and R be initially empty sets called the *knapsack* and *reservation buffer*, respectively. At each step $i \in \{1, \ldots, n\}$, the item x_i of the request sequence is given. An algorithm then has one of the following options:

- **Pack** If $\sum_{x_k \in K} w(x_k) + w(x_i) \leq 1$, set $K := K \cup x_i$.
- **Reserve** Set $R := R \cup x_i$.
- **Reject** Do nothing.

At step $n + 1$, an algorithm is informed that the request sequence has finished. It may then select a subset $R' \subseteq R$ and set $K := K \cup R'$, if $\sum_{x_k \in K} w(x_k) + \sum_{x_r \in R'} w(x_r) \leq 1$.

The Online Simple Knapsack with Reservations problem (Oskr) is for an algorithm ALG to maximize the sum of items in K, minus the *reservation costs*

$$gain_{\texttt{ALG}}(I) := \sum_{x_k \in K} g(x_k) - \alpha \sum_{x_r \in R} g(x_r) .$$

We denote the gain of an optimal algorithm by

$$gain_{\texttt{OPT}}(I) := \max_{K \subseteq I}(\sum_{x_k \in K} g(x_k) \mid \sum_{x_k \in K} w(x_k) \leq 1) .$$

As the instance will be always clear from the context, we will write $gain_{\texttt{ALG}}$ for $gain_{\texttt{ALG}}(I)$ and $gain_{\texttt{OPT}}$ for $gain_{\texttt{OPT}}(I)$. Since we look at the simple knapsack problem, we will write x for $w(x)$ and x for $g(x)$, since $w = g$.

Strictly speaking, any competitive ratio given in this chapter is a function of a fixed value α. To simplify notation, we will write c for $c(\alpha)$, as it is always clear from the context which α the competitive ratio refers to.

Since the size of the optimal solution, i.e. the size of the knapsack is bounded by a constant, we will only look at the *strict competitive ratio*, calling it *competitive ratio* anyway in this chapter for brevity. To further simplify notation, we define $S^{\Sigma} := \sum_{x_i \in S} x_i$ to denote the sum of item sizes in a given set.

We will sometimes be interested in the best possible packing of a given set of available items, i.e. the biggest subset of total size smaller or equal to one. For this, we define the function $\text{🔓} : 2^S \to 2^S$ ("*optpack*"), where S is a set of knapsack items with $\text{🔓}(S) \mapsto \arg\max_{T \subseteq S}(T^{\Sigma} \mid T^{\Sigma} \leq 1)$.

3.2 Related Work and Our Contributions

Many variations of the standard Online Knapsack problem have been studied in the last years, of which we will only discuss a selection here. Iwama and Zhang [49]

Table 3.1: Competitive ratios of the OSKR problem, depending on the given reservation costs α.

Reservation costs	Competitive ratio	Lower bound	Upper bound
$0 < \alpha \leq 0.25$	2	Thm. 3.3.1	Thm. 3.4.1
$0.25 < \alpha \leq \sqrt{2}-1$	$\frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$	Thm. 3.3.2	Thm. 3.4.1
$\sqrt{2}-1 < \alpha < \phi-1$	$2+\alpha$	Thm. 3.3.2	Thm. 3.4.2, 3.4.3
$\phi-1 \leq \alpha < 1$	$\frac{1}{1-\alpha}$	Thm. 3.3.2	Thm. 3.4.4

analyzed the performance of online algorithms if they were allowed to pack a marginally larger knapsack than an offline solution. In this model of *resource augmentation*, an offline algorithm has to pack a knapsack of capacity 1, whereas an online algorithm may pack a knapsack of capacity $1+\gamma$ for some $\gamma > 0$, showing that a ratio of $\frac{1}{\gamma}$ is strict in this setting. Iwama and Taketomi [48] studied a model in which items that have been already packed could be irrevocably removed, which is also called *preemption*. They show among other results for multi-knapsack problems, that the simple knapsack problem becomes ϕ-competitive, where ϕ is the golden ratio. Building on this result, Han and Makino [43] additionally allowed for a limited number of *cuts*, i.e. splitting of items into two sub-items. Han, Kawase and Makino [40], on the other hand, added the restriction that each removal of an item incurs a cost proportional to the size of the removed item. Böckenhauer et al. [14] recently studied a model in which a limited amount of removals could be reversed, showing that for even a single use of so-called *recourse*, there exists a 1.5-competitive online algorithm.

Zhou, Chakrabarty and Lukose [70] analyzed the online knapsack problem under the assumption that the size of each item is much smaller than the knapsack capacity and the ratio between the value and weight of an item is bounded within a given range. Thielen, Tiedemann and Westphal [67] studied a model in which the knapsack capacity increases over a given number of discrete time periods.

Another variation of the model is introducing a *buffer* of constant size, in which items can be stored for a later, final decision. Han et al. [42] studied the case where the buffer has at least as much capacity as the knapsack itself.

We study the behavior of the OSKR problem with reservation costs between $0 < \alpha < 1$. We analyze four sub-intervals for α separately with borders at 0.25, $\sqrt{2}-1$ and $\phi - 1 \approx 0.618$, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. The bounds that we are providing, which are illustrated in Figure 3.1, can be found in Table 3.1. We prove matching upper and lower bounds for each interval.

We also take a look at a subclass of algorithms for this problem, called *nonrejecting* algorithms i.e., algorithms that only pack (and then end) or reserve items.

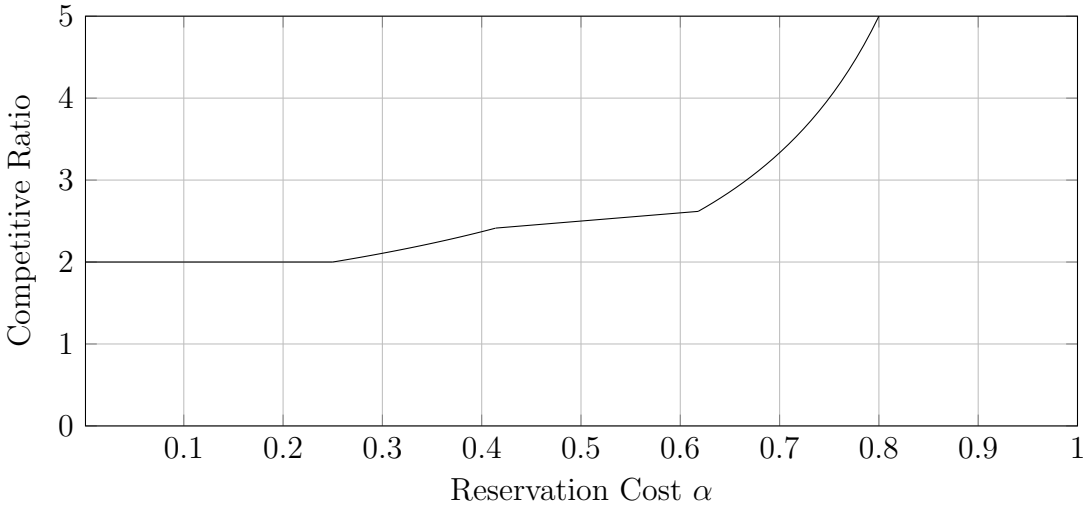


Figure 3.1: Competitive ratio of the Oskr problem, depending on the fixed value of α. The function diverges for $\alpha \to 1$.

We will see that these algorithms behave in a curious way: One may expect that — if the cost of reservation is very small — rejecting an item should not be necessary, as even when an item cannot be packed, the cost of reserving it is negligible. On the other hand, when the reservation cost is rising, aggressively reserving items may seem like a very bad strategy, as the risk of not being able to utilize reserved items may come to mind. Thus, the conclusion may be that nonrejecting algorithms perform well for small values of α and become worse for large values α. It turns out that the inverse is true, as we will show. For small values of α, every algorithm that is unable to reject items is strictly dominated by algorithms that are able to reject items. For larger values of α this is not the case, with our algorithms used for $\alpha \geq \sqrt{2} - 1$ being nonrejecting and matching their lower bounds.

We cannot give a definitive answer on why this behavior can be observed. Our intuition is that, for higher reservation costs, the behavior is more similar to the model without reservations. In this classical model, any errors that occur by ultimately not packing an item are punished severely, so while reserving everything is costly, it is not worse than rejecting any items.

We also look at a variation of the reservation model, in which reservation costs only have to be paid for items that are not used in the final packing. This model can be thought of as providing a down payment for an item, which is reimbursed only if you choose to actually buy it. We find that, curiously, even for reservation costs of $\alpha \geq 1$, the problem remains competitive. We provide a tight bound of 2 until $\alpha \leq 0.5$, at which point the competitive ration becomes $\frac{1}{2}(\sqrt{\alpha^2 + 2\alpha + 5} + \alpha + 1)$. The results, that can be found in Table 3.2, are visualized in Figure 3.2.

Table 3.2: Competitive ratios of the Oskr problem with reimbursements, depending on the given reservation costs α.

Reservation costs	Competitive ratio	Lower bound	Upper bound
$0 < \alpha \leq 0.5$	2	Cor. 3.6.1	Thm. 3.6.2
$0.5 < \alpha$	$\frac{1}{2}(\sqrt{\alpha^2 + 2\alpha + 5} + \alpha + 1)$	Thm. 3.6.1	Thm. 3.6.2

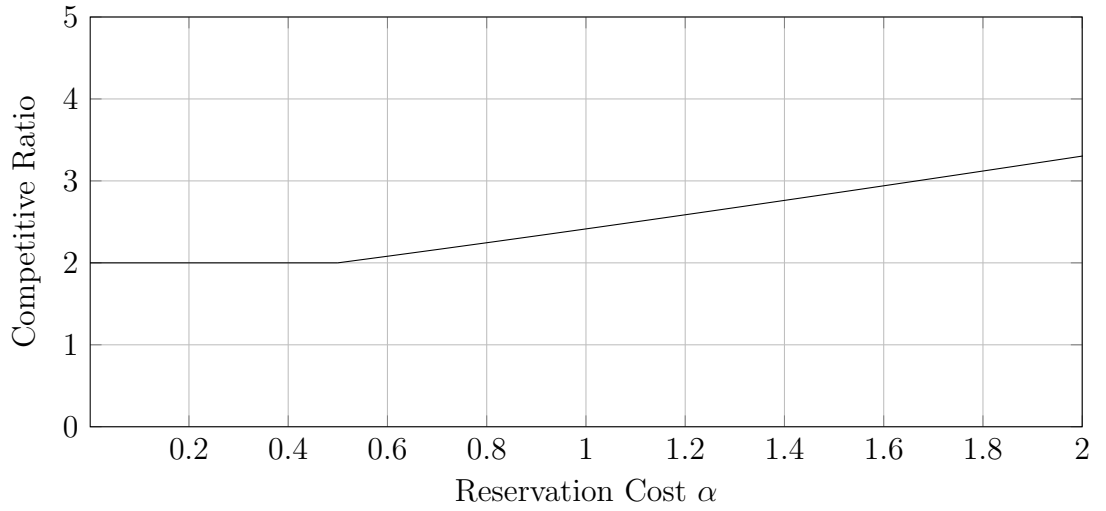


Figure 3.2: Competitive ratio of the Oskr problem with reimbursements, depending on the fixed value of α. The function does *not* diverge for α approaching any fixed value.

Interestingly, these competitive ratios for reservations with reimbursements depending on α match the competitive ratios of Han, Kawase and Makino [40], where they allow the removal of packed items for a given cost. We do however not see any simple reduction between the two models, since they can not arbitrarily "buffer" items as in our model and decisions on removals may have to be made before the complete instance was seen by an online algorithm.

The remainder of this chapter is structured as follows: We start by presenting the lower bounds for each of the mentioned four segments in Section 3.3. We then continue by presenting matching upper bounds in Section 3.4, splitting the section into three parts, with the first one handling the first two segments of α up to $\sqrt{2} - 1$, and another two subsections for the remaining two segments. We then go on to discuss the so-called *nonrejecting algorithms* in Section 3.5 and finally give a complete analysis of the model for reservations with reimbursements in Section 3.6. We conclude with a short discussion of open problems and possible future work in Section 3.7.

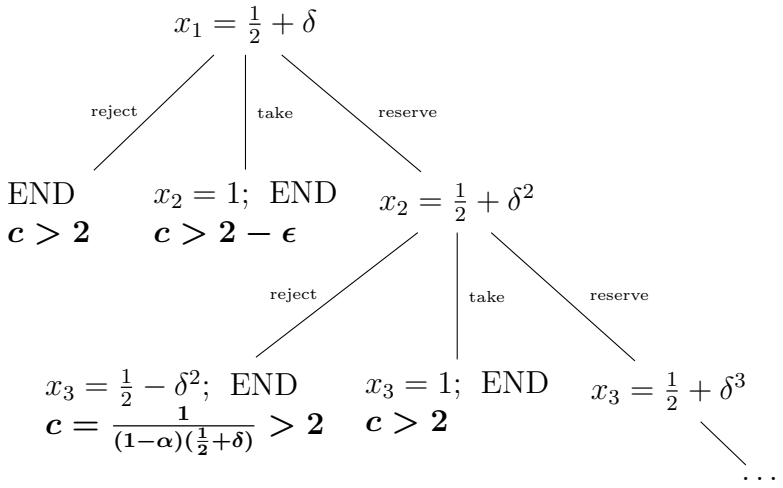


Figure 3.3: Sketch of the adversarial strategy that is used to show a lower bound of 2 on the competitive ratio of any algorithm in the proof of Theorem 3.3.1.

3.3 Lower Bounds

We start our analysis by first proving that no matter how small the reservation costs are, as long as they are positive no algorithm can be better than 2-competitive. The intuition for this is easily explained: As soon as an algorithm packs any item, an adversary can immediately present an item of size 1, which is then optimal. Thus, an algorithm could pay a very small reservation cost to keep all options open and to make sure that the optimal solution only consists of items that the algorithm holds as well. However, since the adversary can arbitrarily pad an instance, as we will see shortly, this option is doomed as well. Once an algorithm rejects an item, this rejected item will naturally be part of an optimal solution.

Theorem 3.3.1. *Given a fixed α with $0 < \alpha < 1$. There exists no algorithm solving the* Oskr *problem with a competitive ratio better than* 2.

Proof. We give a set of adversarial strategies and then do a full case distinction. This set of strategies is depicted in Figure 3.3. Given any $\varepsilon > 0$ and a δ with $0 < \delta \ll \varepsilon$, the adversary first presents an item $x_1 = \frac{1}{2} + \delta$.

Case 1: An algorithm packs $x_1 = \frac{1}{2} + \delta$:

If an algorithm packs x_1, the next item is presented of size 1, which does not fit into the algorithm's knapsack. The competitive ratio is then $1/(\frac{1}{2} + \delta) > 2 - \varepsilon$.

Case 2: An algorithm reserves $x_1 = \frac{1}{2} + \delta$:

If an algorithm reserves x_1, the next item is $x_2 = \frac{1}{2} + \delta^2$.

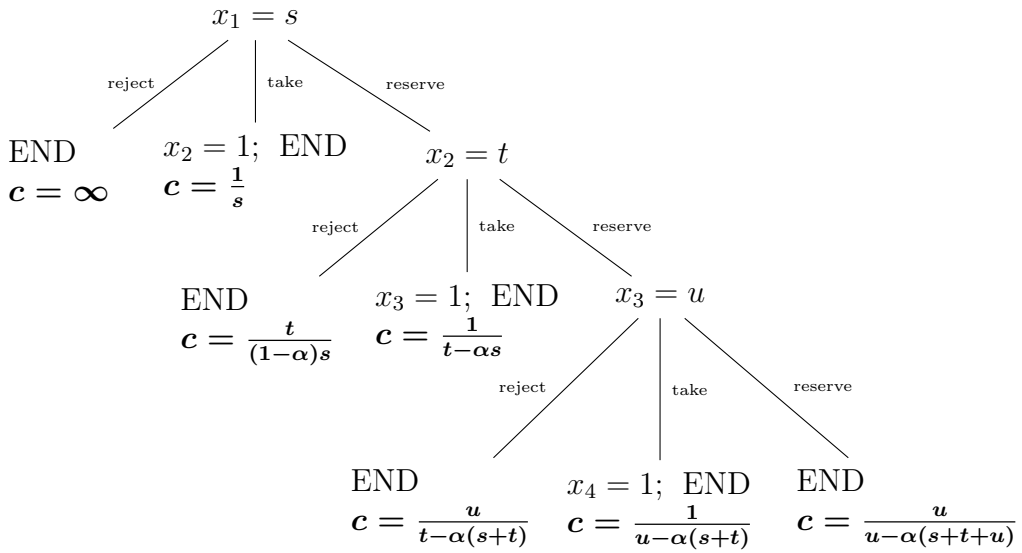


Figure 3.4: Generic adversarial strategy with up to four items, where $s < t < u < 1$ and $s+t > 1$. This strategy is used to prove the lower bounds on the competitive ratio of any algorithm stated in Theorem 3.3.1.

Case 2.1: An algorithm packs $x_2 = \frac{1}{2} + \delta^2$:

If an algorithm packs x_2, the next item is 1, which does not fit into the algorithm's knapsack. The competitive ratio is then $1/(\frac{1}{2} + \delta)(1 - \alpha) > 2$.

Case 2.2: An algorithm reserves $x_2 = \frac{1}{2} + \delta^2$:

The branches for packing and rejecting an item correspond to cases 2.1 and 2.3, with the main difference that the algorithm accumulated additional reservation costs that it has to deduct from its gain.

If an algorithm instead indefinitely reserves items, eventually the sum of reservation costs exceeds 1, at which point the algorithm cannot be competitive.

Case 2.3: An algorithm rejects $x_2 = \frac{1}{2} + \delta^2$:

If an algorithm rejects x_2, the next item is $x_3 = \frac{1}{2} + \delta^2$. This item fits together with x_2, completely filling a knapsack, but not with x_1. Thus, the competitive ratio is bounded by $1/((\frac{1}{2} + \delta^2) - (\frac{1}{2} + \delta)(1 - \alpha)) > 2$.

Case 3: An algorithm rejects $x_1 = \frac{1}{2} + \delta$:

If an algorithm rejects x_1, the instance ends. As the algorithm packed no item, the competitive ratio is unbounded. □

This strategy provides us with a lower bound that does not match the upper bound for $\alpha > 0.25$. For the rest of the range of α, we use a more generic approach. We look at an instance of four items $\{s, t, u, v\}$. We build the complete tree over all options that an algorithm has: packing, reserving and rejecting. We then look at the resulting bounds, which are of course dependent on the sizes of the four

generic items. We then decide in each branch whether the full four items should be presented and calculate fitting values for $\{s, t, u, v\}$. This strategy is depicted in Figure 3.4. We assume by construction that $s < t < u < 1$ and that $s + t > 1$. The competitive ratio of such a strategy is therefore bounded by

$$c \geq \min\Big\{\frac{1}{s},\ \frac{t}{(1-\alpha)s},\ \frac{1}{t-\alpha s},\ \frac{u}{t-\alpha(s+t)},\ \frac{1}{u-\alpha(s+t)},\ \frac{u}{u-\alpha(s+t+u)}\Big\}. \tag{3.1}$$

To prove a lower bound for every $0 < \alpha < 1$, we can choose s, t and u in order to make (3.1) as large as possible. Standard calculus leads to the bounds in the following theorem.

Theorem 3.3.2. *There exists no algorithm solving the* Oskr *problem with a competitive ratio better than*

- $\frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$, *for* $0.25 \leq \alpha \leq \sqrt{2}-1$;
- $2+\alpha$, *for* $\sqrt{2}-1 \leq \alpha < \phi - 1$; *and*
- $\frac{1}{1-\alpha}$, *for* $\phi - 1 \leq \alpha \leq 1$.

Proof. We give a set of adversarial strategies that use at most four items $I = \{s, t, u, v\}$, where we impose the restriction that $s + t > 1$. We will set these items to specific sizes later, depending on the size of α. Let $s \leq 0.5$ be the item that is presented first. As long as there are items left, the adversary will act as follows, depending on the behavior of an algorithm:

Case 1: An algorithm packs *s*:
The next item is $t = 1$, after which no further items are presented. The competitive ratio is then $\frac{1}{s}$.

Case 2: An algorithm reserves *s*:
The adversary presents the next item t.

Case 2.1: An algorithm packs *t*:
The next item is $u = 1$, after which no further items are presented. The competitive ratio is then $\frac{1}{t-\alpha s}$.

Case 2.2: An algorithm reserves *t*:
The adversary presents the next item u such that $u > t$.

Case 2.2.1: An algorithm packs *u*:
The next item is $v = 1$, after which no further items are presented. The competitive ratio is then $\frac{1}{u-\alpha(s+t)}$.

Case 2.2.2: An algorithm reserves *u*:
No further items are presented. The competitive ratio is then $\frac{u}{u-\alpha(s+t+u)}$.

