Master Thesis

Analysis of Incremental Knapsack Problem Variants

Danny-Marvin Rademacher

Department of Computer Science

University of Bonn

1st Examiner	Prof. Dr. Heiko Röglin
2nd Examiner	Prof. Dr. Rolf Klein
Due date	January 18, 2018

Eidesstattliche Erklärung

Hiermit versichere ich, Danny-Marvin Rademacher, an Eides statt und durch meine Unterschrift, dass die vorliegende Arbeit von mir selbstständig, ohne fremde Hilfe angefertigt worden ist. Inhalte und Passagen, die aus fremden Quellen stammen und direkt oder indirekt übernommen worden sind, wurden als solche kenntlich gemacht.

Ferner versichere ich, dass ich keine andere, außer der im Literaturverzeichnis angegebene Literatur verwendet habe. Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Bonn, den 18.01.2018 .....

Ort, Datum

Danny-Marvin Rademacher

Abstract

We study the problem of incrementally packing a knapsack without knowing its capacity. We prepare a chain of solutions such that the solutions are extensions of each other in terms of what items are packed. So once an item is part of a solution every later solution has to include that item. In general, the incremental problem has no optimal solution, so we determine the competitive factor of the chain and the optimal solution of the knapsack problem. We show that there are instances which have no constant competitive factor algorithms.

Then, we consider the problem under the constraint that only such capacities are permitted where no item weight is bigger than that capacity. For that definition we present an efficient 2-competitive algorithm on general instances. For instances, where each item has unit density, we present an efficient algorithm which is $\frac{1+\sqrt{5}}{2}$-competitive. In both cases the algorithms have the best possible competitive factor.

We also investigate these problems when multiple knapsacks are allowed. For multiple knapsacks, we present efficient algorithms with a competitive factor of 4 on general instances and a factor of 2 on unit density instances.

Contents

List of Algorithms

Acknowledgments

I would like to express my sincere gratitude to my supervisors Heiko Röglin and Clemens Rösner. They have been very supportive and provided guidance and valuable advice through the process of this master thesis.

Special thanks go to Sally Chau, Fabian Thorand, Jens Harder, Catherine Capellen and Klaus Tulbure for proofreading this thesis and for many valuable suggestions.

Finally, I would like to express my deepest gratitude to my family for their support and encouragement throughout my studies.

Chapter 1

Introduction

The knapsack problem is a problem in combinatorial optimization. It is well-known and one of the most studied problems in computer science [1]. An instance of the knapsack problem consists of a set of items and has a capacity threshold. Each item has a profit and a weight. The task is to find a subset of items whose accumulated weight does not exceed the capacity threshold while maximizing the accumulated profit of the chosen items. This means that every item demands the binary decision whether to pack it into the knapsack or not [2, 3].

This problem has many applications in real life, for example the logistic problem, the cutting stock problem and portfolio selection problem [2]. The task of the logistic problem is loading transport ships with container of goods. The ship can only transport goods of a certain weight in order to execute the transport without issues. In the cutting stock problem the task is to cut standardized raw material into pieces such that the material waste is minimized. The portfolio selection problem models the decision on choosing good investments.

The knapsack problem also appears as a sub-problem in many other areas. A very general approach to linear optimization problems in practical application is to search for the bottleneck constraint while ignoring the other constraints. The bottleneck constraint can be modelled by the weight of the knapsack and an instance of the knapsack problem is obtained. In this way, the knapsack problem yields a solution which then has to be adjusted to satisfy the ignored constraints which could have been violated.

In this thesis, the knapsack problem is regarded as an incremental optimization problem. An incremental optimization problem is a kind of optimization problem where we have multiple solutions which depend on each other. These solutions are grouped to a chain of solutions in which the order matters. In knapsack variants each solution in the chain has

to contain all items of the previous solutions. The profit of the chain at a fixed capacity is the maximum profit of solutions whose weight does not exceed the capacity. The capacity threshold is unknown at the time when the chain is computed. Thus, a chain has to yield good solutions for every possible capacity. Some applications may be more realistically represented by incremental models, for example use cases where it is not possible to take out an already added item from the knapsack.

The incremental knapsack problem has similar but slightly different applications. In the shipment problem the task is to choose containers without knowing the capacity. So the containers are put onto a ship until the next container does not fit. We have a set of different ships with different weight capacities and containers have to be packed in a previous fixed order onto the next ship that arrives. Then the ship is filled with as much goods as possible without exceeding the ship's capacity. The task is to choose an order of containers so that the shipment is efficient without knowing how much capacity the arriving ship yields. A reasonable assumption is that all containers will fit onto any of the ships. Then each ship is able to load at least the first container. Larger containers receive a special treatment and will not be considered by standard shipment.

The cutting stock problem equivalent would be the unknown size of the raw material and trying to cut pieces until the residual material is not sufficient. In our incremental problem one is not allowed to further test different options once a stop signal is received. If the work on the material fails, then trying to do something else with it is not possible because the residual material may be damaged.

1.1 Related Work

The knapsack problem has many different variants ranging from multidimensional knapsacks to multiple knapsacks. The knapsack problem is known to be $\mathcal{NP}$-complete and it has a fully polynomial time approximation scheme (FPTAS) [2, 3].

The online knapsack problem is a related problem. The online problem has to make decisions which are irreversible since once an item is packed it can not be unpacked. This is very similar to incremental variants since items which are packed once have to be included in later solutions as well. The online problem has the constraint that the items are revealed one at a time and the algorithm has to decide to pack it immediately or reject it completely. This does not hold for the incremental problem and thus the incremental problem is easier to solve in general. Marchetti-Spaccamela and Vercellis show that the online knapsack problem is not solvable in the general sense and therefore research focuses

on a range of different variations which are solvable [4]. In this thesis we show that there is no solution on general instances for the incremental knapsack problem and thus we also modify the problem by adding a relaxation to the problem.

A strongly related problem is the knapsack problem with unknown capacity which has been researched by Disser et. al [5]. This problem utilizes so-called universal knapsack policies which are permutations of items. The permutation of items is the order in which items are packed into the knapsack. Once the permutation is fixed the capacity is chosen. Then the solution is determined in the following way. If adding the item does not violate the capacity constraint it is packed. Otherwise the next item is tried to be packed. This continues until there are either no more items left or the capacity is exhausted. This is a related problem since a permutation of items can be regarded as as a maximal chain which is an alternative characterization of incremental problems. The difference is that a chain of the incremental problem is not allowed to test other items. So if a solution is feasible and the next is not, then the feasible solution is returned without further modifications.