Case 2.2.3: An algorithm rejects *u*:
No further items are presented. The competitive ratio is then $\frac{u}{t-\alpha(s+t)}$.

Case 2.3: An algorithm rejects t:
No further items are presented. The competitive ratio is then $\frac{t}{(1-\alpha)s}$.
Case 3: An algorithm rejects s:
No further items are presented. As the knapsack of the algorithm is empty, it is not competitive.

This strategy is also depicted in Figure 3.4. There are cases in which we will choose not to present all items, which we will handle explicitly.
Lower bound for $0.225 \leq \alpha \leq \sqrt{2}-1$:
We choose

$$s = \frac{2}{3+\sqrt{5-4\alpha}} + \varepsilon, \quad t = \frac{\sqrt{5-4\alpha}-1+2\alpha}{2(1+\alpha)}, \text{ and } u = \frac{\alpha+\sqrt{4(t-\alpha)+\alpha^2}}{2},$$

and next verify that $c \geq \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$ by comparing c to all terms of (3.1), with the given substitutions. We see immediately that $\frac{1}{s} = \frac{3+\sqrt{5-4\alpha}}{2} \geq \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$ for $\alpha \leq \sqrt{2}-1$, that $\frac{1}{t-\alpha s} = \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$, and that $\frac{t}{s-\alpha(s+t)} = \frac{2}{-1+\sqrt{5-4\alpha}-2\alpha} \geq \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$ for $\alpha \leq \sqrt{2}-1$.

The last three expressions to check are $\frac{1}{u-\alpha}$, $\frac{u}{t-\alpha}$ and $\frac{u}{u(1-\alpha)-\alpha}$. For the given u, we see that $\frac{1}{u-\alpha} = \frac{u}{t-\alpha} \geq \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$ for $\alpha \geq \alpha_0$, where $\alpha_0 \approx 0.224$ is the unique positive root of $x^4+2x^3-2x^2+5x-1$. Finally, we observe that $\frac{u}{u(1-\alpha)-\alpha} \geq \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$ for $\alpha \geq 0.19$ and in particular through the desired range.

Note that $2 \geq \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$ for $\alpha \leq 0.25$, so while the lower bound is correct for values of α down to approximately 0.224, the lower bound of 2 from Theorem 3.3.1 dominates it up to $\alpha = 0.25$.
Lower bound for $\sqrt{2}-1 \leq \alpha < \phi - 1$:
We choose

$$s = \frac{1}{2+\alpha}, \quad t = 1 - s + \varepsilon, \text{ and } u = \emptyset,$$

i.e. no element u is presented. This allows us to rewrite Equation (3.1) as

$$c \geq \min\Big\{\frac{1}{s},\ \frac{1}{t-\alpha s},\ \frac{t}{t-\alpha s},\ \frac{t}{t-\alpha(s+t)}\Big\}.$$

Substituting the values of s and t yields

$$c \geq \min\Big\{2+\alpha,\ \frac{1+\alpha}{1-\alpha},\ \frac{1+\alpha}{1-\alpha-\alpha^2}\Big\}.$$

It is easy to check that $\frac{1+\alpha}{1-\alpha} \geq 2+\alpha$ for $\alpha \geq \sqrt{2}-1$ and that $\frac{1+\alpha}{1-\alpha-\alpha^2} \geq \frac{1+\alpha}{1-\alpha}$ as long as the denominator is not negative or zero, i.e. as long as $\alpha < \phi - 1$.

Algorithm 3 $\max\{2, \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}\}$-competitive Algorithm for $0 < \alpha \leq \sqrt{2}-1$

1: Let $\mu := 1/(c(1-\alpha))$
2: **for** $k = 1, \ldots, n$ **do**
3: **if** $x_k + (1-\alpha)R^{\Sigma} < \frac{1}{c}$ **then Reserve** x_k
4: **else if** $x_k + R^{\Sigma} \leq 1$ **then Pack** $\hat{\mathcal{B}}(R \cup x_k)$. END
5: **else if** $\forall j \in R : x_j \leq 1-\mu$ **then Pack** $\hat{\mathcal{B}}(R \cup x_k)$. END
6: **else if** $\hat{\mathcal{B}}(R \cup x_k)^{\Sigma} - \alpha R^{\Sigma} \geq \frac{1}{c}$ **then Pack** $\hat{\mathcal{B}}(R \cup x_k)$. END
7: **else Reject** x_k
8: **Pack** $\hat{\mathcal{B}}(R)$.

Lower bound for $\phi - 1 \leq \alpha < 1$:
For the final range of values of α, we simply choose

$$s = 1-\alpha,\ t = \emptyset, \text{ and } u = \emptyset,$$

i.e. no elements t or u are presented. This simplifies Equation (3.1) to

$$c \geq \min\Big\{\frac{1}{s},\ \frac{s}{s-\alpha s}\Big\}.$$

Putting in $s = 1-\alpha$ yields $c \geq \frac{1}{1-\alpha}$. □

3.4 Upper Bounds

For the upper bound analysis, we have to deal with the reservation costs α lying in four different intervals, as already hinted at in Table 3.1.

3.4.1 Small Reservation Costs

We start by presenting Algorithm 3 that we use to prove upper bounds for the first two sections. This algorithm reserves items aggressively at first but after exceeding a threshold rejects all items not guaranteeing the desired competitive ratio.

Theorem 3.4.1. *Given a fixed α with $0 < \alpha < \sqrt{2}-1$. Algorithm 3 solves the* Oskr *problem with a competitive ratio of at most* $\max\{2, \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}\}$.

Proof. We first deal with the simple cases. Let $\mu := 1/(c(1-\alpha))$ as defined in the algorithm and $c = \max\{2, \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}\}$, depending on the concrete given α.

Only in line 3 can a reservation be made, if one criterion is met: If the gain from the current item plus the gain of everything reserved so far is still below the

wanted gain, the item is reserved. This especially means that the current item and the gain from the reserve itself are both smaller than $\frac{1}{c}$ if this condition triggers.

If an item x_k is *not* reserved, we thus know that this item together with the pure gain of all reserved items sum up to the wanted size, i.e. to $\frac{1}{c}$. It can however be the case that such a packing is invalid by exceeding the size of the knapsack. Thus, the algorithm is either faced with a valid packing and trivially reaches the wanted competitive ratio in line 4, or it continues otherwise.

Assuming thus that packing all reserved items together with x_k would exceed the size of the knapsack, there could still be a packing using only a subset of these items. As the algorithm did not reserve x_k in line 3, we know that $(1-\alpha)R^\Sigma < \frac{1}{c}$ and thus $R^\Sigma < \frac{1}{c(1-\alpha)} = \mu$, since the algorithm does not reserve items that would violate this conditions. As x_k is too large to be packed with all reserved items, which was established by the check in line 4, we can deduce that $x_k > 1-\mu$. If all items from R are smaller than $1-\mu$, as checked in line 5, we know that a packing with our wanted gain must exist: If one simply removes items from R until $x_k + R^\Sigma \leq 1$ holds and the biggest possible item removed this way is bounded by $1-\mu$, the size of $x_k + R^\Sigma$ is then at least $1-(1-\mu) = \mu$. Since the reservation costs are at most $\alpha\mu$, since $R^\Sigma < \mu$, the gain of the algorithm is at least $\mu - \alpha\mu = \frac{1}{c}$.

This leaves us with a corner case in which the previous conditions hold, except that there is at least one item in R which is bigger than $1-\mu$. The algorithm then simply calculates in line 6 if the algorithm is able to achieve the wanted competitive ratio using the new item x_k anyway and reject the item in line 7 otherwise.

We are thus left with the most involved case, which is the algorithm not triggering any of the previous stopping conditions and instead ending in line 8.

We first argue that, whenever a new item x_k is rejected, R contains exactly one item of size at least $1-\mu$. We already know that for x_k to be rejected, all conditionals had to fail. Due to the condition in line 5 we thus know that R always contains such an item bigger than $1-\mu$ once line 7 is reached. Assume now that such an item is already in R and that for the current item x_k it also holds that $x_k \geq 1-\mu$. Then the algorithm will not reserve this item, since

$$\begin{aligned} x_k + (1-\alpha)R^\Sigma &\geq (1-\mu) + (1-\mu)(1-\alpha) \\ &= \left(1 - \frac{1}{c(1-\alpha)}\right)(2-\alpha) \geq \frac{1}{c} \end{aligned}$$

where the last inequality holds for all $\alpha \leq 1 - 1/\sqrt{2} \approx 0.293$ when setting $c = 2$, thus in particular for $\alpha \leq 0.25$. It also holds for any $0 < \alpha < 1$ when setting $c = \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$. Of note is that this inequality does *not* hold for the desired range between $\sqrt{2}-1 < \alpha < \phi - 1$ when setting $c = 2+\alpha$. Summarizing this paragraph,

the algorithm will hold exactly one item of size larger or equal $1-\mu$ in its reserve if it stops in line 7 and any item had to be rejected.

Next, we argue that any item x_k that is rejected is of size larger than 0.5. This can be easily verified seeing that for such an item the condition in line 4 does not trigger and thus $x_k > 1 - R^{\Sigma}$ holds. Since by rejecting this item we know that R contains at least one item of size at least $1-\mu$, it has to hold that $x_k > \mu$, which is bigger than 0.5 for $0 < \alpha < \sqrt{2} - 1$.

The difference between the items that the algorithm can use for its final packing and the items that are used for an optimal solution are exactly those items that are rejected by the algorithm. We now fix an optimal solution. Since all these rejected items are larger than 0.5, the optimal solution and the algorithms solution can thus differ by at most one item. Thus, we can ignore any items of an instance that both the algorithm and an optimal solution discard and fix one item of size larger than 0.5 as the unique item that is part of the optimal solution, but discarded by the algorithm.

Let $x_{\texttt{OPT}}$ be the unique item that the algorithm rejects, but that is part of the optimal solution. Let $x_{\texttt{ALG}}$ be the largest reserved item of the algorithm at the time that $x_{\texttt{OPT}}$ is rejected. As the algorithm discards $x_{\texttt{OPT}}$, we know that both $x_{\texttt{OPT}} \leq \text{🖬}(x_{\texttt{OPT}} \cup R_s) < \frac{1}{c} + \alpha R^{\Sigma}$ and $x_{\texttt{OPT}} + x_{\texttt{ALG}} > 1$ have to hold. We also know that the only rejected item from the optimal solution is $x_{\texttt{OPT}}$ and that every other item of this solution was reserved by our algorithm.

Further, let $r^{\leftarrow}$ be the sum of reserved items at the time $x_{\texttt{OPT}}$ is revealed. Similarly, let $r^{\rightarrow}$ be the sum of items that are reserved after $x_{\texttt{OPT}}$ is rejected. At the moment that $x_{\texttt{OPT}}$ is rejected we know, due to line 6, that $x_{\texttt{OPT}} \leq \text{🖬}(x_i \cup R)^{\Sigma} \leq \frac{1}{c} + \alpha(x_{\texttt{ALG}} + r^{\leftarrow})$. We use this term to get from step (3.2) to step (3.3) of the following equation. Together with $x_{\texttt{OPT}} + x_{\texttt{ALG}} > 1$ we can derive $1 - x_{\texttt{ALG}} < x_{\texttt{OPT}} \leq \frac{1}{c} + \alpha(x_{\texttt{ALG}} + r^{\leftarrow})$ which, solved for $x_{\texttt{ALG}}$, yields the inequality $\frac{1-\frac{1}{c}-\alpha r^{\leftarrow}}{1+\alpha} \leq x_{\texttt{ALG}}$. This inequality is used between steps (3.7) and (3.8) of the following inequality chain.

To summarize, the optimal algorithm has a gain of at most $x_{\texttt{OPT}} + r^{\leftarrow} + r^{\rightarrow}$. Algorithm 3 can also pack the items from $r^{\leftarrow}$ and $r^{\rightarrow}$, but packs $x_{\texttt{ALG}}$ instead of $x_{\texttt{OPT}}$ and has to pay reservation costs for $x_{\texttt{ALG}}, r^{\leftarrow}$ and $r^{\rightarrow}$. Hence, we have

$$\frac{gain_{\texttt{OPT}}}{gain_{\texttt{ALG}}} \leq \frac{x_{\texttt{OPT}} + r^{\leftarrow} + r^{\rightarrow}}{(1-\alpha)(x_{\texttt{ALG}} + r^{\leftarrow} + r^{\rightarrow})} \qquad x_{\texttt{OPT}} \leq \frac{1}{c} + \alpha(x_{\texttt{ALG}} + r^{\leftarrow}) \quad (3.2)$$

$$\leq \frac{1}{1-\alpha} \cdot \frac{\frac{1}{c} + (x_{\texttt{ALG}} + r^{\leftarrow})\alpha + r^{\leftarrow} + r^{\rightarrow}}{x_{\texttt{ALG}} + r^{\leftarrow} + r^{\rightarrow}} \qquad \text{Add } \alpha r^{\rightarrow} \text{ to numerator} \quad (3.3)$$

$$\leq \frac{1}{1-\alpha} \cdot \frac{\frac{1}{c} + x_{\texttt{ALG}}\alpha + (r^{\leftarrow} + r^{\rightarrow})(1+\alpha)}{x_{\texttt{ALG}} + r^{\leftarrow} + r^{\rightarrow}} \qquad \text{Factor out } 1+\alpha \quad (3.4)$$

$$= \frac{1+\alpha}{1-\alpha} \cdot \frac{\frac{1}{c(1+\alpha)} + x_{\texttt{ALG}}\frac{\alpha}{1+\alpha} + r^{\leftarrow} + r^{\rightarrow}}{x_{\texttt{ALG}} + r^{\leftarrow} + r^{\rightarrow}}. \quad (3.5)$$

If the fraction right of $\frac{1+\alpha}{1-\alpha}$ is at most 1, we are done: It holds that $\frac{1+\alpha}{1-\alpha} \leq 2$ for $\alpha \leq 1/3$ and also $\frac{1+\alpha}{1-\alpha} \leq \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$ for $\alpha \leq \sqrt{2}-1$. Otherwise, we use that removing the term $r^{\leftarrow}+r^{\rightarrow}$ from both the numerator and the denominator can only increase the value of the right fraction and continue as follows.

$$\frac{gain_{\texttt{OPT}}}{gain_{\texttt{ALG}}} \leq \frac{1+\alpha}{1-\alpha} \cdot \frac{\frac{1}{c(1+\alpha)} + x_{\texttt{ALG}}\frac{\alpha}{1+\alpha}}{x_{\texttt{ALG}}} \qquad \text{Simplify equation} \tag{3.6}$$

$$= \frac{\frac{1}{cx_{\texttt{ALG}}} + \alpha}{1-\alpha} \qquad \text{Use } x_{\texttt{ALG}} \geq \frac{1-\frac{1}{c}+\alpha r^{\leftarrow}}{1+\alpha} \tag{3.7}$$

$$\leq \frac{\frac{1+\alpha}{c-1+c\alpha r^{\leftarrow}} + \alpha}{1-\alpha} \qquad \text{Remove } \alpha c r^{\leftarrow} \tag{3.8}$$

$$\leq \frac{\frac{1+\alpha}{c-1} + \alpha}{1-\alpha} \tag{3.9}$$

At this point, we handle the two upper bounds that we want to prove separately. We start with $c \geq 2$ for $\alpha \leq 0.25$.

$$\frac{\frac{1+\alpha}{c-1} + \alpha}{1-\alpha} \qquad \text{Replace } c \text{ by } 2 \ (c \geq 2)$$

$$\leq \frac{1+2\alpha}{1-\alpha}$$

This gives us our wanted competitive ratio of at most $c = 2$ for $\alpha \leq 0.25$. Likewise, we handle the case of $c \geq \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$ for $0.25 \leq \alpha \leq \sqrt{2}-1$.

$$\frac{\frac{1+\alpha}{c-1} + \alpha}{1-\alpha} \qquad \text{Replace } c \text{ by } \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$$

$$\leq \frac{1+\sqrt{5-4\alpha}}{2(1-\alpha)}$$

This concludes the proof. □

3.4.2 Medium Reservation Costs

We continue by giving the remaining two upper bounds for values of α between $\sqrt{2}-1$ and $\phi-1$, as well as for values between $\phi-1$ and 1, starting with the first of the two intervals.

For this, we will analyze a very simple algorithm that aggressively reserves items until it can pack an item that guarantees its wanted competitive ratio, namely Algorithm 4.

Algorithm 4 $(2+\alpha)$-competitive Algorithm for $0 < \alpha \leq \phi - 1$

```
for k = 1, ..., n do
    if x_k + (1 − α)R^Σ ≥ 1/(2+α) then
        Pack 🖃R ∪ x_k. END
    else
        Reserve x_k;
Pack 🖃R.
```

We split the proof into two pieces, since proving the upper bound of $2 + \alpha$ for $\alpha \leq 0.5$ is quite simple and we can use the proof as a basis for the more involved proof of the rest of the interval. We start with two simple and one slightly more involved technical lemmata.

Lemma 3.4.1. *During any run of Algorithm 4 it holds that* $R^\Sigma < \frac{1}{(2+\alpha)(1-\alpha)}$.

Proof. Consider a request sequence $x_1, \ldots, x_n$ for Algorithm 4, such that x_k triggers the packing in line 3, i.e. $x_k + (1-\alpha)R^\Sigma \geq \frac{1}{2+\alpha}$.

We know that the previous item of the instance x_{k-1} did not trigger the packing, i.e. $x_{k-1} + (1-\alpha)R'^\Sigma < \frac{1}{2+\alpha}$, where $R' = R \setminus x_{k-1}$. The gain of an item is of course reduced by its reservation costs when reserving it, i.e. $x_{k-1} \geq (1-\alpha)x_{k-1}$, from which we can deduce that $(1-\alpha)R^\Sigma = (1-\alpha)x_{k-1} + (1-\alpha)R'^\Sigma \leq x_{k-1} + (1-\alpha)R'^\Sigma < \frac{1}{2+\alpha}$.

As the inequality chain yields $(1-\alpha)R^\Sigma < \frac{1}{2+\alpha}$, it holds that $R^\Sigma < \frac{1}{(2+\alpha)(1-\alpha)}$. □

This is by construction the largest possible value that R^Σ can take before the algorithm packs an item and stops. The next lemma uses this fact and simply states that if Algorithm 4 should ever end in line 3, its gain is then also sufficiently large to yield the wanted competitive ratio.

Lemma 3.4.2. *If Algorithm 4 is able to pack items of total size at least* $\frac{1}{(2+\alpha)(1-\alpha)}$*, then its competitive ratio is at most* $2 + \alpha$.

Proof. If the algorithm packs items of total size at least $\frac{1}{(2+\alpha)(1-\alpha)}$, its gain is lower bounded by $gain_{\texttt{ALG}} \geq \frac{1}{(2+\alpha)(1-\alpha)} - \alpha R^\Sigma$. By Lemma 3.4.1, we know that $R < \frac{1}{(2+\alpha)(1-\alpha)}$, which means that $gain_{\texttt{ALG}} \geq \frac{1}{(2+\alpha)(1-\alpha)} - \frac{\alpha}{(2+\alpha)(1-\alpha)} = \frac{1}{2+\alpha}$, providing us with the desired upper bound on the competitive ratio. □

For the analysis, the following lemma will be crucial. It essentially allows us to partially reorder any given instance in non-increasing order, without changing the competitive ratio of Algorithm 4.

Lemma 3.4.3. *Let $I = x_1, \ldots, x_n$ be an instance for Algorithm 4, that triggers it to pack some item x_k and assume that the ratio between the gain of an optimal solution and Algorithm 4 was worse than $2+\alpha$ on I. Then there is another instance $I' = x_{i_1}, x_{i_2}, x_{i_3}, \ldots, x_{i_{k-1}}, x_k, \ldots, x_n$ where $(i_1, i_2, \ldots, i_{k-1})$ is a permutation of $(1, \ldots, k-1)$ and $x_{i_1} \geq x_{i_2} \geq \ldots \geq x_{i_{k-1}}$, and the ratio between the gain of the optimal solution and the algorithm is also larger than $2 + \alpha$ on I'.*

Proof. We show how to transform the original sequence such that the number of inversions is reduced by one. Repeating this transformation yields the desired result.