Any algorithm for incremental problems is also a working algorithm for the unknown capacity problem while this is not true the other way around. Despite this, the lower bounds on the competitive factor found in [5] still hold for the variants of the incremental problem studied in this thesis. The competitive ratio for the unknown capacity problem is 2 for general instances and $\frac{1+\sqrt{5}}{2} \approx 1.62$ for unit density instances [5].

1.2 Problem Definitions

1.2.1 Knapsack Problem

An instance of the (single) knapsack problem consists of a set of items $\mathcal{I} := [n] = \{1, 2, ..., n\}$ where n defines the the number of items and t defines the capacity threshold. Each item i has a positive profit p_i and a positive weight w_i. The ratio $\frac{p_i}{w_i}$ of profit to weight of an item i is called density or efficiency. The knapsack problem can be formally defined as selecting a subset of items $S \subseteq \mathcal{I}$ such that the total profit is maximized and the total weight does not exceed the capacity threshold.

An equivalent formulation of the knapsack problem can be given as a linear integer program

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} p_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq t \\ & \text{and } \forall i \in [n] : x_i \in \{0, 1\} \end{aligned}$$

where x is an indicator vector on the set of items $\mathcal{I}$. Here, x_i can be read as a Boolean variable whether item i is packed into the knapsack. Formally speaking $x_i = 1$ if and only if $i \in S$ where S is a subset of items. It follows that the vector x induces a subset of items $S = \{i \in [n] \mid x_i = 1\}$ consisting of the packed items.

The optimal solution will be referred to as OPT and if it is clear from the context OPT will also be the profit of the optimal solution.

When there is a fixed capacity t, one can give a so called competitive factor for any solution S which is defined as

$$\frac{\mathrm{p}(OPT)}{\mathrm{p}(S)} = \frac{\mathrm{p}(OPT)}{\sum_{i \in S} p_i} = \frac{\mathrm{p}(OPT)}{\sum_{i=1}^{n} p_i x_i}$$

depending on whether we have a set of items S or an indicator vector x. The competitive factor is a known concept in the theory of approximation and online algorithms. In both approximation and online algorithms the algorithm will not always give an optimal solution but a good alternative to it. The ratio of the optimal solution to the solution of the algorithm is called the competitive factor. Instances of incremental problems in general do not have an optimal solution. Thus, we use the competitive factor to measure the quality of the algorithm.

There is a common relaxation of the problem which is the fractional variant of the knapsack problem. The fractional variant does not consist of an indicator vector x such that each x_i is either 1 or 0 but a n-dimensional vector $x \in [0, 1]^n$. This variant of the knapsack problem is less restrictive and offers an infinite set of possible solutions. An optimal solution of the fractional knapsack problem is called $FOPT$.

The fractional problem can be solved optimally by sorting the items by density and packing them in that order until the capacity is reached. A more efficient way is to reduce the knapsack problem to a weighted median search. The goal is to find the weighted median given an order of items. The median then empowers us to find all items with at least the density of the median and the most dense item of the other items in linear time. Using a

linear time algorithm for the weighted median search yields the wanted result. Details can be found in Chapter 17 in [3].

1.2.2 Multiple Knapsacks Problem

Another problem that we will discuss is the multiple knapsacks problem. The difference to the knapsack problem is that the items are not chosen for one but for m knapsacks. So $\mathcal{I} = [n]$ is the set of items to be used and each item can occur in at most one knapsack. We require that any instance consist of more items than knapsacks and that the profit p_i and weight w_i of each item is positive. The task is to maximize the total profit over all knapsacks while no knapsack consists of items whose weight exceeds the capacity threshold t. This definition is slightly simplified since for all knapsacks the threshold is the same. The linear integer problem is the following

$$\begin{aligned} \text{maximize } & \sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{i,j} \\ \text{subject to } \forall i \in [m] : & \sum_{j=1}^{n} w_j x_{i,j} \leq t \\ \text{and } \forall j \in [n] : & \sum_{i=1}^{m} x_{i,j} \leq 1 \\ & \forall i,j \in [n] : x_{i,j} \in \{0,1\} \end{aligned}$$

where $x_{i,j}$ is the binary variable stating item i is used in knapsack j.

Replacing the constraint $\forall i,j \in [n] : x_{i,j} \in \{0,1\}$ with $\forall i,j \in [n] : x_{i,j} \in [0,1]$ gives us the definition of the fractional multiple knapsacks problem.

For convenience we refer to a solution with $S \subseteq \mathcal{I}$ and to the items of that solution in knapsack l with S^l given $l \in [m]$. Then, a solution S is a partition of the items split into different S^l. Formally that is $\cup_{l=1}^{m} S^l = S$ and $\forall i \neq j : S^i \cap S^j = \emptyset$.

1.2.3 Incremental Problems

The problem definitions for incremental optimization problems are the same as used by Lin et. al. in [6]. For an optimization problem Π we have a universe of solutions U, a benefit function $\text{ben}(\cdot)$ to determine how worthy a solution is and a cost function $\text{cost}(\cdot)$ to restrict the set of solutions. In maximization problems we want to maximize the benefit while

taking only solutions whose cost is below some chosen threshold. In terms of knapsack problems the function $\text{ben}(S) := \text{p}(S) = \sum_{i \in S} p_i$ is the sum of all profits and the function $\text{cost}(S) := \text{w}(S) = \sum_{i \in S} w_i$ is the sum of all weights considering a subset of items $S \subseteq \mathcal{I}$.

All incremental variants will not have a fixed capacity threshold. Therefore the result of an algorithm is not a single solution but a set of solutions. The solutions in a chain are monotone non-decreasing in both the ben and the cost function. The solutions in a chain are related to each other and depend on their predecessors. If we had no constraints on the chain, the problem would reduce to solving multiple instances of the classic problem independently. The relation is that every solution is an extension of its predecessor. So a next solution is a superset of the items of all previous solutions.

Formally speaking we have a partial order $\prec$ on U and monotonically non-decreasing functions ben and cost. For the knapsack problem we require the subset property such that

$$S' \prec S : \Longleftrightarrow \; S' \subset S$$

and thus,

$$S' \subset S \Rightarrow \text{p}(S') < \text{p}(S) \wedge \text{w}(S') < \text{w}(S)$$

since all profits are positive and $\text{p}(S) = \text{p}(S') + \text{p}(S/S') > \text{p}(S')$ and for $\text{w}(\cdot)$ analogously.