Let $I = x_1, \ldots, x_n$ be an instance for Algorithm 4, such that packing is triggered by x_k with a ratio of gains larger than $2 + \alpha$. Assume that $x_i < x_{i+1}$ for some $i < k - 1$. We prove that we will get an instance with a competitive ratio larger than $2 + \alpha$ again if we swap the positions of x_i and x_{i+1}. It is easy to see that this transformation preserves the sum of the first $k - 1$ items. To show that it is a sequence that does not trigger the packing after the ith or $(i + 1)$st item, we have to verify that

$$x_{i+1} < \frac{1}{2+\alpha} - (1-\alpha)(x_1 + \cdots + x_{i-1}) \tag{3.10}$$

and that

$$x_i < \frac{1}{2+\alpha} - (1-\alpha)(x_1 + \cdots + x_{i-1} + x_{i+1}) \; . \tag{3.11}$$

Here, (3.10) is straightforward, since x_k for $k > i + 1$ triggering the packing in the unmodified instance implies $x_{i+1} < \frac{1}{2+\alpha} - (1 - \alpha)(x_1 + \cdots + x_i)$. We prove (3.11) by means of a contradiction. Assume, that

$$x_i \geq \frac{1}{2+\alpha} - (1-\alpha)(x_1 + \cdots + x_{i-1} + x_{i+1}) \; . \tag{3.12}$$

For $1 \leq j \leq i + 1$, we also know that

$$x_j \leq \frac{1}{2+\alpha} - (1-\alpha)(x_1 - \ldots - x_{j-1}) \; , \tag{3.13}$$

and by construction that

$$x_i < x_{i+1} \; . \tag{3.14}$$

If we define $z = \frac{1}{2+\alpha} - (1 - \alpha)(x_1 + \cdots + x_{i-1})$, then we can rewrite (3.12), (3.13) for $j = i + 1$, and (3.14) as

$$\begin{aligned} x_i &\geq z - (1-\alpha)x_{i+1} \\ x_{i+1} &\leq z - (1-\alpha)x_i \\ x_i &< x_{i+1} \; . \end{aligned}$$

If we modify the first two equations to isolate the term $\frac{z}{1-\alpha}$, we obtain

$$\frac{x_i}{1-\alpha} + x_{i+1} \geq \frac{z}{1-\alpha}$$
$$\frac{x_{i+1}}{1-\alpha} + x_i \leq \frac{z}{1-\alpha} ,$$

which means that

$$\frac{x_{i+1}}{1-\alpha} + x_i \leq \frac{x_i}{1-\alpha} + x_{i+1} ,$$

that is, these two equations are satisfiable if and only if $x_{i+1} \leq x_i$, which is a contradiction to the assumption that $x_i < x_{i+1}$.

It remains to show that the new sequence has a competitive ratio larger than $2+\alpha$. Although the order of $x_1, \ldots, x_{k-1}$ has been changed, the requests are still the same and all of them are reserved and trigger a packing in x_k with the same reservation size, just as before the transformation. Hence, if there was no way to pack the items well enough before the transformation, it also cannot be done afterwards. □

We will use this lemma in the following proofs by first assuming that an instance in its original ordering is "bad" for the algorithm, as otherwise the competitive ratio can obviously be attained. Such a "bad" instance can then be reordered by Lemma 3.4.3 and will always result in the reordered instance being "good", meaning the assumption that the original instance is "bad" was false.

From Lemma 3.4.2 we know that if Algorithm 4 packs at least $\frac{1}{(2+\alpha)(1-\alpha)}$, this guarantees a competitive ratio of at most $2+\alpha$. Thus, if we have enough items that are smaller than the size of the "gap" of $1 - \frac{1}{(2+\alpha)(1-\alpha)}$, these items can be packed greedily and always achieve the desired competitive ratio. Let us call *small items* those of size smaller than

$$1 - \frac{1}{(2+\alpha)(1-\alpha)} = \frac{(2+\alpha)(1-\alpha)-1}{(2+\alpha)(1-\alpha)} = \frac{1-\alpha-\alpha^2}{(2+\alpha)(1-\alpha)} \tag{3.15}$$

This definition is only valid when (3.15) is positive, that is, when $1-\alpha-\alpha^2 \geq 0$, which is the case if $0 < \alpha \leq \phi - 1$, including our desired range. We call the remaining items *large items*. Let α_r be the unique positive real root of the polynomial $1 - 2\alpha - \alpha^2 + \alpha^3$, i.e.

$$\alpha_r = \frac{1}{3} + \frac{2\sqrt{7}}{3} \cos\left(\frac{1}{3} \arccos\left(-\frac{1}{2\sqrt{7}}\right) - \frac{2\pi}{3}\right) \approx 0.445 .$$

Lemma 3.4.4. *Given any request sequence $x_1 \geq x_2 \geq \ldots \geq x_{k-1}, x_k, \ldots, x_n$, where x_k triggers the packing in Algorithm 4, there is at most one large item in $\{x_1, \ldots, x_k\}$ if $0 < \alpha \leq \alpha_r$ and there are at most two large items in $\{x_1, \ldots, x_k\}$ if $\alpha_r < \alpha \leq 0.5$.*

Proof. Assume for a contradiction that there exists a request sequence where $x_1 \geq x_2 \geq \ldots \geq x_i \geq \frac{1-\alpha-\alpha^2}{(2+\alpha)(1-\alpha)}$, with $k > i$. Any item x_j with $j \leq i$ satisfies

$$x_j \leq \frac{1}{2+\alpha} - (1-\alpha)(x_1 + x_2 + \ldots + x_{j-1}) \,,$$

in particular,

$$x_i \leq \frac{1}{2+\alpha} - (1-\alpha)(x_1 + x_2 + \ldots + x_{i-1}) \,. \tag{3.16}$$

By the given ordering, x_i is maximized if $x_1 = x_2 = \ldots = x_i$. In this case, we obtain from (3.16) the upper bound $x_i \leq \frac{1}{2+\alpha} - (1-\alpha)(i-1)x_i$ which we solve for x_i and obtain $x_i \leq \frac{1}{(2+\alpha)(i(1-\alpha)+\alpha)}$. As x_i is a large item, it is also lower bounded by $x_i \geq \frac{1-\alpha-\alpha^2}{(2+\alpha)(1-\alpha)}$.

Thus we get $\frac{1}{(2+\alpha)(i(1-\alpha)+\alpha)} \geq \frac{1-\alpha-\alpha^2}{(2+\alpha)(1-\alpha)}$ which we solve for i to obtain $i \leq 1 + \frac{\alpha^2}{(1-\alpha)(1-\alpha-\alpha^2)}$. In particular, for $i = 2$ we get $2(1-\alpha)(1-\alpha-\alpha^2) \leq (1-\alpha)(1-\alpha-\alpha^2) + \alpha^2$ which is equivalent to $(1-\alpha)(1-\alpha-\alpha^2) \leq \alpha^2$ and thus to $1 - 2\alpha - \alpha^2 + \alpha^3 \leq 0$, which means that the number of large items is strictly smaller than 2 for $\alpha \leq \alpha_r$.

For $i = 3$, we get $3(1-\alpha)(1-\alpha-\alpha^2) \leq (1-\alpha)(1-\alpha-\alpha^2)+\alpha^2$ or equivalently $2(1-2\alpha+\alpha^3) \leq \alpha^2$. Hence, $2-4\alpha-\alpha^2+2\alpha^3 \leq 0$, which means that the number of large items is strictly smaller than 3 for $\alpha \leq 0.5$, since 0.5 is the unique positive real root of the left-hand-side polynomial that is smaller than one. □

With this, we are ready to prove the first part of the upper bound of $2+\alpha$ for values of α up to 0.5.

Theorem 3.4.2. *Given a fixed α with $0 < \alpha < 0.5$. Algorithm 4 solves the* Oskr *problem with a competitive ratio of at most $2+\alpha$.*

Proof. We first consider the case that $\alpha \leq \alpha_r$, where at most one large item can be part of the request sequence without triggering the algorithm to pack, by Lemma 3.4.4. If a given request sequence does not trigger a packing by the algorithm and the it instead ends in line 6, then the optimal packing is $x_1 + \ldots + x_n < 1$ and the competitive ratio is $\frac{1}{1-\alpha} < 2+\alpha$.

Assume thus that there is an instance $I = (x_1, \ldots, x_n)$ with $|I| \geq 2$ which triggers the packing of Algorithm 4 in line 3 at item x_k. If $x_1 + \ldots + x_k \leq 1$, the gain of the algorithm is $x_k + (1-\alpha)(x_1 + \ldots + x_{k-1})$, which is larger than $\frac{1}{2+\alpha}$ by the packing condition itself.

Thus, assume $x_1 + \ldots + x_k > 1$. If all items of the request sequence are *small*, Algorithm 4 packs x_k and then greedily items from its reservation buffer. Since by definition, small items are smaller than $1 - \frac{1}{(2+\alpha)(1-\alpha)}$, as soon as any item does not fit into the greedily packed knapsack, the gain of the algorithm is at least $\frac{1}{2+\alpha}$.

Else, there is exactly one large item in the instance, which has to be x_1 by Lemma 3.4.4. If $x_1 + x_k < 1$ holds, the algorithm can again simply greedily fill the rest of the knapsack with small items, guaranteeing the wanted bound. If $x_1 + x_k > 1$, we consider two cases. If $x_2 + \ldots + x_k \geq \frac{1}{(2+\alpha)(1-\alpha)}$, then by Lemma 3.4.1, the gain of these items is sufficient and the algorithm can pack them greedily. Otherwise, we are in the situation that $x_k > 1 - x_1$ and that $x_2 + \ldots + x_k < \frac{1}{(2+\alpha)(1-\alpha)}$. The algorithm is thus able to pack all items except for x_1 into the knapsack. Note that since x_1 did not trigger a packing, it is bounded by $x_1 < \frac{1}{2+\alpha}$. In this case, the gain of the Algorithm is then

$$\begin{aligned}
&x_2 + \ldots + x_k - \alpha(x_1 + \ldots + x_{k-1}) \\
&= x_k + (1-\alpha)(x_2 + \ldots + x_{k-1}) - \alpha x_1 \\
&> 1 - x_1 - \alpha x_1 + (1-\alpha)(x_2 + \ldots + x_{k-1}) \\
&= 1 - (1+\alpha)x_1 + (1-\alpha)(x_2 + \ldots + x_{k-1}) \\
&> 1 - \frac{(1+\alpha)}{(2+\alpha)} + (1-\alpha)(x_2 + \ldots + x_{k-1}) \\
&= \frac{1}{(2+\alpha)} + (1-\alpha)(x_2 + \ldots + x_{k-1}) \,,
\end{aligned}$$

where we used the fact that $x_k > 1 - x_1$, and that $x_1 < \frac{1}{2+\alpha}$ since it did not trigger the packing.

We next look at values of $\alpha < 0.5$, which means that the request sequence contains at most two large items. Again, if a request sequence does not trigger a packing by the algorithm and the it instead ends in line 6, then the optimal packing is $x_1 + \ldots + x_n < 1$ and the competitive ratio is $\frac{1}{1-\alpha} < 2 + \alpha$. Thus, we again assume that there is some shortest instance $I = (x_1, \ldots, x_n)$ which triggers Algorithm 4 to pack at item x_k in line 3.

Again, if $x_1 + \ldots, x_k \leq 1$ the gain of the algorithm is sufficient by construction. If there is at most one large item in the request sequence, we are back in the first part of the proof. Thus, also assume that there are exactly two large items in the request sequence, which are again ordered by Lemma 3.4.3. If either of the large items triggers a packing, it is either x_1, meaning it is of size at least $\frac{1}{2+\alpha}$. If on the other hand it is x_2 that triggers a packing, it fits with x_1 and yields sufficient gain by construction, as $x_1 \geq x_2$ by Lemma 3.4.3 and since $x_1 < 0.5$.

Thus, assume $x_1 + \ldots, x_k > 1$ and two large items in the instance that themselves do not trigger a packing with $x_1 \geq x_2$. This means that $x_1 < \frac{1}{2+\alpha}$ and that $x_2 < \frac{1}{2+\alpha} - (1-\alpha)x_1$. By using the size ordering we can further state that $x_2 < \frac{1}{2+\alpha} - (1-\alpha)x_2$ and thus $(2-\alpha)x_2 < \frac{1}{2+\alpha}$ which allows us to bound x_2 by $x_2 < \frac{1}{(2+\alpha)(2-\alpha)}$. Now, we fall in one of the following cases:

1. If $x_1 + x_2 + x_k \leq 1$, the algorithm obtains the desired competitive ratio by packing the three items and greedily filling up the knapsack with small items.

2. If $x_1 + x_2 + x_k > 1$, but $x_1 + x_k \leq 1$, we pack x_1 and x_k and pack the small items greedily. If the small items fill the knapsack until at least $\frac{1}{(2+\alpha)(1-\alpha)}$, we are done. Otherwise, the gain obtained by the algorithm is

$$\begin{aligned}
&x_1 + x_3 + \ldots + x_k - \alpha(x_1 + \ldots + x_{k-1}) \\
&= x_k + (1-\alpha)(x_1 + x_3 + \ldots + x_{k-1}) - \alpha x_2 \\
&> 1 - x_1 - x_2 + (1-\alpha)x_1 - \alpha x_2 + (1-\alpha)(x_3 + \ldots + x_{k-1}) \\
&= 1 - \alpha x_1 - (1+\alpha)x_2 + (1-\alpha)(x_3 + \ldots + x_{k-1}) \\
&> 1 - \frac{\alpha}{2+\alpha} - \frac{1+\alpha}{(2+\alpha)(2-\alpha)} + (1-\alpha)(x_3 + \ldots + x_{k-1}) \\
&= \frac{(2+\alpha)(2-\alpha) - \alpha(2-\alpha) - (1+\alpha)}{(2+\alpha)(2-\alpha)} + (1-\alpha)(x_3 + \ldots + x_{k-1}) \\
&= \frac{3(1-\alpha)}{(2+\alpha)(2-\alpha)} + (1-\alpha)(x_3 + \ldots + x_{k-1}) \\
&\geq \frac{1}{2+\alpha} + (1-\alpha)(x_3 + \ldots + x_{k-1}) \,.
\end{aligned}$$

   where the last step is true for any $\alpha \leq 1/2$, and we used $x_k > 1 - x_1 - x_2$ and the upper bounds $x_1 < \frac{1}{2+\alpha}$ and $x_2 < \frac{1}{(2+\alpha)(2-\alpha)}$ for x_1 and x_2.

3. If $x_1 + x_k > 1$, but $x_2 + x_k \leq 1$, we pack x_2 and x_k into the knapsack and the small items greedily. Here, again, if the small items fill the knapsack up to at least $\frac{1}{(2+\alpha)(1-\alpha)}$, we know that we obtain the desired competitive ratio. Otherwise, the gain obtained by the algorithm is $x_2 \ldots + x_k - \alpha(x_1 + \ldots + x_{k-1})$ which is exactly the same as in the case for $\alpha \leq \alpha_r$, and we can perform the same calculations and obtain the desired bound for the competitive ratio.

4. If $x_2 + x_k > 1$, we pack x_k into the knapsack and pack small items greedily. If these fill the knapsack up to at least $\frac{1}{(2+\alpha)(1-\alpha)}$, we know that we obtain the desired competitive ratio. Otherwise, the gain obtained by the algorithm is

$$\begin{aligned}
&x_3 + \ldots + x_k - \alpha(x_1 + \ldots + x_{k-1}) \\
&= x_k + (1-\alpha)(x_3 + \ldots + x_{k-1}) - \alpha x_2 - \alpha x_1 && x_k > 1 - x_2 \\
&\geq 1 - x_2 - \alpha x_1 - \alpha x_2 + (1-\alpha)(x_3 + \ldots + x_{k-1}) \\
&= 1 - \alpha x_1 - (1+\alpha)x_2 + (1-\alpha)(x_3 + \ldots + x_{k-1}) \,,
\end{aligned}$$

   at which point we see that we have seen this term in Case 2 where $x_1 + x_2 + x_k > 1$ but $x_1 + x_k \leq 1$.

□

As the proof of the remaining values of α for $0.5 < \alpha < \phi - 1$ will be an induction over the number of large items in an instance, we will introduce the following lemma to help us not having to explicitly handle small items. Essentially, we will assume in the following upper bound theorem that an optimal solution can always achieve a perfect gain of 1 either way and that an instance containing small items can never worsen the performance of Algorithm 4.

Lemma 3.4.5. *Given an instance without small elements for which the ratio between the gain of an optimal solution the gain of Algorithm 4 is larger than $2+\alpha$. Then adding small elements to this instance will only improve the gain of Algorithm 4.*

Proof. Let $I = (x_1, \ldots, x_n)$ be a request sequence of large elements $x_1 \geq \ldots \geq x_{k-1}, x_k, \ldots, x_n$, where x_k is the element triggering the packing for Algorithm 4, and for which the ratio between the gain of an optimal solution the gain of Algorithm 4 is larger than $2 + \alpha$. By Lemma 3.4.2, the total size of the items packed into the knapsack is then smaller than $\frac{1}{(2+\alpha)(1-\alpha)}$. If we add enough small elements to the request sequence before x_k, the small elements, by definition, can be packed greedily until the knapsack is filled up to $\frac{1}{(2+\alpha)(1-\alpha)}$, achieving the desired competitive ratio. If not enough small items are added before x_k, the small elements requested before x_k will still be able to be packed, so they will never contribute negatively to the total gain of the algorithm. □

We are now ready for the remaining theorem.

Theorem 3.4.3. *Given a fixed α with $0.5 < \alpha < \phi - 1$. Algorithm 4 solves the* Oskr *problem with a competitive ratio of at most $2 + \alpha$.*

Proof. We prove the statement by induction over the number of large elements before the element x_k triggering a packing of Algorithm 4. The base case for zero large elements is trivial, as the algorithm can simply greedily pack small items from its reservation buffer and be optimal. We thus assume that Algorithm 4 achieves a competitive ratio of at most $2 + \alpha$ on any instance with less than $k - 1$ large elements.

For the induction step, we thus assume an instance of $k - 1$ large elements, which we can assume to be ordered in a descending order based on their index, due to Lemma 3.4.4. By Lemma 3.4.5, we assume that the item x_k triggers the packing. We distinguish two cases, one in which x_{k-1} is part of an optimal packing, and the other in which it is not.

First assume that, when the packing is triggered, x_{k-1} is not part of an optimal packing. In this case,

$$\text{🏛}(\{x_1, \ldots, x_{k-2}\})^{\Sigma} + x_{k-1} > 1 \,. \tag{3.17}$$

We also have

$$x_{k-1} + (1-\alpha) \sum_{j<k-1} x_j < \frac{1}{2+\alpha} \,, \tag{3.18}$$

since x_{k-1} would trigger the packing otherwise. The gain that Algorithm 4 achieves is thus

$$\begin{aligned}
&\text{🏛}(\{x_1, \ldots, x_{k-2}\})^{\Sigma} - \alpha R^{\Sigma} \\
&> 1 - x_{k-1} - \alpha \sum_{j \le k-1} x_j && \text{using (3.17)} \\
&= 1 - (1+\alpha) x_{k-1} - \alpha \sum_{j<k-1} x_j \\
&> 1 - (1+\alpha)\left(\frac{1}{2+\alpha} - (1-\alpha) \sum_{j<k-1} x_j\right) - \alpha \sum_{j<k-1} x_j && \text{using (3.18)} \\
&= 1 - \frac{1+\alpha}{2+\alpha} + (1-\alpha^2) \sum_{j<k-1} x_j - \alpha \sum_{j<k-1} x_j && \text{simplify} \\
&= 1 - \frac{1+\alpha}{2+\alpha} + (1-\alpha-\alpha^2) \sum_{j<k-1} x_j \\
&> 1 - \frac{1+\alpha}{2+\alpha} && \alpha < \phi - 1 \\
&= \frac{1}{2+\alpha}
\end{aligned}$$

as we intended.

As the second case, assume that when the packing is triggered, x_{k-1} is part of an optimal packing. We consider two sub-cases.

1. Taking x_{k-1} out of the instance still triggers the packing. This means that $x_k + (1-\alpha)(R^{\Sigma} - x_{k-1}) \ge \frac{1}{2+\alpha}$ holds. We can thus consider the instance $x_1, \ldots, x_{k-2}, x_k$. This instance has $k-2$ large elements, and x_{k-1} cannot be part of its optimal solution, thus its competitive ratio is at most $2+\alpha$ by the induction hypothesis, and the competitive ratio after adding x_{k-1} can only get better.

2. Taking x_{k-1} out of the instance does not trigger the packing.

This means that

$$x_k + (1-\alpha)(R^\Sigma - x_{k-1}) < \frac{1}{2+\alpha} \text{ , and also} \tag{3.19}$$

$$x_k + (1-\alpha)R^\Sigma \geq \frac{1}{2+\alpha} \text{ ,} \tag{3.20}$$

and, if we let x_t be the smallest element that does not get packed, $x_t \geq x_{k-1}$ holds. Also, because of the optimality of the packing $x_k \geq x_t$, (otherwise one can take all of the reserved elements as the packing and obtain a better bound) and

$$\text{🖬}(\{x_1, \ldots, x_{k-2}\})^\Sigma - x_{k-1} + x_t > 1 \tag{3.21}$$

holds. With these bounds, the gain of the algorithm is at least

$$\begin{aligned}
&\text{🖬}(\{x_1, \ldots, x_{k-2}\})^\Sigma - \alpha R^\Sigma \\
&> 1 - x_t + x_{k-1} - \alpha R^\Sigma && \text{using (3.21)} \\
&> 1 - x_t + \frac{x_k}{1-\alpha} + R^\Sigma - \frac{1}{(1-\alpha)(2+\alpha)} - \alpha R^\Sigma && \text{using (3.19)} \\
&= 1 - x_t + \frac{x_k}{1-\alpha} + (1-\alpha)R^\Sigma - \frac{1}{(1-\alpha)(2+\alpha)} \\
&\geq 1 - x_t + \frac{\alpha x_k}{1-\alpha} + \frac{1}{2+\alpha} - \frac{1}{(1-\alpha)(2+\alpha)} && \text{using (3.20)} \\
&= \frac{1}{2+\alpha} + \frac{1-\alpha-\alpha^2}{(1-\alpha)(2+\alpha)} - x_t + \frac{\alpha x_k}{1-\alpha} \\
&\geq \frac{1}{2+\alpha} + \frac{1-\alpha-\alpha^2}{(1-\alpha)(2+\alpha)} + \frac{\alpha-(1-\alpha)}{1-\alpha} x_k && \text{using } x_k \geq x_t \\
&= \frac{1}{2+\alpha} + \frac{1-\alpha-\alpha^2}{(1-\alpha)(2+\alpha)} + \frac{2\alpha-1}{1-\alpha} x_k \\
&\geq \frac{1}{2+\alpha} \text{ ,}
\end{aligned}$$

where the last step is trivially true for any $\alpha \geq 1/2$. Thus we get the desired competitive ratio in the considered range for α.

This proves the induction step, and thus the upper bound on the competitive ratio. □

3.4.3 Large Reservation Costs

For the last remaining values of α in the range between $\phi - 1$ and 1, we will use Algorithm 5, which is identical to Algorithm 4 up to the packing condition. It

Algorithm 5 $\frac{1}{1-\alpha}$-competitive Algorithm for $\phi - 1 \leq \alpha < 1$.

```
for k = 1, ..., n do
    if x_k + (1 − α)R^Σ ≥ 1 − α then
        Pack 🔓R ∪ x_k. END
    else
        Reserve x_k;
Pack 🔓R.
```

should thus be no surprise that the analysis of this algorithm will be very similar to that of Algorithm 4.

We start by bounding the size of the reservation buffer that this algorithm uses.

Lemma 3.4.6. *During any run of Algorithm 5 it always holds that* $R^\Sigma < 1$.

Proof. For any j, let R_j denote the reservation buffer of the algorithm after the items $x_1, \ldots, x_j$ have been presented. If for any k,

$$x_k + (1-\alpha)R_{k-1}^\Sigma < 1 - \alpha \ ,$$

then the algorithm reserves the item x_k and has a new reservation buffer size of $R_k^\Sigma = x_k + R_{k-1}^\Sigma$. Since obviously $x_k \geq (1-\alpha)x_k$, we have

$$(1-\alpha)R_k = (1-\alpha)x_k + (1-\alpha)R_{k-1} \leq x_k + (1-\alpha)R_{k-1} < 1 - \alpha \ .$$

Thus, $R_k^\Sigma < 1$. □

Analogously to Lemma 3.4.3, we will next introduce a lemma that allows us to reorder "bad" instances.

Lemma 3.4.7. *Let* $I = x_1, \ldots, x_n$ *be an instance for Algorithm 4, that triggers it to pack some item* x_k *and assume that the ratio between the gain of an optimal solution and Algorithm 4 was worse than* $\frac{1}{1-\alpha}$ *on* I*. Then there is another instance* $I' = x_{i_1}, x_{i_2}, x_{i_3}, \ldots, x_{i_{k-1}}, x_k, \ldots, x_n$ *where* $(i_1, i_2, \ldots, i_{k-1})$ *is a permutation of* $(1, \ldots, k-1)$ *and* $x_{i_1} \geq x_{i_2} \geq \ldots \geq x_{i_{k-1}}$*, and the ratio between the gain of the optimal solution and the algorithm is also larger than* $\frac{1}{1-\alpha}$ *on* I'*.*

Proof. We show how to transform the original sequence such that the number of inversions is reduced by one. Repeating this transformation yields the desired result.

Let $I = x_1, \ldots, x_n$ be an instance for Algorithm 5, such that packing is triggered by x_k with a competitive ratio larger than $\frac{1}{1-\alpha}$. Assume that $x_i < x_{i+1}$ for some $i < k-1$. We prove that we will get an instance with a competitive ratio larger

than $\frac{1}{1-\alpha}$ again if we swap the positions of x_i and x_{i+1}. It is easy to see that this transformation preserves the sum of the first $k-1$ items. To show that it is a sequence that does not trigger the packing after the ith or $(i+1)$st item, we have to verify that

$$x_{i+1} < (1-\alpha) - (1-\alpha)(x_1 + \cdots + x_{i-1}) \tag{3.22}$$

and

$$x_i < (1-\alpha) - (1-\alpha)(x_1 + \cdots + x_{i-1} + x_{i+1}) \,. \tag{3.23}$$

Here, (3.22) is straightforward, since x_k for $k > i+1$ triggering the packing in the unmodified instance implies $x_{i+1} < (1-\alpha) - (1-\alpha)(x_1 + \ldots + x_i)$. We prove (3.23) by means of a contradiction. Assume, that

$$x_i \geq (1-\alpha) - (1-\alpha)(x_1 + \ldots + x_{i-1} + x_{i+1}) \,. \tag{3.24}$$

For $1 \leq j \leq i+1$, we also know that

$$x_j \leq (1-\alpha) - (1-\alpha)(x_1 - \ldots - x_{j-1}) \,, \tag{3.25}$$

and by construction that

$$x_i < x_{i+1} \,. \tag{3.26}$$

If we define $z = (1-\alpha) - (1-\alpha)(x_1 + \ldots + x_{i-1})$, then we can rewrite (3.24), (3.25) for $j = i+1$, and (3.26) as

$$\begin{aligned} x_i &\geq z - (1-\alpha)x_{i+1}, \\ x_{i+1} &\leq z - (1-\alpha)x_i, \\ x_i &< x_{i+1} \,. \end{aligned}$$

As in the proof of Lemma 3.4.3, we can solve this system of equations and conclude that the first two conditions are only satisfiable if $x_{i+1} \leq x_i$, which gives us the desired contradiction.

Thus, the new sequence has a competitive ratio larger than $\frac{1}{1-\alpha}$. Again, although the order of $x_1, \ldots, x_{k-1}$ has been changed, the same reserved requests trigger a packing with x_k with the same reservation size, just as before the transformation. Hence, if there was no way to pack the items well enough before the transformation, it also cannot be done afterwards. □

With this, we can give the upper bound proof for this last segment for values of α.

Theorem 3.4.4. *Given a fixed α with $\phi - 1 < \alpha < 1$. Algorithm 5 solves the* Oskr *problem with a competitive ratio of at most $\frac{1}{1-\alpha}$.*

Proof. Let us first assume that we run Algorithm 4 on an instance $x_1, \ldots, x_n$, and no element triggers a packing. This means that all elements are reserved. By Lemma 3.4.6 we know that the size of the reservation buffer never exceeds the capacity of the knapsack. This means that both the optimal solution as well as the algorithm pack all offered elements. Since the algorithm pays reservation costs for its fill of the knapsack, it achieves a competitive ratio of

$$\frac{\sum_{i=1}^{n} x_k}{\sum_{i=1}^{n} x_k - \alpha \sum_{i=1}^{n} x_k} = \frac{1}{1-\alpha} \ .$$

Thus, an instance $x_1, \ldots, x_n$ triggers a packing, at some item x_k. We do an induction on the value of the index k. If $k = 1$, the first item offered triggers the packing, thus it holds that $x_1 \geq 1 - \alpha$ and the gain of the algorithm is at least $1 - \alpha$.

For the induction step, we proceed similarly to the proof of Theorem 3.4.3. Let us assume, using Lemma 3.4.7, that we have an instance $x_1 \geq x_2 \geq \ldots \geq x_{k-1}, x_k, \ldots x_n$ where the algorithm packs after being presented x_k. We distinguish two cases.

In the first case, we assume that x_{k-1} is not part of an optimal packing, i.e.

$$🖫(\{x_1, \ldots, x_{k-2}\})^{\Sigma} + x_{k-1} > 1 \ . \tag{3.27}$$

Since x_{k-1} did not trigger the packing, we know that

$$x_{k-1} + (1-\alpha) \sum_{j<k-1} x_j < 1 - \alpha \ , \tag{3.28}$$

and thus especially that $x_{k-1} < 1 - \alpha$. Thus, the gain that Algorithm 4 achieves is

$$\begin{aligned}
&🖫(\{x_1, \ldots, x_{k-2}\})^{\Sigma} - \alpha R^{\Sigma} \\
&\geq 1 - x_{k-1} - \alpha \sum_{j \leq k-1} x_j && \text{using (3.27)} \\
&= 1 - (1+\alpha) x_{k-1} - \alpha \sum_{j<k-1} x_j \\
&> 1 - (1+\alpha) x_{k-1} - \alpha \left(1 - \frac{x_{k-1}}{1-\alpha}\right) && \text{using (3.28)} \\
&= 1 - \alpha + \left(\frac{\alpha}{1-\alpha} - (1+\alpha)\right) x_{k-1} \\
&\geq 1 - \alpha + \frac{-1+\alpha+\alpha^2}{1-\alpha} x_{k-1} \\
&\geq 1 - \alpha && \text{using } \alpha \geq \phi - 1 \ .
\end{aligned}$$

In the second case, we assume that x_{k-1} is part of an optimal packing. For this, we distinguish two sub-cases.

1. Taking x_{k-1} out of the request sequence still triggers the packing. This means that $x_k+(1-\alpha)(R^\Sigma-x_{k-1}) \geq 1-\alpha$ holds. We can thus consider the sequence $x_1, \ldots, x_{k-2}, x_k$. This sequence has $k-1$ items before the packing is triggered, and x_{k-1} cannot be part of its optimal solution, thus its competitive ratio is at most $\frac{1}{1-\alpha}$ by the induction hypothesis, and the competitive ratio after adding x_{k-1} can not be worse.

2. Taking x_{k-1} out of the request sequence does not trigger the packing.

   This means that

   $$x_k + (1-\alpha)(R^\Sigma - x_{k-1}) < 1-\alpha \;, \tag{3.29}$$

   but also that

   $$x_k + (1-\alpha)R^\Sigma \geq 1-\alpha \;, \tag{3.30}$$

   and, if we let x_j be the smallest element that does not get packed, $x_j \geq x_{k-1}$ holds. Because of the optimality of the packing, $x_k \geq x_j$ also holds, as otherwise one can take all of the reserved elements as the packing and obtain a better bound.

   It further holds that

   $$\text{🖬}(\{x_1, \ldots, x_{k-2}\})^\Sigma - x_{k-1} + x_j > 1 \;. \tag{3.31}$$

   Finally, we can bound x_j by $x_j \leq x_1 \leq 1-\alpha$.

   $$\begin{aligned}
   &\text{🖬}(\{x_1, \ldots, x_{k-2}\})^\Sigma - \alpha R^\Sigma \\
   &\geq 1 - x_j + x_{k-1} - \alpha R^\Sigma && \text{using (3.31)} \\
   &\geq 1 - x_j + \frac{x_k}{1-\alpha} + R^\Sigma - 1 - \alpha R^\Sigma && \text{using (3.29)} \\
   &= -x_j + \frac{\alpha}{1-\alpha} x_k + x_k + (1-\alpha)R^\Sigma \\
   &\geq -x_j + \frac{\alpha}{1-\alpha} x_k + (1-\alpha) && \text{using (3.30)} \\
   &= 1 - \alpha - x_j + \frac{\alpha}{1-\alpha} x_k \\
   &\geq 1 - \alpha - x_j + \frac{\alpha}{1-\alpha} x_j && x_k \geq x_j \\
   &= 1 - \alpha + \frac{2\alpha - 1}{1-\alpha} x_j \\
   &\geq 1 - \alpha
   \end{aligned}$$

   as we wanted.

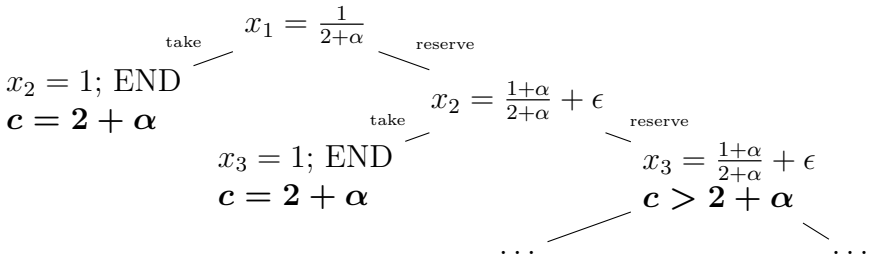


Figure 3.5: Sketch of the adversarial strategy that is used in the proof of Theorem 3.5.1. Upon continued reservation, a new item of the same size is presented repeatedly.

This proves the induction step. □

3.5 Nonrejecting Algorithms

When defining the Oskr problem, we stated that an algorithm always has three options when given a new item of the instance: to pack it, given that there is enough space in the knapsack left, to reserve it for a proportional fee or to reject it altogether. In this section, we will look at the class of *nonrejecting* algorithms for the Oskr problem which forfeit the option to reject an item.

A seemingly plausible intuition for this class of algorithms might be the following: If the cost of reservation is very small, rejecting an item should not be necessary, as even when an item cannot be packed, the cost of reserving it is negligible. On the other hand, when the reservation cost is rising, aggressively reserving items may seem like a very bad strategy, as the risk of not being able to utilize reserved items may come to mind. Interestingly, both of these intuitions turn out to be wrong. To show this, we will which we will show by first giving a lower bound for nonrejecting algorithms that exceeds the upper bound of a rejecting algorithm for small α and that tightly matches the upper bound of a nonrejecting algorithm for bigger α.

We first provide a lower bound for algorithms that are unable to reject items.

Theorem 3.5.1. *Given a fixed α with $0 < \alpha < 1$. There exists no nonrejecting algorithm solving the* Oskr *problem with a competitive ratio better than $2 + \alpha$.*

Proof. Let $\epsilon > 0$. Consider the following set of adversarial instances depicted in Figure 3.5. First the adversary presents an item of size $\frac{1}{2} - \frac{\alpha}{4+2\alpha} = \frac{1}{2+\alpha}$. If an algorithm takes this item the adversary will present an item of size 1, and the algorithm will have a competitive ratio of $2+\alpha$ as claimed. If an algorithm reserves this first item, the adversary will present a second item of size $\frac{1}{2}+\frac{\alpha}{4+2\alpha}+\epsilon = \frac{1+\alpha}{2+\alpha}+\epsilon$. As both items do not fit into the knapsack together, an algorithm can decide to

either take the larger of the two items or to reserve this second item as well. If it takes the larger item, the adversary will again present an item of size one. An algorithm can thus achieve a gain of

$$\frac{1+\alpha}{2+\alpha} + \epsilon - \frac{\alpha}{2+\alpha} = \frac{1}{2+\alpha} + \epsilon$$

which results in the claimed competitive ratio. Finally, whenever an algorithm decides to reserve an item, from this point onward, another item of size $\frac{1+\alpha}{2+\alpha} + \epsilon$ is presented. At any point at which the algorithm decides to take a presented item, an item of size 1 is presented afterwards. As there are no two items in the instance that fit into the knapsack together, the gain of any deterministic nonrejecting algorithm will strictly decrease with every reservation. □

Looking at the upper bound given in Theorem 3.4.1, we see that an algorithm that is unable to reject items performs quite a bit worse than one that is able to reject items, such as Algorithm 3. Thus, an algorithm needs to be able to reject items to become 2-competitive for small values of α. On the other hand, the lower bound provided in Theorem 3.5.1 matches the upper bound of Theorems 3.4.2 and 3.4.3, which are based on the nonrejecting Algorithm 4.

Similarly, by the same proof as for Theorem 3.5.1 with the first item being of size $1 - \alpha$ and all subsequent ones being of size $1 - \alpha + \alpha(1 - \alpha)$, the following statement follows.

Corollary 3.5.1. *Given a fixed α with $0 < \alpha < 1$. There exists no nonrejecting algorithm solving the* Oskr *problem with a competitive ratio better than $\frac{1}{1-\alpha}$.*

Thus, for $\alpha > \sqrt{2}-1$, nonrejecting algorithms are the best possible algorithms.

3.6 Reservation with Reimbursements

So far, we have assumed that any reservation costs paid are final. In many real-world scenarios this is however not the case. In the case of down payments, the party buying a good or service is commonly reimbursed their payment if the good or service is actually bought.

If we want to model this scenario, we only have to adjust the gain function of an online algorithm that was introduced in Section 3.1.

Again, at step $n + 1$, an algorithm is informed that the request sequence is finished. It may then select a subset $R' \subseteq R$ and set $K := K \cup R'$, if $\sum_{x_k \in K} x_k + \sum_{x_r \in R'} x_r \leq 1$. The Online Simple Knapsack with Reservations problem

(OSKR) with reimbursements is then for an algorithm ALG to maximize the sum of items in K, minus the *reservation costs*

$$gain_{\texttt{ALG}}(I) := \sum_{x_k \in K} x_k - \alpha \sum_{x_r \in R \setminus R'} x_r \,.$$

Now that we have formally introduced this slightly different model, we will analyze how the competitive ratios depending on the value of α differ from the classical model. We will see that the competitive ratios behave differently from before. While for values $0 < \alpha \leq 0.5$, the competitive ratio stays at a constant value of 2, it from there on rises with a function of $\frac{1}{2}(\sqrt{\alpha^2 + 2\alpha + 5} + \alpha + 1)$. Noticeably, there is no longer a sharp cut at $\alpha = 1$, as the model does not "collapse" to the classical model for any finite value of α. As before, we start with the lower bounds.

3.6.1 Lower Bounds

The family of instances for the lower bound of 2 and values of α in $0 < \alpha \leq 0.5$ is identical to the lower bound proof of Theorem 3.3.1 and the proof works by the same scheme. As the algorithm can never fill its knapsack beyond a level of 0.5 (up to some ϵ term), it does not make a difference if any reimbursements happen. Therefore, we give the following result as a corollary.

Corollary 3.6.1. *Given a fixed α with $\alpha > 0$. There exists no algorithm solving the* OSKR *problem with reimbursements with a competitive ratio better than* 2*.*

The other lower bound is again based on a generic adversarial strategy of three items, similar to Theorem 3.3.2.

Theorem 3.6.1. *Given a fixed α with $0.5 < \alpha$. There exists no algorithm solving the* OSKR *problem with reimbursements with a competitive ratio better than $\frac{1}{2}(\sqrt{\alpha^2 + 2\alpha + 5} + \alpha + 1)$.*

Proof. We give a set of adversarial strategies that use at most three items $I = \{s, t, u\}$, where we impose the restriction that $s + t > 1$. We will set these items to specific sizes later and first discuss if and when they are presented. Let $s \leq 0.5$ be the item that is presented first. As long as there are items left, the adversary will act as follows, depending on the behavior of an algorithm:

Case 1: An algorithm packs *s*:

The next item is $t = 1$, after which no further items are presented. The competitive ratio is then $\frac{1}{s}$.

Case 2: An algorithm reserves *s*:

The adversary presents the next item t with $t > s$.

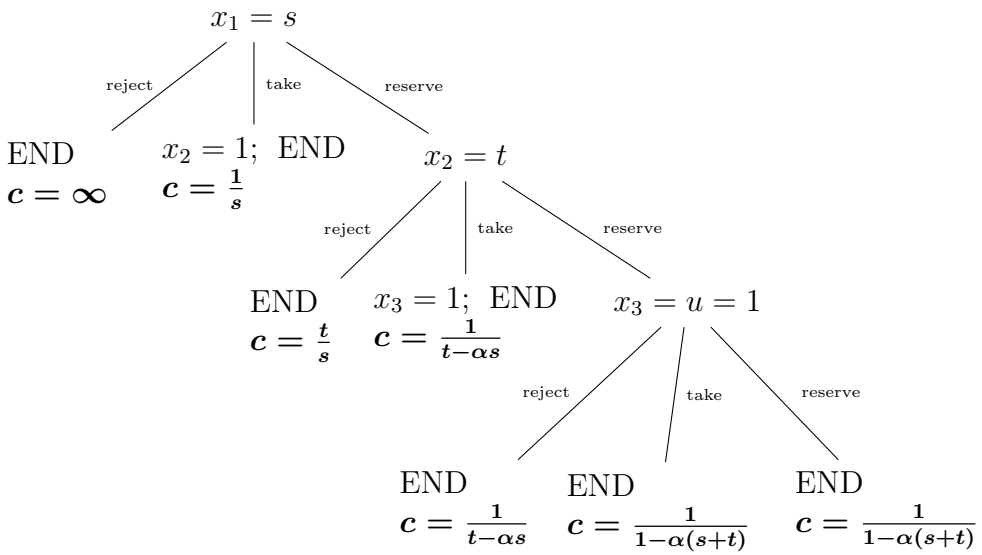


Figure 3.6: Generic adversarial strategy with up to three items, where $s < t < u = 1$ and $s + t > 1$.