Definition 1.1. *A solution to the incremental knapsack problem is a* chain $\mathcal{C}$ *of* $k \in \mathbb{N}$ *solutions such that* $\mathcal{C}_0 \preceq \mathcal{C}_1 \preceq \mathcal{C}_2 \preceq ... \preceq \mathcal{C}_{k-1}$ *and* $\mathcal{C}_0 = \emptyset$.

It follows that for two solutions $i < j$ the solution $\mathcal{C}_i$ precedes $\mathcal{C}_j$ and that the set of items of solution $\mathcal{C}_j$ is a superset of the items of $\mathcal{C}_i$.

Then we can define the competitive ratio of $\mathcal{C}$ as

$$\text{ratio}'(\mathcal{C}) = \sup_{0 < t \leq \max\{\text{w}(u) | u \in U\}} \frac{\text{p}(OPT(t))}{\text{p}(\mathcal{C}(t))}$$

where $\max\{\text{cost}(u) \mid u \in U\}$ is the maximum cost of any solution and $\mathcal{C}(t)$ is the most profitable solution in the chain whose cost does not exceed the threshold t. Since an incremental problem does not have a single fixed capacity t, the optimal solution OPT is not a solution but a function $OPT(t)$ which yields the optimal solution in the knapsack problem given a capacity t.

For randomized algorithms the definition is different, since there is no concrete chain. Thus, we use the expected value of the chain. It is therefore defined as

$$\text{ratio}'(\mathcal{C}) = \sup_{0 < t \leq \max\{\text{w}(u) | u \in U\}} \frac{\text{p}(OPT(t))}{\text{p}(\mathbb{E}[\mathcal{C}(t)])}$$

where $\mathrm{p}(\mathbb{E}[\mathcal{C}(t)])$ is an abuse of notation. The output $\mathcal{C}$ of a randomized algorithm is a probability distribution over all incremental chains. Then, the expected value of $\mathcal{C}$ is the total value of all chains at the capacity t multiplied by the probability that the chain is chosen.

We will see at Section 2.1 that the definition of ratio$'$ has issues. There is no algorithm with a constant competitive factor using that definition. We will instead research the problem when capacities are reasonably large. That means we exclude capacities were not every item would fit alone into the knapsack. Formally, we introduce the modified competitive ratio which will be used throughout the whole thesis with exception of Section 2.1.

Definition 1.2. *The* modified competitive ratio *of the knapsack problem is defined as*

$$\mathrm{ratio}(\mathcal{C}) = \sup_{\max w_i \leq t \leq \sum_{i=1}^{n} w_i} \frac{\mathrm{p}(\mathcal{C}(t))}{\mathrm{p}(OPT(t))}$$

where $\sum_{i=1}^{n} w_i$ *is the maximum cost of all solutions and* $\max w_i$ *is the maximum cost of all one item solutions. Solutions* $\mathcal{C}(t)$ *and* $OPT(t)$ *are defined as for* ratio$'$.

For randomized algorithms the competitive ratio is defined as

$$\mathrm{ratio}(\mathcal{C}) = \sup_{\max w_i \leq t \leq \sum_{i=1}^{n} w_i} \frac{\mathrm{p}(\mathbb{E}[\mathcal{C}(t)])}{\mathrm{p}(OPT(t))}$$

using the same notation as in the previous definitions.

We do not mention the capacity t and the profit function explicitly when the context is clear. The only incremental problems we will deal with are knapsack problems.

It is possible to define a fractional variant of the incremental problem but that problem is solved using the same algorithm as the fractional knapsack problem. A solution for the fractional knapsack problem can be transformed into a solution for the fractional incremental problem. The major difference is that a fractional chain has a solution for every capacity. That means that for any real interval of capacity possibilities the chain has an infinite number of solutions.

1.2.4 Maximal Chains

An important part of the analysis of incremental problems in the context of knapsacks are maximal chains.

Definition 1.3. *A chain $\mathcal{C}$ is a* maximal chain *if and only if the number of solutions equals $n+1$ where n is the number of items of the instance.*

We observe that every chain has at most $n+1$ elements given an item set $\mathcal{I}$ with n items.

The cardinality of each set has to be unique and thus each set's cardinality has to be a number between 0 and n. The reason for that is that the cardinality is always non-negative and each set has to be a subset of $\mathcal{I}$ which has n elements. So $n+1$ is the total of number of different cardinalities and thus the maximal length of every chain. For a chain with maximal length it holds that $\mathcal{C}_0 = \emptyset$ and $\mathcal{C}_{n+1} = \mathcal{I}$, since the start of the chain has to be the empty set and the end of the chain has to be the full set of items.

Every given chain $\mathcal{C}$ of length $k < n+1$ can be replaced by a chain $\mathcal{C}'$ of length $k+1$. If $\mathcal{C}_k$ is not $\mathcal{I}$ we just add $\mathcal{I}$ as the $k+1$-th solutions. If $\mathcal{C}_0$ is not $\emptyset$ we insert $\emptyset$ as the first solution. Otherwise we need to find a solution i such that $\mid \mathcal{C}_{i+1}/\mathcal{C}_i \mid > 1$. Then, let x be an arbitrary item from the set $\mathcal{C}_{i+1}/\mathcal{C}_i$. For such x it holds that

$$\mathcal{C}_i \preceq \mathcal{C}_i \cup x \preceq \mathcal{C}_{i+1}$$

since $\mathcal{C}_i$ is a subset of $\mathcal{C}_i \cup x$ which is a subset of $\mathcal{C}_{i+1}$. For two solutions A, B with $A \preceq B$ we know that the profit of A is smaller than the profit of B.

For every capacity a chain uses the most profitable solution so the competitive factor will not decrease by adding in more solutions. Since the new solution satisfies the partial ordering, we can add it into the old chain without violating the incremental chain definitions.

A new chain is constructed by

$$\mathcal{C}' := (\mathcal{C}_0, \mathcal{C}_1, ..., \mathcal{C}_i, (\mathcal{C}_i \cup x), \mathcal{C}_{i+1}, ..., \mathcal{C}_k).$$

This construction can be repeated multiple times. After at most n applications of this construction the result will be a maximal chain with a competitive factor at least as good as the first chain.

There are two nice properties of maximal chains which are useful in proofs. The first is that the cardinality of the i-th solution of a maximal chain $\mid \mathcal{C}_i \mid$ is i. Then, $\mathcal{C}_0$ is always the empty set and $\mathcal{C}_n$ is the set of all items of the instance by definition.