Case 2.1: An algorithm packs t:

The next item is $u = 1$, after which no further items are presented. The competitive ratio is then $\frac{1}{t-\alpha s}$.

Case 2.2: An algorithm reserves t:

The adversary presents the next item $u = 1$, after which no further items are presented.

Case 2.2.1: An algorithm packs u:

The competitive ratio is then $\frac{1}{1-\alpha(s+t)}$.

Case 2.2.2: An algorithm reserves u:

The competitive ratio is then $\frac{1}{1-\alpha(s+t)}$, as the algorithm gets reimbursed for u.

Case 2.2.3: An algorithm rejects u:

The competitive ratio is then $\frac{1}{t-\alpha(s+t)}$.

Case 2.3: An algorithm rejects t:

No further items are presented. The competitive ratio is then $\frac{t}{s}$, as the algorithm gets reimbursed for s.

Case 3: An algorithm rejects s:

No further items are presented. As the knapsack of the algorithm is empty, it is not competitive.

This strategy is also depicted in Figure 3.6. We choose

$$s = \frac{3 + \alpha - \sqrt{5 + 2\alpha + \alpha^2}}{2\alpha + 2} + \varepsilon,\ t = \frac{1}{2}(\sqrt{s}\sqrt{\alpha^2 s + 4} + \alpha s), \text{ and } u = 1,$$

Algorithm 6 $\max\{2, \frac{1}{2}(\sqrt{\alpha^2+2\alpha+5}+\alpha+1)\}$-competitive Algorithm for $\alpha > 0$.

1: Let $\mu := 1/(c(1-\alpha))$
2: **for** $k = 1, \ldots, n$ **do**
3: **if** $x_k + R^\Sigma < \frac{1}{c}$ **then Reserve** x_k.
4: **else if** $x_k + R^\Sigma \leq 1$ **then Pack** $\mathfrak{F}(R \cup x_k)$. END
5: **else if** $\exists R' \subseteq R.\mathfrak{F}(R' \cup x_k)^\Sigma - \alpha(R \setminus R')^\Sigma \geq \frac{1}{c}$ **then**
6: **Pack** $\mathfrak{F}(R' \cup x_k)$. END
7: **else Reject** x_k.
8: **Pack** $\mathfrak{F}(R)$.

for some $\varepsilon > 0$ and next verify that $c \geq \frac{1}{2}(\sqrt{\alpha^2+2\alpha+5}+\alpha+1)$ for all $\alpha > 0.5$. We already know the competitive ratios of each possible outcome due to the previous case distinction, which are

$$c \geq \min(\infty, \frac{1}{s}, \frac{t}{s}, \frac{1}{t-\alpha s}, \frac{1}{1-\alpha(s+t)}) = \min(\frac{t}{s}, \frac{1}{t-\alpha s}, \frac{1}{1-\alpha(s+t)}) \, .$$

We now simply substitute the mentioned values for s and t and prove that these choices yield the wanted lower bound. Using standard calculus, the first two terms $\frac{t}{s}$ and $\frac{1}{t-\alpha s}$ become exactly $\frac{1}{2}(\sqrt{\alpha^2+2\alpha+5}+\alpha+1)$.

The third term $\frac{1}{1-\alpha(s+t)}$ has to be treated a bit more carefully. By simply plugging in s and t one can easily verify that the resulting function is larger than $\frac{1}{2}(\sqrt{\alpha^2+2\alpha+5}+\alpha+1)$ for all values of $\alpha \in (0.5, 1)$. Note that the gain of any algorithm then becomes negative, as $s + t > 1$. Thus, this branch even results in an unbounded competitive ratio for all $\alpha \geq 1$. □

3.6.2 Upper Bounds

The algorithm that we use to match the lower bounds of the previous section is a simplified version of Algorithm 3, which we first saw in Section 3.4.

The proof is thus in most parts identical to that of Theorem 3.4.1.

Theorem 3.6.2. *Given a fixed α with $\alpha > 0$. Algorithm 6 solves the* Oskr *problem with reimbursements with a competitive ratio of* $\max\{2, \frac{1}{2}(\sqrt{\alpha^2+2\alpha+5}+\alpha+1)\}$.

Proof. We first deal with the simple cases. Only in line 3 can a reservation be made, if one criterion is met: If the gain from the current item plus the gain of everything reserved so far is still below the wanted gain, the item is reserved. This especially means that the current item and the gain from the reserve itself are both smaller than $1/c$ if this condition triggers.

If an item x_k is *not* reserved, we thus know that this item together with the pure gain of all reserved items sum up to the wanted size, i.e. to $\frac{1}{c}$. It can however be the case that such a packing is invalid by exceeding the size of the knapsack. Thus, the algorithm is either faced with a valid packing and trivially reaches the wanted competitive ratio in line 4, or it continues otherwise.

The algorithm then simply calculates in line 6 if the algorithm is able to achieve the wanted competitive ratio using the new item x_k anyway and reject the item in line 6 otherwise.

We are thus left with the most involved case, which is the algorithm not triggering any of the previous stopping conditions and instead ending in line 8. As any item that is rejected by the algorithm violated the condition in line 3, such an item has to be of size at least $\frac{1}{2}$. This is the case as together with the items from the reservation buffer, $R^\Sigma < \frac{1}{c} \leq \frac{1}{2}$, their sum had to exceed 1 due to line 4. Thus, the optimal can contain at most one such large item that the algorithm rejects. Any items that are of size at most $\frac{1}{2}$ are shared by both the optimal solution and the solution of the algorithm.

We fix an optimal solution and let $x_{\texttt{OPT}}$ be the unique item that the algorithm rejects but that is part of the optimal solution. Let $x_{\texttt{ALG}}$ be the largest reserved item of the algorithm at the time that $x_{\texttt{OPT}}$ is rejected. If the algorithm stops in line 8, it never reserved an item that is not part of its solution. This means its complete reservation costs are reimbursed in this case and the competitive ratio in this case is only dependent on the size of the packed knapsacks.

As the algorithm rejected $x_{\texttt{OPT}}$, we know that $x_{\texttt{OPT}} - \alpha x_{\texttt{ALG}} < \frac{1}{c}$. We also know that $x_{\texttt{OPT}} + x_{\texttt{ALG}} > 1$, since by line 4, the algorithm would have packed them together. The competitive ratio is only not reached if $c < \frac{(x_{\texttt{OPT}} \cup R)^\Sigma}{x_{\texttt{ALG}} \cup R^\Sigma} \leq \frac{x_{\texttt{OPT}} \cup R^\Sigma}{x_{\texttt{ALG}} \cup R^\Sigma}$ holds. We lead this assumption to a contradiction when choosing $c > \max\{2, \frac{1}{2}(\sqrt{\alpha^2 + 2\alpha + 5} + \alpha + 1)\}$.

From $\frac{x_{\texttt{OPT}} \cup R^\Sigma}{x_{\texttt{ALG}} \cup R^\Sigma} > c > 1$ we deduce that $\frac{x_{\texttt{OPT}} \cup R^\Sigma}{x_{\texttt{ALG}} \cup R^\Sigma} < \frac{x_{\texttt{OPT}}}{x_{\texttt{ALG}}}$ and especially that $\frac{x_{\texttt{OPT}}}{x_{\texttt{ALG}}} > c$. This means that $x_{\texttt{ALG}} < \frac{x_{\texttt{OPT}}}{c}$. We next upper bound the size of $x_{\texttt{OPT}}$.

$$
\begin{aligned}
& x_{\texttt{OPT}} - \alpha x_{\texttt{ALG}} < \frac{1}{c} && \text{Solve for } x_{\texttt{ALG}} \\
\Leftrightarrow\ & \frac{x_{\texttt{OPT}} - \frac{1}{c}}{\alpha} < x_{\texttt{ALG}} && \text{Use } x_{\texttt{ALG}} < \frac{x_{\texttt{OPT}}}{c} \\
\Rightarrow\ & \frac{c x_{\texttt{OPT}} - 1}{\alpha} < x_{\texttt{OPT}} && \text{Solve for } x_{\texttt{OPT}} \\
\Leftrightarrow\ & x_{\texttt{OPT}} < \frac{1}{c - \alpha}
\end{aligned}
$$

We use this to upper bound the competitive ratio as follows.

$$
\begin{aligned}
& x_{\texttt{OPT}} + x_{\texttt{ALG}} > 1 \text{ and } x_{\texttt{OPT}} - \alpha x_{\texttt{ALG}} < \frac{1}{c} && \text{Solve each for } x_{\texttt{OPT}} \\
\Leftrightarrow\ & 1 - x_{\texttt{ALG}} < x_{\texttt{OPT}} < \frac{1}{c} + \alpha x_{\texttt{ALG}} && \text{Add } x_{\texttt{ALG}} \text{ to both sides} \\
\Leftrightarrow\ & 1 < \frac{1}{c} + \alpha x_{\texttt{ALG}} && \text{Use } x_{\texttt{ALG}} < \frac{x_{\texttt{OPT}}}{c} \\
\Rightarrow\ & 1 < \frac{1}{c} + (1+\alpha)\frac{x_{\texttt{OPT}}}{c} && \text{Multiply both sides with } c \\
\Leftrightarrow\ & c < 1 + (1+\alpha) x_{\texttt{OPT}} && \text{Use } x_{\texttt{OPT}} < \frac{1}{c-\alpha} \\
\Leftrightarrow\ & c < 1 + \frac{1+\alpha}{c-\alpha}
\end{aligned}
$$

This last term leads to the wanted contradiction, since it only holds for $c < \frac{1}{2}(\sqrt{\alpha^2+2\alpha+5} + \alpha + 1)$ when $\alpha \geq 0.5$ and for $c < 2$ for $0 < \alpha \leq 0.5$. We however assumed that c is chosen larger than each of these terms.

Thus, the algorithm is always able to achieve the targeted competitive ratio. □

3.7 Open Problems

While we have completely analyzed the Oskr problem in two different reservation models, there are of course related problems open that would also be very reasonable to study. The reservation model allows for more modifications, such as paying only a price for the maximum size of the reservation buffer that was reached during a run of an algorithm. Such a model may motivate an algorithm to pack items from its reserve before the instance has ended. Another interesting variation would be to vary the reservation costs over the course of an instance.

Clearly, the more general Online Knapsack problem with reservations is still left open. In this case, one has to first decide what reservation costs should be paid for: the size of an item or its value.

Finally, combinations with other popular online knapsack variations could be another venue of research, such as predictions, randomization, advice complexity, multi-knapsack or preemption with recourse.

Chapter 4

The Online Simple Knapsack Problem with Bounded Predictions

The view of online algorithms having to navigate problems about which nothing is assumed to be known is quite pessimistic compared to the real world. Commonly, one can obtain a very good prediction of the data that is incoming, such as a supermarket chain re-stocking their stores based on past sales numbers, the season and weather, location of a store or current trends predicted by their marketing departments. Thus, while a supermarket cannot know the exact amount of items that will be sold in a given period, most of the time it can predict these sales withing a fixed margin of error.

In this chapter, we assume that an algorithm already has a good estimate on an upcoming instance. Specifically, an algorithm is tasked with solving the Online Simple Knapsack problem, given an estimate of the sizes of all upcoming items and a fixed deviation rate $\delta \in [0, 1]$, called the *distortion* of the algorithm. Naturally, this means that an algorithm knows the length of the input in advance.

4.1 Problem Definitions and Notation

As in Chapter 3, we will abuse notation by using x_i for both the label of an item and for the size (or gain) of the same item.

Definition 4.1.1 (The Online Simple Knapsack with Bounded Predictions Problem)**.** Given a constant $\delta \in [0, 1]$, called *distortion*. Given a set of items $I = (x_1, \ldots, x_n)$ as a request sequence of items that arrive sequentially. Let $P = (x_1^-, \ldots, x_n^-)$ be a sequence of items called the *prediction* such that $x_i \in [x_i^-, \min(x_i^+, 1)]$, where $x_i^+ := \frac{1+\delta}{1-\delta} x_i^-$. Let $w : I \to [0, 1] \subset \mathbf{Q}$ be a size

function and $g : I \to [0,1] \subset \mathbf{Q}$ with $w = g$ be a gain function. Let K be an initially empty sets called the *knapsack*. An algorithm is given P and δ before the first item of I is revealed. At each step $i \in \{1, \dots, n\}$, the item x_i of the request sequence is given. An algorithm then has one of the following options:

- **Pack** If $\sum_{x_k \in K} w(x_k) + w(x_i) \leq 1$, set $K := K \cup x_i$.
- **Reject** Do nothing.

The Online Simple Knapsack with Bounded Predictions problem (Oskp) is then for an algorithm $\texttt{ALG}$ to maximize the sum of items in K, i.e.

$$gain_{\texttt{ALG}}(I) := \sum_{x_k \in K} g(x_k)\,.$$

We denote the gain of an optimal algorithm by

$$gain_{\texttt{OPT}}(I) := \max_{K \subseteq I}\Big(\sum_{x_k \in K} g(x_k) \mid \sum_{x_k \in K} w(x_k) \leq 1\Big)\,.$$

As the instance will be always clear from the context, we will write $gain_{\texttt{ALG}}$ for $gain_{\texttt{ALG}}(I)$ and $gain_{\texttt{OPT}}$ for $gain_{\texttt{OPT}}(I)$. We will write $P_{[i,n]}$ to denote the infix of P from index i to n, including both endpoints. Since we look at the simple knapsack problem, we will write x for $w(x)$ and x for $g(x)$, since $w = g$.

When referring to an item x^-, we will sometimes speak of the *minimum (possible) size* of an item, as well as speaking of the *maximum (possible) size* of an item when referring to an item x^+. For every instance I with predictions P, we define $b^- := \max_{x^- \in P}(x^-)$ as the *announced largest item* of the instance. This is of course not necessarily the item of largest *actual* size. If b^- is not unique, we refer to the first of such items regarding the ordering of the prediction sequence.

Note that restricting the maximum *actual* size of items to 1 is necessary to ensure competitiveness, as otherwise the classical counterexample by Marchetti-Spaccamela and Vercellis [57] can be reconstructed by letting $P = (\varepsilon, 1)$ for some $\varepsilon > 0$ and setting the second item to a value larger than one if an algorithm rejects the first item.

Strictly speaking, any competitive ratio given in this chapter is a function of a fixed value δ. To simplify notation, we will write c for $c(\delta)$, as it is always either clear from the context or not of relevance which δ the competitive ratio refers to.

4.2 Related Work and Our Contributions

As we already gave an overview over variations of the knapsack problem in the previous chapter and talked about the model of predictions in Subsection 1.7,

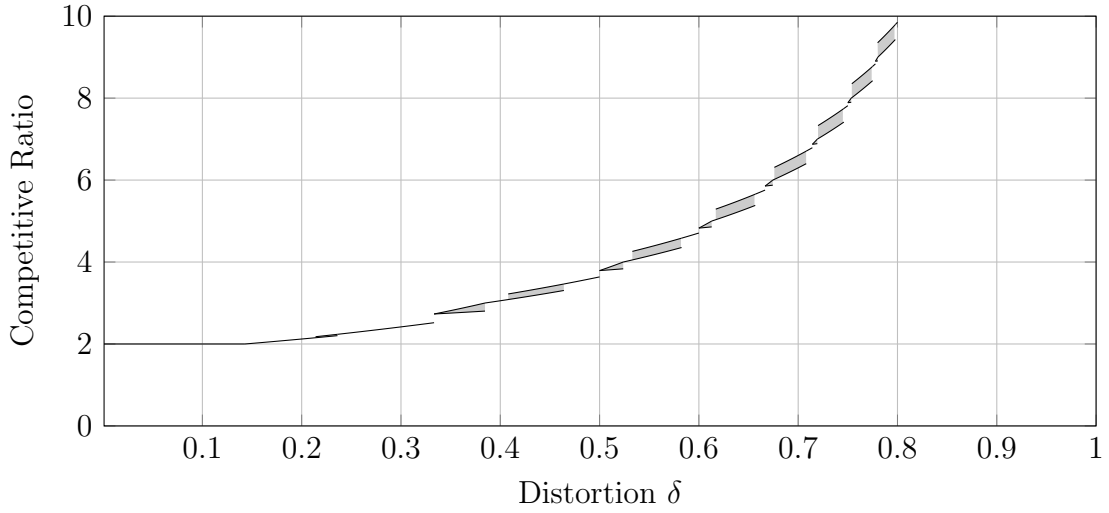


Figure 4.1: Competitive ratio of the OSKP problem, depending on the distortion δ. The gray areas signify a gap between the best known lower and upper bounds.

we focus here only on more closely related work. The *robust knapsack problem* by Monaci, Pferschy and Serafini [59] is very similar in that it also allow for an uncertain input with a multiplicative factor, but the model is concerned with *offline* algorithms that see the complete permuted instance at once and are compared to the performance of a non-perturbed instance. Im et al. [47] recently looked at the general knapsack problem, with a prediction only on the number of items of the instance. Angelopoulos, Kamali and Shadkami [1] looked at the online bin packing problem with predictions on the frequency of item sizes in the instance. Boyar, Favrholdt and Larsen [23] very recently studied the online simple knapsack with predictions, but working with predictions on the *average* size of the items an optimal solution would pack.

We study the behavior of the OSKP problem with a distortion between $0 < \delta < 1$. The difference between the predicted size and the actual size of an item is determined by a relative error.

While we are not able to give tight bounds on the competitive ratio for all values of δ between 0 and 1, we are able to carve out the following, partial picture, which is visualized in Figure 4.1, with the bounds that we prove also given in Table 4.1. Many calculations for this chapter have been supported or done through the use of computer algebra systems, namely `Maxima`[1] and `WolframAlpha`[2].

Up to $\delta \leq \frac{1}{7}$, we give a tight bound of 2 on the competitive ratio. From there on, three bounds on the competitive ratio repeat periodically, given a fixed $k \in \mathbf{N}$, thus

[1] `maxima.sourceforge.io/`
[2] `www.wolframalpha.com/`

Table 4.1: Competitive ratios of the Oskp problem, given a fixed $k \in \mathbf{N}$ and distortion δ.

Distortion	Competitive ratio: Lower bound
$0 < \delta \leq \frac{1}{7}$	2
$\frac{k^2+k-1}{k^2+3k+3} < \delta \leq \delta_k$	$\sqrt{(2+k)\frac{1+\delta}{1-\delta}}$
$\delta_k \leq \delta < -1-k+\sqrt{4k+k^2}$	$\sqrt{(2+k)\frac{1+\delta}{1-\delta}}$
$-1-k+\sqrt{4k+k^2} \leq \delta \leq \frac{k}{k+2}$	$\frac{(1+\delta)(\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$
$\frac{k}{k+2} < \delta < \frac{k^2+k-1}{k^2+3k+3}$	$\frac{1+k}{2} - \frac{\sqrt{(-1+\delta)(-5-k(2+k)+\delta(-1+k)(3+k))}}{-2+2\delta}$

$\delta_k := \frac{12k+8}{32^{\frac{2}{3}}(3\sqrt{3}\sqrt{4k^3+11k^2+28k+44}-9k-34)^{\frac{1}{3}}} + \frac{1}{3}2^{\frac{2}{3}}(3\sqrt{3}\sqrt{4k^3+11k^2+28k+44}-9k-34)^{\frac{1}{3}} + 5/3$

the competitive ratio not only depends on δ but also on k, which determines which repitition of each function we are studying. Between values of δ between $\frac{k^2+k-1}{k^2+3k+3}$ and $\delta_k := \frac{12k+8}{32^{\frac{2}{3}}(3\sqrt{3}\sqrt{4k^3+11k^2+28k+44}-9k-34)^{\frac{1}{3}}} + \frac{1}{3}2^{\frac{2}{3}}(3\sqrt{3}\sqrt{4k^3+11k^2+28k+44}-9k-34)^{\frac{1}{3}} + 5/3$, we are able to prove a tight bound of $\sqrt{(2+k)\frac{1+\delta}{1-\delta}}$. From δ_k to $-1-k+\sqrt{4k+k^2}$, we are able to prove a lower bound of $\sqrt{(2+k)\frac{1+\delta}{1-\delta}}$ and an upper bound of $\frac{(1+\delta)(\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$. For the penultimate segment of $-1-k+\sqrt{4k+k^2} \leq \delta \leq \frac{1}{k+2}$, we give a matching upper and lower bound of $\frac{(1+\delta)(\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$. The final segment, which connects to the first segment for the next higher value of k, we are only able to prove a lower bound of $\frac{1+k}{2} - \frac{\sqrt{(-1+\delta)(-5-k(2+k)+\delta(-1+k)(3+k))}}{-2+2\delta}$ with an upper bound of $\frac{1+\delta+\sqrt{5+2\delta-3\delta^2}}{2-2\delta}$. Noticeably, the function of the competitive ratio depending on δ is not continuous between segments two and three.