The second property is an alternative characterization of chains. One can identify the insertion order of items since the difference between two solutions is exactly one item. A permutation of items is an equivalent representation of a maximal chain. So we can

produce a maximal chain given the permutation of items and also reverse that. For a maximal chain the permutation is unique and fixed. For a chain, which is not maximal, there can be multiple permutations which are as good as the chain. For a maximal chain we know which item is inserted at first into the knapsack.

As a general note we ignore the capacities 0 and $\sum_{i=1}^{n} w_i$ in the analyses. All maximal chains and all chains created by our algorithms start with the empty set $\emptyset$ and end with the complete set $\mathcal{I}$. That means at capacity 0 the chain has not packed anything and that is also true for the optimal solution. For capacity $\sum_{i=1}^{n} w_i$ the chain and the optimal solution at that capacity will pack all items and thus the competitive ratio is 1 here.

1.3 Knapsack Approximation Algorithm

For the knapsack problem, there exists a number of approximation algorithms. We now introduce a well-known algorithm for approximating the knapsack problem from Chapter 2 in [7]. The algorithm will be presented with a modification which has no effect on the competitive factor.

Algorithm 1 is called ApproxKP and takes as input item set $\mathcal{I}$ and capacity threshold t. The profits and weights of an item i are accessible by p_i and w_i for item i. Some ideas of ApproxKP will carry over to other algorithms. It is used as a subroutine in algorithm DetIMKP which we present in Chapter 3.

Algorithm ApproxKP yields a competitive factor of 2 compared to the optimal solution and also yields a factor of 2 compared to the optimal fractional solution. The idea of the algorithm is to compute two solutions and choose the one with higher profit. The first solution is computed by sorting the items by descending density and pack as many items in this order as possible without exceeding the capacity threshold. The second solution consists of the single next item which the first solution could not fit anymore.

The solution consisting of the most dense items is called the greedy solution. The solution consisting of the next single item is called the swap item solution.

Definition 1.4. *An item s is a* swap item *if and only if $p_s > \sum_{i=1}^{s-1} p_i$.*

The check for swap items is important since on the instance $p_1 = 1$, $w_1 = 0.5$, $p_2 = t$, $w_2 = t$ the greedy solution at capacity t would yield a competitive factor of t. Since t is arbitrary, the greedy solution can become arbitrarily bad.

Algorithm 1 ApproxKP$(\mathcal{I}, t)$

```
1: Sort items by descending density
2: W, P ← 0
3: for i = 1 to n do
4:     W ← W + w_i
5:     if t < W then
6:         break
7:     else
8:         P ← P + p_i
9:     end if
10: end for
11: if P < p_i then
12:     return {i}
13: else
14:     return {1, 2, ..., i − 1}
15: end if
```

We now show that the algorithm has at least half of the profit of the fractional optimal solution. Since the fractional optimal solution is never worse than the optimal solution, the algorithm is also 2-competitive.

Lemma 1.5. *The ratio of* ApproxKP *to FOPT is 2.*

Proof. We assume that all items are already sorted by density as done in line 1 of ApproxKP. $FOPT$ is the optimal solution to the fractional knapsack problem. As already mentioned earlier the solution $FOPT$ consists of the greedy solution and a fraction of the most dense item which is not part of the greedy solution. Let k be the index of that next item. Then, $FOPT$ has profit of at most $\sum_{i=1}^{k} p_i$ since the item p_k is not completely used in $FOPT$. ApproxKP has a profit of $\max\{\sum_{i=1}^{k-1} p_i, p_k\}$.

Then the factor is

$$\frac{\mathrm{p}(FOPT)}{\mathrm{p}(\textsc{ApproxKP})} < \frac{\sum_{i=1}^{k} p_i}{\max\{\sum_{i=1}^{k-1} p_i, p_k\}} \leq \frac{\sum_{i=1}^{k} p_i}{\frac{1}{2}\left(\sum_{i=1}^{k-1} p_i + p_k\right)} = \frac{2\sum_{i=1}^{k} p_i}{\sum_{i=1}^{k} p_i} = 2$$

using $\max\{a, b\} = \frac{1}{2}\max\{a, b\} + \frac{1}{2}\max\{a, b\} \geq \frac{1}{2}a + \frac{1}{2}b = \frac{1}{2}(a + b)$. □

We now show that this factor is tight. The instance has two items which are defined as $p_1 = p_2 = w_1 = w_2 = 1$ and a capacity of $2 - \delta$. For any $0 < \delta < 1$, the factor is $2 - \delta$ since $FOPT$ has as much profit as capacity and any integer solution has a profit of 1. The factor 2 of the algorithm is therefore optimal.

Chapter 2

Incremental Knapsack Problem

In this chapter, we first show a result about issues of incremental knapsack problems. It becomes impossible to construct an algorithm which is competitive against the optimal solution. Due to this, we modify the competitive factor of incremental problems.

This proof is of general purpose and includes the multiple knapsacks variants as well. For the incremental knapsack problem we show an algorithm achieving a competitive factor of 2 and also that this competitive factor is tight. Then, we examine the setting in which we only consider a subset of instances with items of unit density. We show that the lower bounds of the competitive factor for the unknown capacity problem can be applied to the incremental knapsack problem. Here, we yield a competitive factor of approximately 1.62.

2.1 Issues of Incremental Knapsack

We start by proving that no algorithm achieves a constant competitive factor. The idea is that any algorithm performs bad on some capacity depending on the algorithm starting with low weight items or high weight items.

Lemma 2.1. *For any $\alpha > 0$, there exists an instance of the incremental (multiple) knapsack problem where no deterministic algorithm achieves a competitive factor of α.*

Proof. Let m be the number of knapsacks. Assume for contradiction there is an algorithm achieving a competitive factor α.

The instance $\mathcal{I}$ consists of $m+1$ items such that an item i has weight $w_i = i$. The profit of item $i = 1$ is also 1. The profits for items $i > 1$ are recursively defined by $p_i = 1+\alpha \sum_{j=1}^{i-1} p_j$. Observe that for all items $i > j$ the weight w_i is greater than the weight w_j. Furthermore,

the profit p_i is more than α times bigger than the profit of all items with smaller weight added up. That forces a solution to include the biggest available item to be α-competitive for every capacity.

Let chain $\mathcal{C}$ be the output of the algorithm on that instance. We assume $\mathcal{C}$ to be a maximal chain and can therefore identify in which order items are packed into the knapsacks. We have a look at each of the m knapsacks and construct a set M consisting of exactly those items which are packed first into any knapsack. Then, we can determine an item which is missing in M and denote it by $x \in \mathcal{I}/M$. Since there are more items than knapsacks, there exists such x.

For $x = 1$, there will be no item packed at capacity $t = 1$ compared to OPT. Since OPT has profit $1 > 0$ the solution S is not competitive at all. This is a contradiction.

For $x \neq 1$, we examine the solution of the chain at capacity $t = w_x$. We know that OPT will contain item x in a knapsack and thus has at least a profit of p_x. In the examined solution, we know by our definition of x that it is not packed as a first item in any knapsack. After putting any item into a knapsack there is not enough capacity left over to also include x since $w_i + w_x > w_x = t$. Thus, item x is not included in solution $\mathcal{C}(t)$. The profit of $\mathcal{C}(t)$ is at most the profit of all items with weight smaller than w_x. Any item with index $i > x$ has more weight than the capacity and item x itself does not fit. This yields the upper bound $\sum_{j=1}^{m-1} p_j$ on the profit of $\mathcal{C}$. The competitive ratio is then

$$\frac{\mathrm{p}(OPT(t))}{\mathrm{p}(\mathcal{C}(t))} \geq \frac{p_x}{\sum_{j=1}^{x-1} p_j} > \frac{\alpha \sum_{j=1}^{x-1} p_j}{\sum_{j=1}^{x-1} p_j} = \alpha$$

which completes the proof for deterministic algorithms.

□

In that proof we use exactly one item more than there are knapsacks. On instances with n items and $m \geq n$ knapsacks the problem can be solved optimal by using a different knapsack for each item. The number of items in the example is minimal.

A similar result can be achieved for randomized algorithms using the same idea.

Lemma 2.2. *For every $\alpha > 0$, there exists an instance of the incremental (multiple) knapsack problem where no randomized algorithm achieves a competitive factor of α.*

Proof. Let m be the number of knapsacks. Assume for contradiction there exists an algorithm achieving a competitive factor of α.

The instance $\mathcal{I}$ consists of $n = \lceil 2\alpha m + m \rceil$ items which are defined as $p_1 = 1$ and $p_i = 2\alpha \sum_{j=1}^{i-1} p_j$ for $i > 1$ and $w_i = i$.

The output of the algorithm is a randomized chain $\mathcal{C}$. We now assume to know the probabilities of the algorithm's output. Let C be an arbitrary maximal chain, then we say that $\Pr[\mathcal{C}(t) = C]$ is the probability of C to be the chain $\mathcal{C}$. The term $\sum_C$ means to sum over all maximal chains using the item set $\mathcal{I}$. The term $\sum_{C|i \text{ first}}$ means to sum over all maximal chains, in which the item i is packed first in any knapsack, using the item set $\mathcal{I}$. Using this knowledge we can calculate for each item i the probability q_i that this item is one of the items to be packed first by $\mathcal{C}$ by

$$q_i = \sum_{C|i \text{ first}} \Pr[\mathcal{C}(t) = C]$$

which can be done with the same reasoning as in Lemma 2.1.

The sum over all probabilities is $\sum_C \Pr[\mathcal{C}(t) = C] = 1$ by probability laws. There are m knapsacks, so at each chain there can be at most m items which are packed first. This gives us

$$\sum_{i=1}^{n} q_i = \sum_{i=1}^{n} \sum_{C|i \text{ first}} \Pr[\mathcal{C}(t) = C] = \sum_{i=1}^{m} \sum_{C} \Pr[\mathcal{C}(t) = C] = m$$

since summing over q_i uses each chain C exactly m times.

We consider item x which is the least probable initial item over all chains, formally defined as $x = \arg\min_{i \in [n]} q_i$. Then, q_x can be upper bounded by

$$q_x \leq \frac{\sum_{i=1}^{n} q_i}{n} \leq \frac{m}{2\alpha m + m} = \frac{1}{2\alpha + 1} < \frac{1}{2\alpha}$$

which is used in the final computation of the competitive factor.

At capacity $t = w_x$, all items $j \leq x$ can be filled into knapsacks but not a single item $j > x$. The optimal solution consists of at least item x and thus has at least a profit of p_x.

We want to upper bound the expected profit of solution $\mathcal{C}(t)$. The major step is to split the chains C into two groups. One group has x as the initial item and the other does not. For a chain C without item x as the first item we have $\mathrm{p}(C(t)) \leq \sum_{i=1}^{x-1} p_i$ and for a chain with item x we have $\mathrm{p}(C(t)) \leq \sum_{i=1}^{x} p_i$. We use the same argumentation as in Lemma 2.1. Then, we see that both bounds consist of $\sum_{i=1}^{x-1} p_i$. The difference between both is that the second type of chains has an additional profit p_x.

The expected profit is

$$\begin{aligned}
\mathrm{p}(\mathbb{E}[\mathcal{C}(t)]) &= \sum_C \Pr[\mathcal{C}(t) = C] \cdot \mathrm{p}(C(t)) \\
&\leq \left(\sum_C \Pr[\mathcal{C}(t) = C] \cdot \sum_{i=1}^{x-1} p_i \right) + \left(\sum_{C|i \text{ first}} \Pr[\mathcal{C}(t) = C] \cdot p_x \right) \\
&= \left(\sum_{i=1}^{x-1} p_i \cdot \sum_C \Pr[\mathcal{C}(t) = C] \right) + \left(p_x \cdot \sum_{C|i \text{ first}} \Pr[\mathcal{C}(t) = C] \right) \\
&= \sum_{i=1}^{x-1} p_i + p_x \cdot q_x \\
&< \sum_{i=1}^{x-1} p_i + 2\alpha \cdot \sum_{i=1}^{x-1} p_i \cdot \frac{1}{2\alpha} \\
&= 2 \sum_{i=1}^{x-1} p_i.
\end{aligned}$$

Now we just put together the lower bound for OPT and the upper bound for $\mathcal{C}$ yielding a competitive factor of

$$\begin{aligned}
\frac{\mathrm{p}(OPT)}{\mathrm{p}(\mathbb{E}[\mathcal{C}])} &> \frac{p_x}{2 \sum_{i=1}^{x-1} p_i} \\
&= \frac{2\alpha \sum_{i=1}^{x-1} p_i}{2 \sum_{i=1}^{x-1} p_i} \\
&= \alpha
\end{aligned}$$

which shows the claim for randomized algorithms. □

For $k = 1$, Lemma 2.1 and Lemma 2.2 show that the incremental knapsack problem has no competitive solution. For that reason we will use the modified competitive ratio in the incremental setting to solve instances in general.