The remainder of this chapter is structured as follows: We analyze Oskp problem by first proving a number of helpful structural lemmata in Section 4.3, followed by upper bound proofs in Section 4.4 and lower bounds in Section 4.5. We conclude with a short discussion of open problems and possible future work in Section 4.6.

Table 4.1 (continued)

Competitive ratio	Reference	
Upper bound	Lower bound	Upper bound
2	Thm. 4.5.1	Thm. 4.4.1
$\sqrt{(2+k)\frac{1+\delta}{1-\delta}}$	Thm. 4.5.2	Thm. 4.4.2
$\frac{(1+\delta)(\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$	Thm. 4.5.2	Thm. 4.4.3
$\frac{(1+\delta)(\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$	Thm. 4.5.3	Thm. 4.4.3
$\frac{1+\delta+\sqrt{5+2\delta-3\delta^2}}{2-2\delta}$	Thm. 4.5.4	Thm. 4.4.4

4.3 Structural Observations

When trying to design algorithms that are supposed to achieve some given competitive ratio $c \geq 1$, there are many instances for which a solution can be found more or less trivially. Since these solutions have to be handled explicitly by almost every of our algorithms, we handle them such that we can refer to them in our algorithm designs without risking redundancy. We first start with four very simple cases, that nevertheless need to be explicitly handled by our algorithms.

The first lemma deals with instances of very small possible total size.

Lemma 4.3.1. *Any instance I of the* OSKP *problem with predictions P, such that $\sum_{x^- \in P} x^- \leq \frac{1-\delta}{1+\delta}$, can be solved optimally by packing all items greedily.*

Proof. Since $x^+ \leq \frac{1+\delta}{1-\delta} x^-$, it holds that

$$\sum_{x \in I} x \leq \sum_{x^- \in P} \frac{1+\delta}{1-\delta} x^- = \frac{1+\delta}{1-\delta} \sum_{x^- \in P} x^- \leq \frac{1+\delta}{1-\delta} \frac{1-\delta}{1+\delta} = 1 \, .$$

Thus, all items of the instance are guaranteed to fit into the knapsack. □

The second lemma explicitly deals with instances that contain a subset of items that is both guaranteed to fit together and of sufficient size.

Lemma 4.3.2. *Any instance I of the* OSKP *problem with predictions P such that $\exists S \subseteq P. \sum_{x^- \in S} x^- \in [\frac{1}{c}, \frac{1-\delta}{1+\delta}]$ can be solved with a competitive ratio of at most c by packing exactly S.*

Proof. By definition, the items of S are both guaranteed to fit together in the knapsack and guaranteed to be of total size at least $\frac{1}{c}$. □

The third lemma ensures that we will only need to deal with instances in which no single item ensures the wanted competitive ratio.

Lemma 4.3.3. *Any instance I of the* Oskp *problem with predictions P such that $x^- \in P$ with $x^- \geq \frac{1}{c}$ can be solved with a competitive ratio of at least $c1$ by packing exactly x.*

The fourth lemma is slightly more interesting. While the previous statement upper bounded the size of the announced largest item b^- by $\frac{1}{c}$, we are also able to lower bound the size of this item as follows.

Lemma 4.3.4. *Any instance I of the* Oskp *problem with predictions P such that $b^- \leq (1 - \frac{1}{c})/\frac{1+\delta}{1-\delta}$ admits a c-competitive, greedy algorithm with c.*

Proof. Since b^- is by definition the announced largest item of the instance, we know that the size of the largest actual item is upper bounded by $\frac{1+\delta}{1-\delta}b^-$. Thus, the largest item that a greedy algorithm can *not* fit into its knapsack is of size at most $\frac{1+\delta}{1-\delta}b^- \leq \frac{1+\delta}{1-\delta}(1 - \frac{1}{c})/\frac{1+\delta}{1-\delta} = 1 - \frac{1}{c}$. If such an item does not fit into the knapsack, then the knapsack is already packed to size at least $\frac{1}{c}$. □

Thus, we can assume that $b^- \in [(1 - \frac{1}{c})/\frac{1+\delta}{1-\delta}, \frac{1}{c}]$, which allows us to partition the predicted items into two categories, which we will call *small* and *large*. We call an item small if its announced size together with b^- does not guarantee a packing of sufficient size, assuming they are presented as small as possible. Formally, an item $x^- \in P$ is small iff $x^- + b^- < \frac{1}{c} \Leftrightarrow x^- < \frac{1}{c} - b^-$. On the other hand, we call an item large if, together with b^-, it may be the case that the two items do not fit together in the knapsack (or if the sum of their announced sizes already exceeds 1.) Formally, an item $x^- \in P$ is large iff $\frac{1+\delta}{1-\delta}x^- + \frac{1+\delta}{1-\delta}b^- > 1 \Leftrightarrow x^- > \frac{1-\delta}{1+\delta} - b^-$.

Note that the existence of any item that falls in neither category already implies a trivial solution of size at least $\frac{1}{c}$ by Lemma 4.3.2, with the exception of b^- itself. However, we can see that b^- exceeds the minimum size of a large item, as

$$b^- > \frac{1-\delta}{1+\delta} - b^- \Leftrightarrow 2(1 - \frac{1}{c})/\frac{1+\delta}{1-\delta} > \frac{1-\delta}{1+\delta}$$

holds true for all δ iff $c > 2$ by substituting b^- by its minimum possible value, which will be the case for all bounds that we prove in this work. Thus, the size of b^- is always large enough to call it a large item as well.

This partition has some very useful implications, with the very central one being that an algorithm can essentially ignore small items of the instance. To show this, we first start by showing that the total sum of small items in P is limited or the instance has a trivial solution.

Lemma 4.3.5. *Let I be an instance of the* Oskp *problem with predictions P and $S := \{x^- \in P \mid x^- < \frac{1}{c} - b^-\}$ for $c \geq \frac{\delta^2+3}{2-2\delta}$. If P does not admit a solution due to Lemmata 4.3.1, 4.3.2, 4.3.3 or 4.3.4 and if $\sum_{x^- \in S} x^- \geq \frac{1}{c} - b^-$ holds, then there exists an algorithm that is c-competitive.*

Proof. The algorithm works as follows: It first selects an arbitrary subset of small items $S \subseteq P$ such that $\sum_{x^- \in S} x^- \in [\frac{1}{c} - b^-, 2(\frac{1}{c} - b^-)]$. Such a subset always exists — and can be found greedily — since each individual small item is announced smaller than $\frac{1}{c} - b^-$.

The algorithm packs b^- when it is revealed and small items from S greedily, but stopping to pack further small items as soon as their total size is at least $\frac{1}{c} - b^-$. Together with b^-, the gain is obviously at least $\frac{1}{c}$, so the only thing left to show is that such a subset is always guaranteed to fit in the knapsack and does not exceed its capacity.

In the worst case, the algorithm packs a small item of size $\frac{1}{c} - b^- - \varepsilon_1$ and is then presented an item of size $\frac{1+\delta}{1-\delta}(\frac{1}{c} - b^- - \varepsilon_2)$ for some $\varepsilon_1, \varepsilon_2 > 0$. Afterwards, b^- is presented of maximum possible size $\frac{1+\delta}{1-\delta} b^-$. Yet even in this case, all items fit into the knapsack, as

$$\begin{aligned}
&\frac{1}{c} - b^- - \varepsilon_1 + \frac{1+\delta}{1-\delta}(\frac{1}{c} - b^- - \varepsilon_2) + \frac{1+\delta}{1-\delta} b^- \\
&< \frac{1}{c} - b^- + \frac{1+\delta}{1-\delta}(\frac{1}{c} - b^-) + \frac{1+\delta}{1-\delta} b^- \\
&= \frac{1}{c} - b^- \frac{1+\delta}{1-\delta} \frac{1}{c} \\
&= (1 + \frac{1+\delta}{1-\delta})\frac{1}{c} - b^- \\
&< (1 + \frac{1+\delta}{1-\delta})\frac{1}{c} - (1 - \frac{1}{c})/\frac{1+\delta}{1-\delta}
\end{aligned}$$

Solving $(1 + \frac{1+\delta}{1-\delta})\frac{1}{c} - (1 - \frac{1}{c})/\frac{1+\delta}{1-\delta} \leq 1$ for c yields that these items fit if $c \geq \frac{\delta^2+3}{2-2\delta}$, which is lower than every upper bound proven in this work for all δ. □

Thus, we can assume that the *sum* of all announced small items is smaller than $\frac{1}{c} - b^-$, or else a trivial solution exists. This in turn lets us show a nice relation between small and large items, which is that the larger the sum of small items is, the smaller any large item may be announced or there is again a trivial solution.

Lemma 4.3.6. *Let I be an instance of the* Oskp *problem with predictions P and $S := \{x^- \in P \mid x^- < \frac{1}{c} - b^-\}$ with $c \geq 1$. If P does not admit a solution due to Lemmata 4.3.1, 4.3.2, 4.3.3 4.3.4 or 4.3.5 and $b^- \geq \frac{1}{c} - \sum_{x^- \in S} x^-$, then there exists an algorithm that is c-competitive.*

Proof. The algorithm packs b^- together with the items from S. Since it holds that $\sum_{x^- \in S} x^- < \frac{1}{c} - b^-$ due to Lemma 4.3.5, the total size is at least $b^- + \sum_{x^- \in S} x^- \geq \frac{1}{c} - \sum_{x^- \in S} x^- + \sum_{x^- \in S} x^- = \frac{1}{c}$. □

This lemma has the pleasant consequence that we will not have to worry about small items in most of our algorithm design beyond checking whether they are part of a trivial solution. Since in the upcoming analysis, we often bound the packed items of our algorithm against the largest item of the instance, it makes no difference to us whether there are no small items in the instance and thus a very large b^-, or if b^- is made smaller for the sake of additional small items in the instance.

The next lemma allows us to lower bound the announced size of large items depending on the size of b^-.

Lemma 4.3.7. *Let I be an instance of the* Oskp *problem with predictions P and $x^- \in P$ be a large item such that $x^- \neq b^-$. If $x + \frac{1+\delta}{1-\delta} b^- \leq 1$, packing x together with b guarantees a gain of $\frac{1}{c}$ with $c \geq 1$.*

Proof. A large item has actual size at least $\frac{1-\delta}{1+\delta} - b^-$, thus $x + \frac{1+\delta}{1-\delta} b^- > \frac{1-\delta}{1+\delta} - b^- + b^- = \frac{1-\delta}{1+\delta} > \frac{1}{c}$. □

This means that whenever an algorithm is guaranteed to be able to pack any large item together with b^-, its gain is sufficient.

Using these observations, we define with Algorithm 7 a small subroutine that will be called by our other algorithms in order to rule out trivial solutions to an instance.

Algorithm 7 Subroutine *filter_trivial*

if $\sum_{x^- \in P} x^+ \leq 1$ **then** ▷ Lemma 4.3.1
    Pack b. END
if $b^- \geq \frac{1}{c}$ **then** ▷ Lemma 4.3.3
    Pack b. END
if $b^- \leq \frac{1}{c} / \frac{1+\delta}{1-\delta}$ **then** ▷ Lemma 4.3.4
    Greedily **pack** items. END
if $\exists S \in 2^P . \sum_{x^- \in S} x^- \in [\frac{1}{c}, \frac{1-\delta}{1+\delta}]$ **then** ▷ Lemma 4.3.2
    Pack all items of S. END
$T := \{x^- \in P \mid x^- < \frac{1}{c} - b^-\}$
if $\sum_{x^- \in T} x^- \geq \frac{1}{c} - b^-$ **then** ▷ Lemma 4.3.5
    Pack a subset $R \in T$ with $\sum_{x^- \in R} x^- \in [\frac{1}{c} - b^-, 2(\frac{1}{c} - b^-)]$ and b. END
if $b^- \geq \frac{1}{c} - \sum_{x^- \in T} x^-$ **then** ▷ Lemma 4.3.6
    Pack all items of T and b. END

4.4 Upper Bounds

The analysis of upper bound algorithms remains complicated, even with the filters of Subsection 4.3. While we are quite confident that the lower bounds of Subsection 4.5 should not be improvable, finding matching upper bounds is an aim that we can only fulfill for parts of the whole range of values for δ.

We start with a simple 2-competitive algorithm for all values of δ up to $\frac{1}{7}$.

Algorithm 8 2-competitive Algorithm for $0 < \delta \leq \frac{1}{7}$.

1: *filter_trivial()*
2: Reveal items up to and including the first *large* item x_1.
3: **if** $x_1 \leq 1 - \frac{1+\delta}{1-\delta}b^-$ and $x_1 \neq b$ **then**
4: **Pack** x_1 with b. END
5: **else**
6: Greedily **pack** large items, including x_1.

Theorem 4.4.1. *Given a fixed δ with $0 < \delta \leq \frac{1}{7}$. Algorithm 8 solves the* Oskp *problem with a competitive ratio of at most* 2.

Proof. If the algorithm ends after the call of *filter_trivial*, the algorithm is at most 2-competitive. Assuming the condition in line 3 is met, the algorithm is at least 2-competitive by Lemma 4.3.7. Thus, we only have to prove that the algorithm is not worse than 2-competitive if it ends in line 6.

Let us first assume that $x_1 \neq b$. Then x_1 is packed and $x_1 > 1 - \frac{1+\delta}{1-\delta}b^-$. If any other large item x_i can be packed by the algorithm, the knapsack will be filled up to at least $\frac{1}{2}$, since

$$\begin{aligned} & x_1 + x_i \\ & > 1 - \frac{1+\delta}{1-\delta}b^- + \frac{1-\delta}{1+\delta} - b^- \\ & > 1 - \frac{1+\delta}{1-\delta}\frac{1}{2} + \frac{1-\delta}{1+\delta} - \frac{1}{2} \\ & \geq \frac{1}{2}\,, \end{aligned}$$

where the last inequality holds for $\delta \leq 3 - 2\sqrt{2} \sim 0.171$.

If no second large item fits into the algorithm's knapsack, an optimal solution cannot contain more than one large item either. This item of the optimal solution is bounded by $\frac{1+\delta}{1-\delta}b^- < \frac{1+\delta}{1-\delta}\frac{1}{2}$. The remaining items of an optimal solution then consist of all small items that the algorithm has ignored. We can bound the

total announced size of these small items, by $\frac{1}{2} - b^-$, using Lemma 4.3.5 and in consequence their actual size by $\frac{1+\delta}{1-\delta}(\frac{1}{2} - b^-)$. Putting it all together, in the worst case Algorithm 8 packs exactly one large item of size slightly larger than $1 - \frac{1+\delta}{1-\delta}b^-$. The optimal solution packs one large item of size slightly smaller than $\frac{1+\delta}{1-\delta}\frac{1}{2}$ and the maximum sum of small items. This results in a competitive ratio of

$$\frac{\frac{1+\delta}{1-\delta}b^- + \frac{1+\delta}{1-\delta}(\frac{1}{2} - b^-)}{1 - \frac{1+\delta}{1-\delta}b^-} = \frac{\frac{1+\delta}{1-\delta}\frac{1}{2}}{1 - \frac{1+\delta}{1-\delta}b^-} < \frac{\frac{1+\delta}{1-\delta}\frac{1}{c}}{1 - \frac{1+\delta}{1-\delta}\frac{1}{c}} \leq 2\,,$$

for $\delta \leq \frac{1}{7}$.

The only case left to handle is that $x_1 = b$. If $b \geq 1 - \frac{1+\delta}{1-\delta}b^-$, we can use the same argumentation as before. If however $b < 1 - \frac{1+\delta}{1-\delta}b^-$, then, since b is by definition the announced largest item, *all* large items are announced smaller or equal than $1 - \frac{1+\delta}{1-\delta}b^-$ and thus of actual size at most $\frac{1+\delta}{1-\delta}(1 - \frac{1+\delta}{1-\delta}b^-) < \frac{1+\delta}{1-\delta}(1 - \frac{1+\delta}{1-\delta}\frac{1}{2}/\frac{1+\delta}{1-\delta}) = \frac{1+\delta}{1-\delta}\frac{1}{2}$. As each large item is of size at least $\frac{1-\delta}{1+\delta} - b^-$, packing a second large item is still sufficient, since $2(\frac{1-\delta}{1+\delta} - b^-) > 2\frac{1-\delta}{1+\delta} - 1 \geq 1/2$ for $\delta \leq \frac{1}{7}$. Since no large item can exceed an actual size of $\frac{1+\delta}{1-\delta}b^-$ and since $x_1 < 1 - \frac{1+\delta}{1-\delta}b^-$, we are guaranteed that a second large item fits into the knapsack of the algorithm. The only case left is that there is no second large item. Then the algorithm and the optimal solution both pack the same large item $x_1 = b$, while the optimal solution can still add the maximum number of small items. The resulting competitive ratio is then $\frac{b^- + \frac{1+\delta}{1-\delta}(\frac{1}{2} - b^-)}{b^-} < 2$ for all δ in the given range. □

The second repeating upper bound matches the lower bound of Theorem 4.5.2 for almost the complete range of δ. For brevity of stating the theorem, let $\delta_k := \frac{12k+8}{32^{\frac{2}{3}}(3\sqrt{3}\sqrt{4k^3+11k^2+28k+44}-9k-34)^{\frac{1}{3}}} + \frac{1}{3}2^{\frac{2}{3}}(3\sqrt{3}\sqrt{4k^3+11k^2+28k+44}-9k-34)^{\frac{1}{3}} + 5/3$.

Theorem 4.4.2. *Given a fixed $k \in \mathbf{N}$ and a fixed δ with $\delta < \delta_k$. Algorithm 9 solves the* Oskp *problem with a competitive ratio of at most $c_k = \sqrt{(2+k)\frac{1+\delta}{1-\delta}}$.*

Proof. If the algorithm ends after the call of *filter_trivial*, the algorithm is at most c_k-competitive. The algorithm first reveals and discards small items and reveals the first large item x_1. If $x_1 \geq \frac{1}{k+2}$, the algorithm packs x_1 and from there on greedily any large item that still fits into the knapsack. If any second large item fits into the knapsack of the algorithm, its gain is at least $\frac{1}{k+2} + \frac{1-\delta}{1+\delta} - b^- > \frac{1}{c_k}$. Thus, an optimal solution consists of at most one large item of maximum size or a sum of large and small items adding up to at most the same size, due to Lemma 4.3.6. The competitive ratio is then $\frac{1+\delta}{1-\delta}\frac{1}{c_k}/\frac{1}{k+2} = c_k$.

Assuming $x_1 = b$, we know from Lemma 4.3.7 that any large item that is guaranteed to fit with b yields a ratio of at most c_k. Again, the algorithm packs

Algorithm 9 $\sqrt{(2+k)\frac{1+\delta}{1-\delta}}$-competitive Algorithm for a fixed $k \in \mathbf{N}$ and $\delta < \delta_k$.

filter_trivial()
Reveal items up to and including the first *large* item x_1.
if $x_1 \geq \frac{1}{k+2}$ or $x_1 = b$ **then**
    Greedily **pack** large items, including x_1. END
if $x_1 \leq 1 - \frac{1+\delta}{1-\delta}b^-$ **then**
    Pack x_1 with b. END
Let $B := \{x^- \in P \mid x^- > \frac{1-\delta}{1+\delta} - b^-\}$
if $\exists x_j^- \in B . x_j^- \leq (1-x_1)/\frac{1+\delta}{1-\delta}$ **then**
    Pack x_1 and x_j. END
for Reveal next $x_i \in P_{[i,n-1]}$ **do**
    if $\exists S \subseteq P_{[i,n-1]} . x_1 + \sum_{x^- \in S} x^- \geq \frac{1}{c_k} \wedge x_1 + \sum_{x^- \in S} x^+ \leq 1$ **then**
        Pack x_1 and S. END
Pack x_n.

x_1 and large items greedily. If any second large item fits into the knapsack of the algorithm, its gain is at least $b + (1-b)/\frac{1+\delta}{1-\delta} > \frac{1}{c_k}$ for the complete range of δ. Thus, an optimal solution consists of at most one large item of maximum size. The competitive ratio is then at worst $\frac{1+\delta}{1-\delta}b/b < c_k$ for the complete range of α.