The instance in Lemma 2.2 consists of a lot more items than the one in Lemma 2.1. The number of items is not constant but depends on the competitive factor. The lemma has no impact on the existence of a non-constant competitive factor. Thus, we can try to give an algorithm with a competitive factor depending on the number of items.

We will present a simple randomized algorithm NonConstIKP and let $\mathcal{C}$ be the random chain returned by the algorithm. This chain will consist of exactly three solutions. Only one solution in the chain is affected by randomization. In the proof we show that for all

capacities there is a small probability that the chain $\mathcal{C}$ will obtain the same profit as the optimal solution.

Algorithm 2 NonConstIKP($\mathcal{I}$)

1: $S \leftarrow 2^{\mathcal{I}}/\{\emptyset, \mathcal{I}\}$
2: Choose $x \in S$ uniform random
3: **return** $\mathcal{C} = (\emptyset, x, \mathcal{I})$

Lemma 2.3. NonConstIKP *has a competitive factor of* 2^n.

Proof. S consists of all solutions which are not the empty set or the set $\mathcal{I}$. Let capacity t be arbitrary and OPT the optimal solution for that capacity. We differentiate cases whether OPT is trivial or not.

Case 1: $t = 0 \vee t = \sum_{i=1}^{n} w_i$
Both capacities are trivial for both $OPT(t)$ and $\mathcal{C}(t)$. As both solutions are identical the competitive factor is $1 < 2^n$.

Case 2: $0 < t < \sum_{i=1}^{n} w_i$
We know that the optimal solution $OPT(t)$ is neither the set $\mathcal{I}$ nor the empty set. So the optimal solution lies in S as the chain does in every case. The optimal solution could only lie outside of S if the capacity is 0 or $\sum_{i=1}^{n} w_i$. The probability that $\mathcal{C}(t)$ equals a fixed element from S is $\frac{1}{2^n-2}$. So it holds

$$\mathrm{p}(\mathbb{E}[\mathcal{C}(t)]) = \sum_{C} \Pr[\mathcal{C}(t) = C] \cdot \mathrm{p}(C) \geq \Pr[\mathcal{C} = OPT] \cdot \mathrm{p}(OPT) > 2^{-n}\,\mathrm{p}(OPT)$$

and the competitive factor is

$$\frac{\mathrm{p}(OPT)}{\mathrm{p}(\mathbb{E}[\mathcal{C}(t)])} \leq \frac{\mathrm{p}(OPT)}{2^{-n} \cdot \mathrm{p}(OPT)} = 2^n.$$

□

We just showed that a randomized algorithm can be competitive although the competitive factor is not a constant. We will not continue using the definition of incremental problems using ratio$'$ because we want to have a constant competitive factor. The major problem is that our algorithm is forced to always include the item with the lowest weight as soon as that becomes possible. Thus, the packing order is partially fixed and high weight items with high density are a problem.

Further we normalize every instance for our algorithms such that $\max\{w_i \mid 1 \leq i \leq n\} = 1$. This can be done by determining the weight $\max\{w_i \mid 1 \leq i \leq n\}$ and divide every item profit and item weight by it.

2.2 Algorithms

The first algorithm we present is a simple algorithm using randomization with a competitive factor of 2. The second algorithm is a deterministic algorithm which also has a competitive factor of 2.

2.2.1 Randomized Algorithm

The randomized algorithm RandIKP is based on a simple idea. We have already seen that there is an approximation algorithm with a competitive factor of 2 where the greedy solution and the swap item solution are computed and the better of both is returned. This approach does not exactly carry over to the incremental problem. We can describe the greedy solution as a chain, but we cannot describe the swap item as a chain since it depends on the capacity. The capacity is unknown when the chain is computed.

The greedy solution can be computed as a chain as done in RandIKP. The swap item is replaced by the item with the most profit. So the solution which includes the item with the highest profit has at least as much profit as the swap item. Since the competitive ratio definition gives us the opportunity to always include one item into a solution, that solution is also feasible.

Algorithm 3 RandIKP($\mathcal{I}$)

1: Sort items by descending density
2: $j \leftarrow \arg\max_i p_i$
3: $A \leftarrow (\emptyset, \{j\}, \mathcal{I})$
4: $B \leftarrow (\cup_{i=1}^{0}\{i\}, \cup_{i=1}^{1}\{i\}, ..., \cup_{i=1}^{n}\{i\})$
5: **return** A and B each with probability $\frac{1}{2}$

Lemma 2.4. RandIKP *is* 2*-competitive.*

Proof. Let capacity t be arbitrary such that $0 < t < \sum_{i=1}^{n} w_i$ and let $\mathcal{C}$ be a chain.

The solution retrieved by B consists of the most dense items which are filled into the knapsack until no more item fits which is essentially the greedy solution for that capacity.

The solution from chain A is $\{j\}$ since $w_j \leq 1 \leq t$ holds. For that solution we obtain $p_j > p_i$ for all i. k is the number of items in the greedy solution and all our indices are sorted according to the algorithm.

Our fractional optimal solution is bounded by the profit of the first $k+1$ items.

The expected profit of $\mathcal{C}(t)$ is

$$\mathrm{p}(\mathbb{E}[\mathcal{C}(t)]) \geq \frac{1}{2}p_j + \frac{1}{2}\sum_{i=1}^{k} p_i \geq \frac{1}{2}\sum_{i=1}^{k+1} p_i \geq \frac{1}{2}\,\mathrm{p}(FOPT) \geq \frac{1}{2}\,\mathrm{p}(OPT)$$

□

In general, the algorithm will not achieve a competitive ratio of $2-\delta$ for any $\delta > 0$. Let $N = \lceil \frac{4}{\delta} \rceil$ and $\mathcal{I}$ an instance with $N+1$ items. For $i \leq N$ we define item i as $(\frac{1}{N}, \frac{1}{N})$ and item $N+1$ as $(\frac{2}{N}, 1)$. Then, RandIKP will have chain A containing item $N+1$ and chain B containing the first N items with profit $\frac{1}{N}$ each, yielding

$$\mathrm{p}(\mathbb{E}[\mathcal{C}(t)]) = \frac{1}{2}\cdot\frac{2}{N} + \frac{1}{2}\cdot N \cdot \frac{1}{N} = \frac{1}{2} + \frac{1}{N}.$$

Whereas the optimal solution uses only the first N items, and therefore achieving a profit of 1 which is exactly the greedy solution. Our competitive factor is

$$\frac{\mathrm{p}(OPT)}{\mathrm{p}(\mathbb{E}[\mathcal{C}])} = \frac{1}{\frac{1}{2}+\frac{1}{N}} = \frac{\frac{2}{2}+\frac{2}{N}}{\frac{1}{2}+\frac{1}{N}} - \frac{\frac{2}{N}}{\frac{1}{2}+\frac{1}{N}} > 2 - \frac{4}{N} \geq 2 - \frac{4}{\frac{4}{\delta}} = 2-\delta.$$

The best competitive factor the algorithm can achieve is 2.

2.2.2 Deterministic Algorithm

After seeing a simple randomized algorithm we now present a deterministic algorithm. Disser et. al. present the algorithm Universal which shares some ideas of the algorithm presented in this thesis [5]. Both algorithms yield a competitive factor of 2 in the context of their respective problems. Universal solves the unknown capacity problem and thus the analysis is quite different from what we do here. That algorithm does not work on the incremental knapsack problem but with a small modification it could achieve a competitive factor of 2. We present the algorithm DetIKP which is in some regard a simplified version of the algorithm presented by Disser et. al. because in the unknown capacity problem also positive capacities below 1 are considered.

Swap items are a concept that we have already seen. A swap item s has more profit than all items with higher density in total. The item with the highest density is always a swap item because there are no items with higher density. The algorithm will sort the items first and then compute the swap item with the highest index which is by definition also the one with the greatest profit. The solution of the chain for low capacities will be good since the swap item solution is 2-competitive. This also holds for high capacities since the greedy solution is 2-competitive.

Algorithm 4 DetIKP($\mathcal{I}$)

```
Sort items by descending density
l ← 1
P ← p_1
for i = 2 to n do
    if p_i > P then
        l ← i
    end if
    P ← P + p_i
end for
C_1 ← l
C_i ← ∪_{j=1}^{i+l-2} {j} for i = 2 to n − l + 2
return C
```

Theorem 2.5. DetIKP *is 2-competitive and has a run time of* $\mathcal{O}(n \log n)$.

Proof. As in algorithm DetIKP, let l be the index of the greatest swap item. Let capacity t be arbitrary such that $0 < t < \sum_{i=1}^{n} w_i$ and determine $k \in [n]$ such that $\sum_{i=1}^{k} w_i \leq t < \sum_{i=1}^{k+1} w_i$.

Case 1: $k < l$

For $k < l$ we can upper bound the optimal solution by $\sum_{i=1}^{k+1} p_i$. The optimal solution has at most as much profit as $FOPT$ and $FOPT$ has at most as much profit as the first $k+1$ items.

For $\mathcal{C}$ we can at least use solution $\mathcal{C}_1$ since the solution has only one item and the capacity is at least 1, that means any one item solution fits. Thus, it follows

$$2\,\mathrm{p}(\mathcal{C}_1) = 2p_l = \mathrm{p}_l + p_l \geq p_l + \sum_{i=1}^{l-1} p_i = \sum_{i=1}^{l} p_i \geq \sum_{i=1}^{k+1} p_i \geq \mathrm{p}(OPT)$$

which shows a competitive factor of 2 for this case.

Case 2: $l \leq k < n$

The profit of the optimal solution still has the upper bound of $\sum_{i=1}^{k+1} p_i$. The chain yields

the solution $\mathcal{C}_{k-l+2} = \cup_{i=1}^{(k-l+2)+l-2}\{i\} = \cup_{i=1}^{k}\{i\}$ consisting of items 1 to k. From the definition of k it follows that $\mathcal{C}_{k-l+2}$ is a feasible solution of the chain for capacity t.

Since l is the swap item with the biggest index, the item $k+1 > l$ is not a swap item and thus $p_{k+1} \leq \sum_{i=1}^{k} p_i$. Then, the competitive factor is

$$\mathrm{p}(\mathcal{C}_{k-l+2}) = \sum_{i=1}^{k} p_i \geq \frac{1}{2}\left(\sum_{i=1}^{k} p_i + p_{k+1}\right) = \frac{1}{2}\left(\sum_{i=1}^{k+1} p_i\right) \geq \frac{1}{2}\,\mathrm{p}(OPT).$$

The run time of DetIKP is $\mathcal{O}(n \log n)$. The run time of line 1 has a worst case complexity of $\mathcal{O}(n \log n)$ using any fast sorting algorithm. The computations of line 2 and 3 are constant time operations. Lines 4 to 9 consist of a loop in which each iteration can be computed in constant time. There are at most $n-1$ iterations and thus the total run time of these lines is linear in n. The run time of lines 10-12 depends on the data structure of incremental chains. If we assume that we assign every item just the index of the solution in the chain where it appears first, this also can be done in linear time. The run time is dominated by the sorting algorithm and thus we get the total run time of $\mathcal{O}(n \log n)$. □

2.3 Unit Densities

In this chapter, we will discuss the incremental knapsack problem for a special subset of instances. These instances only consist of the items where the profit is equal to the weight, i.e. $p_i = w_i$ for each item i. We will only refer to the weights in this chapter since the profits are not interesting anymore. Informally speaking, we have a set of weights and want to use as much space of the knapsack as possible, so that the residual space is minimized.

While this variant is called unit density an alternative description would be proportional densities. Since all work on unit densities only uses the weights, every instance for which an α exists such that for all solutions it holds that

$$\mathrm{p}(S) = \alpha \cdot \mathrm{w}(S),$$

the following proofs apply to as well.

The unit density variant of the unknown capacity problem was investigated by Disser et. al. [5]. They showed a lower bound of φ and a φ-approximation algorithm with φ being $\frac{1+\sqrt{5}}{2}$. The lower bound also applies for the incremental problem but the approximation algorithm does not work. The proof of the lower bound is deferred to Section 2.4.

Before the analysis begins, we state an important equality used during this chapter which is:

$$\varphi^i + \varphi^{i-1} = (\varphi + 1) \cdot \varphi^{i-1} = \frac{1 + \sqrt{5} + 2}{2} \cdot \varphi^{i-1} = \frac{(1 + \sqrt{5})^2}{2^2} \cdot \varphi^{i-1} = \varphi^2 \cdot \varphi^{i-1} = \varphi^{i+1}.$$

We first discuss the algorithm UNIVERSAL which is φ-competitive on unknown capacity problem instances. It was proposed in [5] and does not work in our setting. The resulting permutation is based on a number of phases. The first phase starts with the smallest item and is followed by all items in order of increasing weight until one item is φ times as big as the first item. That item is the first item of the next phase. The permutation starts with the last phase and puts the items from all phases together.