Finally, if $x_1 \leq 1 - \frac{1+\delta}{1-\delta}b^-$, the algorithm packs x_1 together with b and has sufficient gain by Lemma 4.3.7.

Thus, we assume that $1 - \frac{1+\delta}{1-\delta}b^- < x_1 < \frac{1}{k+2}$. The algorithm next checks whether any other large item is of announced size such that it is both guaranteed to fit together with x_1 and guaranteed to fit into the knapsack together with x_1, i.e. if any $x_j^- \in B$ exists with $x_j^- \in [\frac{1}{c_k} - x_1, \frac{1-x_1}{\frac{1+\delta}{1-\delta}}]$. Note that $\frac{1}{c_k} - x_1 < \frac{1-\delta}{1+\delta} - b^-$, i.e. there exists no large item on the lower end of this bound if $\delta < \delta_k$. Thus, each large item apart from x_1 is of actual size larger than $(1-x_1)/\frac{1+\delta}{1-\delta}$.

If all of these bounds hold and thus do not admit a solution, the algorithm discards x_1 and continues to reveal large items. It only packs anything before the last large item if either the revealed item itself is of size $\frac{1}{c_k}$ or if it admits a packing that is guaranteed to fit and of size at least $\frac{1}{c_k}$. If neither is the case, the algorithm packs the last large item of size larger than $(1-x_1)/\frac{1+\delta}{1-\delta}$.

In the worst case, the optimal solution then consists of $k+1$ large items of size slightly smaller than $\frac{1}{c_k}$ each. The competitive ratio is then bounded by

$$\frac{k+1}{\sqrt{(2+k)\frac{1+\delta}{1-\delta}}} / \frac{1-\frac{1}{k+2}}{\frac{1+\delta}{1-\delta}} = \sqrt{(2+k)\frac{1+\delta}{1-\delta}} \, .$$

Note that the first item is too large to be packed on top of the $k+1$ items as $\frac{k+1}{\sqrt{(2+k)\frac{1+d}{1-d}}} + (1 - \frac{1+d}{1-d}\frac{1}{\sqrt{(2+k)\frac{1+d}{1-d}}}) > 1$ for $\delta < \frac{k}{k+2}$. □

The next segment's proof is the most involved one. It tightly matches the lower bound of Theorem 4.5.3 for $-1-k+\sqrt{4k+k^2} \leq \delta \leq \frac{k}{k+2}$, for any $k \in \mathbf{N}$.

Algorithm 10 $\frac{(1+\delta)(\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$-competitive Algorithm for a fixed $k \in \mathbf{N}$ and $\delta \leq \frac{k}{k+2}$.

filter_trivial()
Reveal items up to and including the first *large* item x_1.
Let $\ell_k := \frac{8}{9+8\delta-\delta^2-\sqrt{-1+\delta^2}\sqrt{-17-16\delta+\delta^2-16k}+8k}$
if $x_1 \geq \ell_k$ or $x_1 = b$ **then**
    Greedily **pack** large items, including x_1. END
if $x_1 \leq 1 - \frac{1+\delta}{1-\delta}b^-$ **then**
    Pack x_1 with b. END
Let $B := \{x^- \in P \mid x^- > \frac{1-\delta}{1+\delta} - b^-\}$
if $\exists x_j^-, x_l^- \in B . x_1 + x_j^+ + x_l^+ \leq 1$ **then**
    Pack x_1, x_j and x_l. END
else if $\exists x_L^- \in B . x_1 + x_L^- \geq \frac{1}{c_k} \wedge x_1 + x_L^+ \leq 1$ **then**
    Pack x_1 and x_L. END
for Reveal next $x_i \in P_{[i,n-1]}$ **do**
    if $\exists S \subseteq P_{[i,n-1]} . x_1 + \sum_{x^- \in S} x^- \geq \frac{1}{c_k} \wedge x_1 + \sum_{x^- \in S} x^+ \leq 1$ **then**
        Pack x_1 and S. END
Pack x_n.

Theorem 4.4.3. *Given a fixed $k \in \mathbf{N}$ and a fixed δ with $\delta \leq k/(k+2)$. Algorithm 10 solves the* Oskp *problem with a competitive ratio of at most* $c_k = \frac{(1+\delta)(\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$.

Proof. If the algorithm ends after the call of *filter_trivial*, the algorithm is at most c_k-competitive. Let $\ell_k = \frac{8}{9+8\delta-\delta^2-\sqrt{-1+\delta^2}\sqrt{-17-16\delta+\delta^2-16k}+8k}$, which is the solution to the equation $\frac{1+\delta}{1-\delta}\frac{1}{c_k}/\ell_k = c_k$.

The algorithm first reveals and discards small items and reveals the first large item x_1. If $x_1 \geq \ell_k$, the algorithm packs x_1 and from there on greedily any large item that still fits into the knapsack. If any second large item fits into the knapsack of the algorithm, its gain is at least $\ell_k + \frac{1-\delta}{1+\delta} - b^- > \frac{1}{c_k}$. Thus, an optimal solution consists of at most one large item of maximum size. The competitive ratio is then $\frac{1+\delta}{1-\delta}\frac{1}{c_k}/\ell_k = c_k$.

Assuming $x_1 = b$, we know from Lemma 4.3.7 that any large item that is guaranteed to fit with b yields a ratio of at most c_k. Again, the algorithm packs x_1 and large items greedily. If any second large item fits into the knapsack of the algorithm, its gain is at least $b + (1-b)/\frac{1+\delta}{1-\delta} > \frac{1}{c_k}$ for the complete range of δ. Thus, an optimal solution consists of at most one large item of maximum size. The competitive ratio is then at worst $\frac{1+\delta}{1-\delta} b/b < c_k$ for the complete range of α.

Finally, if $x_1 \leq 1 - \frac{1+\delta}{1-\delta} b^-$, the algorithm packs x_1 together with b and has sufficient gain by Lemma 4.3.7.

Thus, we assume that $1 - \frac{1+\delta}{1-\delta} b^- < x_1 < \ell_k$. If another two large items are guaranteed to fit together with x_1, the gain of the algorithm is again sufficient: $1 - \frac{1+\delta}{1-\delta} b^- + 2(\frac{1-\delta}{1+\delta} - b^-) > 1 - \frac{1+\delta}{1-\delta}\frac{1}{c_k} + 2(\frac{1-\delta}{1+\delta} - \frac{1}{c_k}) > \frac{1}{c_k}$. Thus, we may assume that there is at most one other large item x_j that is guaranteed to fit into the knapsack together with x_1. Its announced size is then in between $\frac{1-\delta}{1+\delta} - b^-$ and $\frac{1}{c_k} - x_1 < \frac{1}{c_k} - (1 - \frac{1+\delta}{1-\delta} b^-) < \frac{1}{c_k} - (1 - \frac{1+\delta}{1-\delta}\frac{1}{c_k}) = (1 + \frac{1+\delta}{1-\delta})\frac{1}{c_k} - 1$. Obviously, it has to hold that $x_1 + x_j < \frac{1}{c_k}$, otherwise the two items together are sufficient. Since the remaining large items $x_L \in I$ must not be announced with a size that guarantees that they fit into the knapsack together with x_1 *and* guarantee a packing of at least $\frac{1}{c_k}$, they have to be of announced size $x_L^- > \frac{1-x_1}{\frac{1+\delta}{1-\delta}}$. Since $x_1 < \ell_k < \frac{1}{k+2}$, we can lower bound the size of these large items by $x_L^- > (1-x_1)/\frac{1+\delta}{1-\delta} > (1 - \frac{1}{k+2})/\frac{1+\delta}{1-\delta} \geq \frac{1}{k+2}$, for all $\delta \leq \frac{1}{k+2}$. Since $\frac{1}{k+2} + \frac{1-\delta}{1+\delta} - b^- > \frac{1}{c_k}$, the item x_j that was guaranteed to fit together with x_1 must not be guaranteed to fit together with any other large item as well. This lower bounds its size by $x_j^- > (1 - \frac{1}{c_k})/(\frac{1+\delta}{1-\delta}) > x_1/\frac{1+\delta}{1-\delta}$.

If all of these bounds hold and thus do not admit a solution, the algorithm discards x_1 and continues to reveal large items. It only packs anything before the last large item if either the revealed item itself is of size $\frac{1}{c_k}$ or if it admits a packing that is guaranteed to fit and of size at least $\frac{1}{c_k}$. If neither is the case, the algorithm packs the last large item of size larger than $(1-x_1)/\frac{1+\delta}{1-\delta}$.

Recall that $x_L^- > (1-x_1)/\frac{1+\delta}{1-\delta}$, which means that $(k+1)x_L^- + x_1 > 1$ for $\delta \leq \frac{1}{k+2}$. Since also $x_1 + x_j > \frac{1}{c_k} > x_L^-$, the largest packing of a knapsack consists of k items x_L or size slightly smaller than $\frac{1}{c_k}$, together with both items x_1 and x_j. The competitive ratio is then bounded by

$$(x_1 + x_j + k\frac{1}{c_k})/\frac{1-x_1}{\frac{1+\delta}{1-\delta}} \ .$$

We are now interested in maximizing this ratio by choosing appropriate values for x_1 and x_j. The gain of the algorithm is minimized if the value of x_1 is maximized. Since $x_j^- > x_1/\frac{1+\delta}{1-\delta}$ and $x_1 + x_j < \frac{1}{c_k}$ both have to hold, the ratio is maximized to a value of c_k when choosing $x_1 = \frac{2\sqrt{-1+\delta^2}}{\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k}}$ and thus $x_j = \frac{1+\delta}{1-\delta} x_1$. □

For the remainder of unmatched lower bounds, we re-use Algorithm 8 that was used to prove the upper bound of 2 for $0 < \delta \leq \frac{1}{7}$. The only difference is the competitive ratio that is aimed for, which is $\frac{1+\delta+\sqrt{5+2\delta-3\delta^2}}{2-2\delta}$ for $\frac{1}{7} \leq \delta$. The proof is almost identical to that of Theorem 4.4.1.

Theorem 4.4.4. *Given a fixed $k \in \mathbf{N}$ and a fixed δ with $\frac{1}{7} \leq \delta$. Algorithm 8 solves the* Oskp *problem with a competitive ratio of at most $c = \frac{1+\delta+\sqrt{5+2\delta-3\delta^2}}{2-2\delta}$.*

Proof. If the algorithm ends after the call of *filter_trivial*, the algorithm is at most c-competitive. Assuming the condition in line 3 is met, the algorithm is at least c-competitive by Lemma 4.3.7. Thus, we only have to proof that the algorithm is not worse than c-competitive if it ends in line 6.

Let us first assume that $x_1 \neq b$. Then x_1 is packed and $x_1 > 1 - \frac{1+\delta}{1-\delta}b^-$. If any other large item x_1 can be packed by the algorithm, the knapsack will be filled up to at least $\frac{1}{2}$, since

$$\begin{aligned} & x_1 + x_i \\ & > 1 - \frac{1+\delta}{1-\delta}b^- + \frac{1-\delta}{1+\delta}b^- \\ & > 1 - \frac{1+\delta}{1-\delta}\frac{1}{c} + \frac{1-\delta}{1+\delta}\frac{1}{c} \\ & \geq \frac{1}{c}\,, \end{aligned}$$

where the last inequality holds for all values of δ between 0 and 1.

If no second large item fits into the algorithm's knapsack, an optimal solution cannot contain more than one large item either. This item of the optimal solution is bounded by $\frac{1-\delta}{1+\delta}b^- < \frac{1-\delta}{1+\delta}\frac{1}{c}$. The remaining items of an optimal solution then consist of all small items that the algorithm has ignored. We can bound the total announced size of these small items, by $\frac{1}{c} - b^-$, using Lemma 4.3.5 and in consequence their actual size by $\frac{1+\delta}{1-\delta}(\frac{1}{c} - b^-)$. Putting it all together, in the worst case Algorithm 8 packs exactly one large item of size slightly larger than $1 - \frac{1+\delta}{1-\delta}b^-$. The optimal solution packs one large item of size slightly smaller than $\frac{1-\delta}{1+\delta}\frac{1}{c}$ and the maximum sum of small items. This results in a competitive ratio of

$$\frac{\frac{1+\delta}{1-\delta}b^- + \frac{1+\delta}{1-\delta}(\frac{1}{c} - b^-)}{1 - \frac{1+\delta}{1-\delta}b^-} = \frac{\frac{1+\delta}{1-\delta}\frac{1}{c}}{1 - \frac{1+\delta}{1-\delta}b^-} < \frac{\frac{1+\delta}{1-\delta}\frac{1}{c}}{1 - \frac{1+\delta}{1-\delta}\frac{1}{c}} = c\,,$$

for $\delta \leq \frac{1}{7}$.

The only case left to handle is that $x_1 = b$. If $b \geq 1 - \frac{1+\delta}{1-\delta}b^-$, we can use the same argumentation as before. If however $b < 1 - \frac{1+\delta}{1-\delta}b^-$, then, since b is by definition

the announced largest item, *all* large items are announced smaller than $1 - \frac{1+\delta}{1-\delta}b^-$ and thus of actual size at most $\frac{1+\delta}{1-\delta}(1 - \frac{1+\delta}{1-\delta}b^-) < \frac{1+\delta}{1-\delta}(1 - \frac{1+\delta}{1-\delta}\frac{1}{c}/\frac{1+\delta}{1-\delta}) = \frac{1+\delta}{1-\delta}\frac{1}{c}$. Since no large item can exceed an actual size of $\frac{1+\delta}{1-\delta}b^-$ and since $x_1 < 1 - \frac{1+\delta}{1-\delta}b^-$, we are guaranteed that a second large item fits into the knapsack of the algorithm. In the worst case, an optimal solution packs b and an item of size $\frac{1+\delta}{1-\delta}b^-$, while the algorithm only packs b and a large item of minimal size. The competitive ratio is then

$$(\frac{1+\delta}{1-\delta}b^- + b^-)/(b^- + \frac{1-\delta}{1+\delta} - b^-) < c\,,$$

where the last inequality holds for all values of δ between 0 and 1.

The only case left is that there is no second large item. Then the algorithm and the optimal solution both pack the same large item $x_1 = b$, while the optimal solution can still add the maximum number of small items. The resulting competitive ratio is then $\frac{b^- + \frac{1+\delta}{1-\delta}(\frac{1}{c} - b^-)}{b^-} < c$ for all δ between 0 and 1. □

4.5 Lower Bounds

We are able to provide lower bounds for all values of δ. These bounds turn out to be hard to find but easy to verify. We can observe that different bounds dominate one another in an alternating fashion, which is a consequence of the scaling factor $\frac{1+\delta}{1-\delta}$ crossing certain thresholds.

We start with a simple lower bound of 2 for all values of $0 < \delta \leq \frac{1}{7}$.

Theorem 4.5.1. *Given a fixed δ with $0 < \delta < 1$. There exists no algorithm solving the* Oskp *problem with a competitive ratio better than* 2.

Proof. Let $\varepsilon > 0$ such that $\varepsilon < \frac{\delta}{4}$. The algorithm is given the following prediction:

$$P = (\frac{1}{2} + \varepsilon, \frac{1}{2} - \varepsilon, \frac{1}{2})$$

We do a full case distinction on the possible behaviors of an algorithm. The first item is revealed to be of size $x_1 = \frac{1}{2} + \varepsilon$.

Case 1: An algorithm packs x_1 with $x_1 = \frac{1}{2} + \varepsilon$:

The next two items are presented of size $x_2 = x_3 = \frac{1}{2}$, fitting together with one another but not with x_1. The competitive ratio is then $\frac{1}{\frac{1}{2}+\varepsilon} = 2\frac{1}{1+2\varepsilon}$.

Case 2: An algorithm rejects x_1 with $x_1 = \frac{1}{2} + \varepsilon$:

The second item is then presented to be of size $x_2 = \frac{1}{2} - \varepsilon$, with the last item presented of size $x_3 = \frac{1}{2} + 2\varepsilon$, independent of the algorithm's decision on the item x_2. The optimal solution consists of the first two items, while the biggest packing an algorithm can achieve is $\frac{1}{2} + 2\varepsilon$. The competitive ratio is then $\frac{1}{\frac{1}{2}+2\varepsilon} = 2\frac{1}{1+4\varepsilon}$. □

The next bound connects seamlessly to the previous bound of 2. As with all following bounds, the following theorem describes a set of bounds that only become valid once a certain value of δ is reached.

Theorem 4.5.2. *For every $k \in \mathbf{N}$ and $\frac{k^2+k-1}{k^2+3k+3} \leq \delta$, there exists no algorithm solving the* Oskp *problem with a competitive ratio better than $c_k = \sqrt{(2+k)\frac{1+\delta}{1-\delta}}$-competitive.*

Proof. Let $\varepsilon > 0$. The algorithm is given the following prediction:

$$P = (\frac{1}{2+k}, \underbrace{\frac{1}{c_k}, \ldots, \frac{1}{c_k}}_{k+1 \text{ many}}, \frac{1-\frac{1}{2+k}}{\frac{1+\delta}{1-\delta}})$$

We do a full case distinction on the possible behaviors of an algorithm. The first item is presented as $\frac{1}{2+k} + \varepsilon$.

Case 1: An algorithm packs $\frac{1}{2+k} + \varepsilon$:

The next items are all presented of maximum possible size. Since $\frac{1+\delta}{1-\delta}\frac{1}{c_k} + \frac{1}{2+k} + \varepsilon > 1$ for each choice of k and all $\delta \geq \frac{k^2+k-1}{k^2+3k+3}$, this means that no two items fit together into the knapsack. The biggest item of an optimal packing is then $\frac{1}{c_k}\frac{1+\delta}{1-\delta}$. The competitive ratio is then $\frac{1+\delta}{1-\delta}\frac{1}{c_k}/(\frac{1}{2+k} + \varepsilon) \overset{\varepsilon \to 0}{=} c_k$.

Case 2: An algorithm rejects $\frac{1}{2+k} + \varepsilon$:

The next item is presented of size $\frac{1}{c_k}$.

Case 2.1: An algorithm packs $\frac{1}{c_k}$:

The next items are each presented of size $1 - \frac{1}{2+k} - \varepsilon < \frac{1+\delta}{1-\delta}\frac{1}{c_k}$, allowing an optimal packing to reach a size of 1. Since $\frac{1}{c_k} > \frac{1}{2+k}$ for all values of δ in the given range for the given k, the algorithm cannot pack either of these items, resulting in the target competitive ratio.

Case 2.2: An algorithm rejects $\frac{1}{c_k}$:

The next items, up to and including the penultimate one, are presented to be of size $\frac{1}{c_k}$ as well. If an algorithm packs one of these items, Case 2.1 applies. We thus assume that all these items are rejected. The last item is then presented to be of size $\frac{1-\frac{1}{2+k}}{\frac{1+\delta}{1-\delta}}$.

If the algorithm does not pack this item, it is not competitive. An optimal solution can pack all $k+1$ items of size $\frac{1}{c_i}$ if $\delta \geq \frac{k^2+k-1}{k^2+3k+3}$. The competitive ratio is then $\frac{k+1}{c_k}/\frac{1-\frac{1}{2+k}}{\frac{1+\delta}{1-\delta}} = c_k$. □

We continue with a lower bound superseding the previous ones for (relative) higher values of δ.

Theorem 4.5.3. *For every $k \in \mathbf{N}$ and $-1-k+\sqrt{4k+k^2} \leq \delta$, there exists no algorithm solving the* Oskp *problem with a competitive ratio better than* $c_k = \frac{(1+\delta)(\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k})}{4\sqrt{-1+\delta^2}}$.

Proof. Let $i_k = \frac{2\sqrt{-1+\delta^2}}{\sqrt{-1+\delta^2}+\sqrt{-17-16\delta+\delta^2-16k}}$, which is the solution to the equation $i_k + \frac{i_k}{\frac{1+\delta}{1-\delta}} = \frac{1}{c_k}$ and $\varepsilon > 0$. The algorithm is given the following prediction:

$$P = (i_k, \frac{i_k}{\frac{1+\delta}{1-\delta}} + \varepsilon, \underbrace{\frac{1-i_k}{\frac{1+\delta}{1-\delta}} + \varepsilon, \ldots, \frac{1-i_k}{\frac{1+\delta}{1-\delta}} + \varepsilon}_{i \text{ many}})$$

Note that $\frac{1-i_k}{\frac{1+\delta}{1-\delta}} \leq \frac{1}{c_k}$ only if $\delta \geq -1-k+\sqrt{4k+k^2}$.

We do a full case distinction on the possible behaviors of an algorithm. The first item is presented as i_k.

Case 1: An algorithm packs i_k:

The next item is revealed as $i_k/\frac{1+\delta}{1-\delta} + \varepsilon$.

Case 1.1: An algorithm packs $i_k/\frac{1+\delta}{1-\delta} + \varepsilon$:

The next item is revealed as $1-i_k$, thus fitting together with the first item, but not into the knapsack of the algorithm. All subsequent items are presented as $1-i_k$ as well. The optimal packing is thus of size 1, while the algorithm only achieves a gain of $i_k + i_k/\frac{1+\delta}{1-\delta} + \varepsilon = \frac{1}{c_k} + \varepsilon$.

Case 1.2: An algorithm rejects $i_k/\frac{1+\delta}{1-\delta} + \varepsilon$:

All subsequent items are each revealed to be of size $1-i_k+\varepsilon$, thus not fitting into the knapsack of the algorithm. An optimal strategy packs the items $i_k/\frac{1+\delta}{1-\delta}+\varepsilon$ and $1-i_k+\varepsilon$, while the algorithm only has the first item of size i_k in its knapsack, resulting in a ratio of

$$\frac{\frac{i_k}{\frac{1+\delta}{1-\delta}} + 1 - i_k + 2\varepsilon}{i_k} > c_k \ .$$

Case 2: An algorithm rejects i_k:

The second item is revealed of size $i_k + \varepsilon$.

Case 2.1: An algorithm packs $i_k + \varepsilon$:

The remaining items are each revealed to be of size $1-i_k$, thus not fitting into the knapsack of the algorithm. An optimal packing is the first item together with one of the last items, adding up to a gain of 1. The algorithm can only pack $i_k+\varepsilon$, resulting in a ratio of $\frac{1}{i_k} \geq c_i$.

Case 2.2: An algorithm rejects $i_k + \varepsilon$:

The next item is revealed of size $\frac{1}{c_k}$.

Case 2.2.1: An algorithm packs $\frac{1}{c_k}$:
The remaining items are revealed of size $1 - i_k$, not fitting into the knapsack of the algorithm but fully filling an optimal knapsack together with the first item, resulting exactly in a ratio of c_k.
Case 2.2.2: An algorithm rejects $\frac{1}{c_k}$:
Up to and including the penultimate item, the next items are also revealed as $\frac{1}{c_k}$. If an algorithm packs any of them, the rest of the instance behaves as in Case 2.2.1.

Assuming this is not the case, the last item is presented as $(1 - i_k)/(\frac{1+\delta}{1-\delta}) + \varepsilon$, which is the only item the algorithm can pack. An optimal algorithm packs all items except the last one. These items are of sum at most one if $\delta \geq -1 - k + \sqrt{4k + k^2}$. The resulting competitive ratio is

$$\frac{2i_k + \varepsilon + (k-1)\frac{1}{c_k}}{\frac{1-i_k}{\frac{1+\delta}{1-\delta}} + \varepsilon},$$

which converges to c_k for ε going to 0. □

Just like with the two previous bounds, the last lower bound dominates the other two for yet (relative) higher values of δ.

Theorem 4.5.4. *For every $k \in \mathbf{N}_{>1}$ and $\frac{k-1}{k+1} < \delta < \frac{k}{k+2}$, there exists no algorithm solving the* Oskp *problem with a competitive ratio better than $c_k = \frac{1+k}{2} - \frac{\sqrt{(-1+\delta)(-5-k(2+k)+\delta(-1+k)(3+k))}}{-2+2\delta}$.*

Proof. Let $i_k = \frac{2(\delta-1)}{3\delta - \sqrt{-(\delta-1)(-4\delta(k-1)+k^2+2k+5)}+\delta k-k-3}$ be the solution to the equation $c_k = \frac{1}{i_k} - 1$ and let $j_k = \frac{i_k^2}{1-(k+2)i_k}$. The algorithm is given the following prediction:

$$P = (i_k + \varepsilon, \underbrace{j_k}_{i \text{ many}})$$

We do a full case distinction on the possible behaviors of an algorithm. The first item is presented as i_k.
Case 1: An algorithm packs $i_k + \varepsilon$:
Each of the next items are presented of maximum possible size $\frac{1+\delta}{1-\delta} j_k = 1 - i_k$, which is the counterpart to the first item, except for the additional ε. Since $1 - i_k > \frac{1}{2}$, no two items fit into the knapsack in this case. The optimal solution thus packs an item of size $1 - i_k$. The competitive ratio is then $\frac{1-i_k}{i_k+\varepsilon} \overset{\varepsilon \to 0}{=} c_k$.
Case 2: An algorithm rejects $i_k + \varepsilon$:
The next items is presented to be of size j_k.

Case 2.1: An algorithm packs j_k:

If an algorithm packs such an item before the last item, each subsequent item is presented to be of almost maximum size $\frac{1+\delta}{1-\delta}j_k - \varepsilon$. For $\delta < \frac{k}{k+2}$ it holds that $j_k + \frac{1+\delta}{1-\delta}j_k - \varepsilon > 1$. Thus, the algorithm only packs an item of size j_k, while the optimal solution is 1, as such an item of almost maximum size perfectly fits together with the first item of the instance. The competitive ratio is then $\frac{1}{j_k} > c_k$, as $j_k < \frac{1}{c_k}$ for $\delta > \frac{1}{3}$.

Case 2.2: An algorithm rejects j_k:

Up to and including the penultimate item, the next items are also revealed as j_k. If an algorithm packs any of them, the rest of the instance behaves as in Case 2.1.

Assuming this is not the case, the last item is presented as j_k, which is the only item the algorithm can pack. An optimal algorithm packs all items except the last one. These items are of sum at most 1 if $\delta > \frac{1}{3}$. The competitive ratio is then $\frac{i_k+\varepsilon+kj_k}{j_k} \overset{\varepsilon\to 0}{=} c_k$. □

4.6 Open Problems

The Oskp problem turns out to be very involved problem. While we were able to give tight bounds on the competitive ratio for some values of δ, the analysis is not complete. We are quite confident that the lower bounds that we determined should reflect the actual competitive ratio in the remaining open segments, but finding a proof to do so is beyond our capability.

A logical next step would be to look at the general Online Knapsack problem in the setting of predictions. However, one would first have to determine which part of the input is predicted: The size, the weight or the density of the items. The model of predictions offers to be applied to — and already has been applied to — further problems beyond knapsack problems. To our knowledge, more discretized problems, such as graph problems like the Minimum Vertex Cover problem have not been studied in the setting of predictions. Finding an appropriate, unifying model on what to predict in such problems would be interesting.

Chapter 5

Conclusion

We opened this work with the observation that some classical optimization problems do not fit the mold of online analysis due to specialized families of instance barring any online algorithm from being competitive. For the Online Simple Knapsack the Online Minimum Vertex Cover and the more generalized $\mathcal{F}$-Node- and $\mathcal{F}$-Edge-Deletion problems, we have seen that slight modifications of the classical model for online computations allowed for a fine-grained competitive analysis, and for the analysis of advice complexity.

In Chapter 2 we have analyzed general $\mathcal{F}$-Node- and $\mathcal{F}$-Edge-Deletion problems in the classical model, the *late accept* model introduced by Boyar et al. and — most in-depth — regarding the advice complexity of such deletion problems. We have found that both the competitive ratio in the late accept model as well as the advice complexity are dominated by the size of the biggest forbidden graph for most families $\mathcal{F}$. However, we also found out that for certain families $\mathcal{F}$, advice logarithmic in the size of the optimal solution or even no advice can be sufficient to optimally solve the corresponding $\mathcal{F}$-Node-Deletion problem. In the case of $\mathcal{F}$-Edge-Deletion problems, we found that for connected families $\mathcal{F}$, the advice is heavily dependent on the concrete forbidden $\mathcal{F}$, as deleting an edge from one $H_i \in \mathcal{F}$ could result in another graph $H_j \in \mathcal{F}$ being induced in the online graph.

In Chapter 3 we looked at the Online Simple Knapsack problem with *reservations*, that allow an algorithm to delay decisions on elements of the input indefinitely, for a cost relative to the gain of the element. The resulting competitive ratio dependent on the concrete reservation costs $\alpha \in [0, 1]$ was partitioned into four function segments, resulting in a continuous, but non-differentiable function. We also looked at a slightly modified model, in which reservations were reimbursed for each reserved item that was ultimately packed. Here, the competitive ratio did not diverge for α tending to one as in the first model, but rather increased continuously. Interestingly, even for arbitrarily small reservation costs, no online algorithm can be better than 2-competitive in either reservation model.

This fact could also be observed for the model that we studied in Chapter 4. Here, an algorithm was given a *prediction* of the upcoming instance, with the caveat that these predictions could not be completely trusted. Each item's actual size is bounded by an error term, and the actual size of the item is only revealed when an algorithm is to decide whether to pack or reject an item. While the competitive analysis in this model was incomplete due to its complexity, the partial image is already very interesting, with the resulting competitive ratio dependent on the concrete *distortion* $\delta \in [0, 1]$ being a non-continuous, apparently period function of three function segments.

5.1 Where to Go from Here

Our focus in this thesis has lain very much on the analysis of concrete problems under novel variations of the classical online model. A very natural question would be to ask whether these models share a connection.

We see potential in further refining the given models, as already hinted at in Section 4.6. The model for predictions is still very young and has to our knowledge so far only been applied to rather "continuous" problems, i.e. to problems where a perturbed element is still a perfectly valid element for an algorithm to compute on. Once we go to a more discretized world, e.g. to that of predictions on graph optimization problems, such as Feedback Vertex Set, Coloring, Longest Path or others, it is not immediately clear how a prediction or indeed a perturbed element of the input should look like. One could however decide to open up the model to allow for other parameters to be predicted, such as the degree of each node of the input, with appropriate rounding for the actual degree.

The model for reservations is more flexible in this regard, as the costs of reservations can be made constant for each reserved element and subtracted from the gain of an algorithm. We would be most interested to see further problems be analyzed under this and other variations of the model.

Finally, the late accept model is arguably one of the most natural feeling ones that we studied in this work. While weakening the strictness of immediate decisions, only asking for an action when an action is warranted is the way humans treat most tasks. However, since only few problems suffer from the specific problem that an action is asked where none is yet necessary, we suspect that not many more results beyond the incompletely analyzed problems will occur in the future.

While the world of online analysis soon turns 40 — if one sees the paper of Sleator and Tarja [65] as the first paper in the field — we still see an active research community trying to better understand a very specialized class of approximation algorithms, producing results that go beyond numbly applying competitive analysis to yet another problem. We think that there is a lot of merit left in looking at

further adjustments of the classical online analysis in order to better understand actions that guarantee us that the worst case may not always be so bad after all.

Bibliography

[1] Spyros Angelopoulos, Shahin Kamali, and Kimia Shadkami. Online bin packing with predictions. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 4574–4580. ijcai.org, 2022.

[2] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 345–355. PMLR, 2020.

[3] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Mixing predictions for online metric algorithms. *CoRR*, abs/2304.01781, 2023.

[4] David Arthur, Bodo Manthey, and Heiko Röglin. k-means has polynomial smoothed complexity. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 405–414. IEEE Computer Society, 2009.

[5] Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1070–1080. ACM, 2021.

[6] Yossi Azar, Eldad Peretz, and Noam Touitou. Distortion-oblivious algorithms for scheduling on multiple machines. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[7] Eric Balkanski, Vasilis Gkatzelis, and Xizhi Tan. Strategyproof scheduling with predictions. In Yael Tauman Kalai, editor, *14th Innovations in Theoret-*

ical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA, volume 251 of *LIPIcs*, pages 11:1–11:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[8] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis of the multi-level feedback algorithm. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 462–471. IEEE Computer Society, 2003.

[9] René Beier and Berthold Vöcking. Random knapsack in expected polynomial time. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 232–241. ACM, 2003.

[10] Magnus Berg, Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. Online minimum spanning trees with weight predictions. *CoRR*, abs/2302.12029, 2023.

[11] Niklas Berndt and Henri Lotze. Advice complexity bounds for online delayed $\mathcal{F}$-node-, H-node- and H-edge-deletion problems. In Sun-Yuan Hsieh, Ling-Ju Hung, and Chia-Wei Lee, editors, *Combinatorial Algorithms - 34th International Workshop, IWOCA 2023, Tainan, Taiwan, June 7-10, 2023, Proceedings*, volume 13889 of *Lecture Notes in Computer Science*, pages 62–73. Springer, 2023.

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

[13] Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theor. Comput. Sci.*, 554:95–108, 2014.

[14] Hans-Joachim Böckenhauer, Ralf Klasing, Tobias Mömke, Peter Rossmanith, Moritz Stocker, and David Wehner. Online knapsack with removal and recourse. In Sun-Yuan Hsieh, Ling-Ju Hung, and Chia-Wei Lee, editors, *Combinatorial Algorithms - 34th International Workshop, IWOCA 2023, Tainan,*

Taiwan, June 7-10, 2023, Proceedings, volume 13889 of *Lecture Notes in Computer Science*, pages 123–135. Springer, 2023.

[15] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. Online algorithms with advice: The tape model. *Inf. Comput.*, 254:59–83, 2017.

[16] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On the advice complexity of the k-server problem. *J. Comput. Syst. Sci.*, 86:159–170, 2017.

[17] Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theor. Comput. Sci.*, 527:61–72, 2014.

[18] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[19] Joan Boyar, Stephan J. Eidenbenz, Lene M. Favrholdt, Michal Kotrbcík, and Kim S. Larsen. Online dominating set. In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*, volume 53 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[20] Joan Boyar, Lene M. Favrholdt, Shahin Kamali, and Kim S. Larsen. Online interval scheduling with predictions. *CoRR*, abs/2302.13701, 2023.

[21] Joan Boyar, Lene M. Favrholdt, Michal Kotrbcík, and Kim S. Larsen. Relaxing the irrevocability requirement for online graph algorithms. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, volume 10389 of *Lecture Notes in Computer Science*, pages 217–228. Springer, 2017.

[22] Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *ACM Comput. Surv.*, 50(2):19:1–19:34, 2017.

[23] Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. Online unit profit knapsack with untrusted predictions. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2022, June 27-29, 2022, Tórshavn, Faroe Islands*, volume 227 of *LIPIcs*, pages 20:1–20:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[24] Elisabet Burjons, Janosch Fuchs, and Henri Lotze. The slotted online one-sided crossing minimization problem on 2-regular graphs. In Cristina Bazgan and Henning Fernau, editors, *Combinatorial Algorithms - 33rd International Workshop, IWOCA 2022, Trier, Germany, June 7-9, 2022, Proceedings*, volume 13270 of *Lecture Notes in Computer Science*, pages 243–256. Springer, 2022.

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

[26] Elisabet Burjons, Matthias Gehnen, Henri Lotze, Daniel Mock, and Peter Rossmanith. The online simple knapsack problem with reservation and removability. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 29:1–29:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[27] Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996.

[28] Li-Hsuan Chen, Ling-Ju Hung, Henri Lotze, and Peter Rossmanith. Further results on online node- and edge-deletion problems with advice. In Leszek Gasieniec, Ralf Klasing, and Tomasz Radzik, editors, *Combinatorial Algorithms - 31st International Workshop, IWOCA 2020, Bordeaux, France, June 8-10, 2020, Proceedings*, volume 12126 of *Lecture Notes in Computer Science*, pages 140–153. Springer, 2020.

[29] Li-Hsuan Chen, Ling-Ju Hung, Henri Lotze, and Peter Rossmanith. Online node- and edge-deletion problems with advice. *Algorithmica*, 83(9):2719–2753, 2021.

[30] Marc Demange and Vangelis Th. Paschos. On-line vertex-covering. *Theor. Comput. Sci.*, 332(1-3):83–108, 2005.

[31] René Descartes. *Discours de la methode pour bien conduire sa raison, et chercher la verité dans les sciences. Plus la Dioptrique. Les Meteores. Et la Geometrie. – Qui sont des essais de cete Methode.* De l'Imprimerie de Ian Maire, 1637.

[32] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 10393–10406, 2021.

[33] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *ITA*, 43(3):585–613, 2009.

[34] Jérôme Dohrau. Online makespan scheduling with sublinear advice. In Giuseppe F. Italiano, Tiziana Margaria-Steffen, Jaroslav Pokorný, Jean-Jacques Quisquater, and Roger Wattenhofer, editors, *SOFSEM 2015: Theory and Practice of Computer Science - 41st International Conference on Current Trends in Theory and Practice of Computer Science, Pec pod Sněžkou, Czech Republic, January 24-29, 2015. Proceedings*, volume 8939 of *Lecture Notes in Computer Science*, pages 177–188. Springer, 2015.

[35] Jan Dreier, Henri Lotze, and Peter Rossmanith. Hard problems on random graphs. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[36] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theor. Comput. Sci.*, 412(24):2642–2656, 2011.

[37] Jessica A. Enright and Kitty Meeks. Deleting edges to restrict the size of an epidemic: A new application for treewidth. *Algorithmica*, 80(6):1857–1889, 2018.

[38] Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlöter. Learning-augmented query policies for minimum spanning tree with uncertainty. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 49:1–49:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[39] Daniel H. Greene and Donald E. Knuth. *Mathematics for the Analysis of Algorithms*. Birkhäuser, 3rd edition, 1990.

[40] Xin Han, Yasushi Kawase, and Kazuhisa Makino. Online unweighted knapsack problem with removal cost. *Algorithmica*, 70(1):76–91, 2014.

[41] Xin Han, Yasushi Kawase, Kazuhisa Makino, and He Guo. Online removable knapsack problem under convex function. *Theor. Comput. Sci.*, 540:62–69, 2014.

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

[43] Xin Han and Kazuhisa Makino. Online removable knapsack with limited cuts. *Theor. Comput. Sci.*, 411(44-46):3956–3964, 2010.

[44] Xin Han and Kazuhisa Makino. Online minimization knapsack problem. *Theor. Comput. Sci.*, 609:185–196, 2016.

[45] P. Henrici. *Applied and Computational Complex Analysis*, volume 1. Wiley, 1988.

[46] Juraj Hromkovič, Rastislav Královič, and Richard Královič. Information complexity of online problems. In Petr Hliněný and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 24–36. Springer, 2010.

[47] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Online knapsack with frequency predictions. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 2733–2743, 2021.

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

[49] Kazuo Iwama and Guochuan Zhang. Online knapsack with resource augmentation. *Inf. Process. Lett.*, 110(22):1016–1020, 2010.

[50] Billy Jin and Will Ma. Online bipartite matching with advice: Tight robustness-consistency tradeoffs for the two-stage model. In *NeurIPS*, 2022.

[51] Dennis Komm. *An Introduction to Online Computation - Determinism, Randomization, Advice*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.

[52] Mukkai S. Krishnamoorthy and Narsingh Deo. Node-deletion np-complete problems. *SIAM J. Comput.*, 8(4):619–625, 1979.

[53] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1859–1877. SIAM, 2020.

[54] Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[55] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980.

[56] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311. PMLR, 2018.

[57] Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Math. Program.*, 68:73–104, 1995.

[58] Nadav Merlis, Hugo Richard, Flore Sentenac, Corentin Odic, Mathieu Molina, and Vianney Perchet. On preemption and learning in stochastic scheduling. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 24478–24516. PMLR, 2023.

[59] Michele Monaci, Ulrich Pferschy, and Paolo Serafini. Exact solution of the robust knapsack problem. *Comput. Oper. Res.*, 40(11):2625–2631, 2013.

[60] Assaf Natanzon, Ron Shamir, and Roded Sharan. Complexity classification of some edge modification problems. *Discret. Appl. Math.*, 113(1):109–128, 2001.

[61] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9684–9693, 2018.

[62] Marc P. Renault, Adi Rosén, and Rob van Stee. Online algorithms with advice for bin packing and scheduling problems. *Theor. Comput. Sci.*, 600:155–170, 2015.

[63] Peter Rossmanith. On the advice complexity of online edge- and node-deletion problems. In Hans-Joachim Böckenhauer, Dennis Komm, and Walter 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.

[64] Sebastian Seibert, Andreas Sprock, and Walter Unger. Advice complexity of the online coloring problem. In Paul G. Spirakis and Maria J. Serna, editors, *Algorithms and Complexity, 8th International Conference, CIAC 2013, Barcelona, Spain, May 22-24, 2013. Proceedings*, volume 7878 of *Lecture Notes in Computer Science*, pages 345–357. Springer, 2013.

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

[66] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 296–305. ACM, 2001.

[67] Clemens Thielen, Morten Tiedemann, and Stephan Westphal. The online knapsack problem with incremental capacity. *Math. Methods Oper. Res.*, 83(2):207–242, 2016.

[68] Mihalis Yannakakis. Node- and edge-deletion np-complete problems. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 253–264. ACM, 1978.

[69] Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM J. Comput.*, 10(2):310–327, 1981.

[70] Yunhong Zhou, Deeparnab Chakrabarty, and Rajan M. Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In Christos H. Papadimitriou and Shuzhong Zhang, editors, *Internet and Network Economics, 4th International Workshop, WINE 2008, Shanghai, China, December 17-20, 2008. Proceedings*, volume 5385 of *Lecture Notes in Computer Science*, pages 566–576. Springer, 2008.