We give a counter example and consider the output of UNIVERAL given the following instance. Let $0 < \varepsilon < (\sqrt{5} - 1)/4 \approx 0.309$ such that $n = \frac{1}{\varepsilon}$ is a natural number. Our instance has $n + 2$ items, where $w_1 = 1 + \varepsilon$, $w_2 = \varphi$ and $w_i = \varepsilon$ for $i > 2$. The last n items have a total weight of 1. The first phase covers all items from 3 to $n + 2$. The next biggest item would be item 1 which is smaller or equal to $\varphi \cdot \varepsilon$. So item 1 starts a new phase and item 2 is also in that phase. The resulting insertion order is therefore $(1, 2, 3, ..., n + 2)$.

Assuming we have a maximal chain $\mathcal{C}$. We choose a capacity t of $1 + \varphi$. The chain would just consist of item 1 since the solution with two items has a weight greater than the capacity as seen by

$$\mathrm{w}(\mathcal{C}_2) = w_1 + w_2 = 1 + \varepsilon + \varphi > 1 + \varphi = t.$$

Then we have a feasible solution $\mathcal{C}_1$ with weight of

$$\mathrm{w}(\mathcal{C}_1) = w_1 = 1 + \varepsilon < 1 + \frac{\sqrt{5} - 1}{4} = \frac{1}{2} \cdot \left(1 + \frac{1 + \sqrt{5}}{2}\right) = \frac{1}{2} \cdot (1 + \varphi)\,.$$

The optimal solution contains the items 2 to n and has a weight of $1 + \varphi$. The competitive factor is greater than

$$\frac{\mathrm{w}(OPT)}{\mathrm{w}(\mathcal{C})} > \frac{1 + \varphi}{\frac{1}{2} \cdot (1 + \varphi)} = 2$$

which is even worse than using the earlier presented algorithm DETIKP.

Therefore a different algorithm, called UNITIKP, is presented which is tailored to unit density instances and is φ-competitive. The algorithm sorts the items by ascending weight and then normalizes all the weights, that is $w_i = \frac{w_i}{w_n}$.

The algorithm looks at all items with weight smaller than $\varphi^{-1} \approx 0.61$ and calculates their sum. We denote the index of the biggest item with weight below φ^{-1} by a and the index of the smallest item with weight at least φ^{-1} by b. For simplicity reasons, we consider $w_0 = 0$, which results in $a = 0$ and $b = 1$ if there are no weights below φ^{-1}. If the first a items are very small compared to all other items, the algorithm ignores them and goes on with the big items. These are the ones with weight at least φ^{-1}. If the first a items have a significant total weight, the biggest item with weight 1 is packed followed by all small items in ascending order. This way, the algorithm is φ-competitive for low capacities. After that the algorithm has packed at least a weight of φ.

We observe that after packing weight φ any order of items is φ-competitive. This is the case since all items have at most weight 1. To violate the competitive factor of φ the weight of OPT has to be φ times greater than the weight of the solution returned by the algorithm. The difference between those capacities is

$$t\varphi - \varphi = \varphi(t-1) \geq \varphi(\varphi^1 - \varphi^0) = \varphi \cdot \varphi^{-1} = 1$$

and in the capacity range from t to φt there has to be another solution.

Algorithm 5 UNITIKP($\mathcal{I}$)

1: Sort items by ascending weight
2: Let $b \in [n]$ such that $w_{b-1} < \varphi^{-1} \leq w_b$
3: $a \leftarrow b - 1$
4: **if** $w_b \geq \varphi^{-1} \cdot (1 + \sum_{i=1}^{a} w_i)$ **then**
5: **return** $\mathcal{C}(b, b+1, ..., n, 1, 2, ..., a)$
6: **else**
7: **return** $\mathcal{C}(n, 1, 2, ..., n-1)$
8: **end if**

Theorem 2.6. UNITIKP *is* 2*-competitive on unit density instances and has a run time of* $\mathcal{O}(n \log n)$.

Proof. We analyze the cases based on how the algorithm decides in the condition in line 4. We then separately determine the competitive factor for the maximal chain the algorithm returns.

Case 1: $w_b \geq \varphi^{-1} \cdot (1 + \sum_{i=1}^{a} w_i)$
In this case, the main idea is to ignore the a smallest items, since they are negligible. Regarding only the big items $i \geq b$, the algorithm starts with the least weight items. This maximizes the number of big items at any capacity. Thus, the number of big items packed by the optimal solution is not greater. As each big item has at least a weight of φ^{-1}, there

can be no item with more than φ times that weight. So the algorithm is φ-competitive if only the big items count. The total weight of the small items $\sum_{i=1}^{a} w_i$ can now be added on top of the profit of OPT. Since it is that small, it does not change the outcome of the calculation too much.

Let k be the number of big items in $\mathcal{C}$. The weight of $\mathcal{C}$ is at least

$$\mathrm{w}(\mathcal{C}) \geq k \cdot w_b \geq k \cdot \varphi^{-1} \cdot \left(1 + \sum_{i=1}^{a} w_i\right) > \varphi^{-1} \left(k + \sum_{i=1}^{a} w_i\right)$$

while at the same time the weight of OPT is at most

$$\mathrm{w}(OPT) \leq k \cdot w_n + \sum_{i=1}^{a} w_i = k + \sum_{i=1}^{a} w_i$$

and thus, this gives us a competitive factor of φ.

Case 2: $w_b < \varphi^{-1} \cdot (1 + \sum_{i=1}^{a} w_i)$
In this case, the analysis will exploit the fact that all solutions of the chain have at most φ times the weight of their preceding solution. The chain starts with the biggest item n and adds all items with weight at most φ^{-1} to the knapsack. Compared to item n a single small item does not exceed the φ factor. Item b is the first big item after n is packed. Here, we know that the total weight of item n and the small items are significant. Once item b is packed, any order of items would give us a φ-competitive chain.

Let t be a capacity with $0 < t < \sum_{i=1}^{n} w_i$ and let k be an index such that $\mathcal{C}_k \leq t < \mathcal{C}_{k+1}$. Since OPT has a weight of at most the capacity, $\mathcal{C}_{k+1}$ is an upper bound for OPT.

For $k \leq a$, we have

$$\frac{\mathrm{w}(OPT)}{\mathrm{w}(\mathcal{C})} \leq \frac{\mathrm{w}(\mathcal{C}_{k+1})}{\mathrm{w}(\mathcal{C}_k)} = \frac{\mathrm{w}(\mathcal{C}_k) + w_k}{\mathrm{w}(\mathcal{C}_k)} = 1 + \frac{w_k}{\mathrm{w}(\mathcal{C}_k)} \leq 1 + \frac{\varphi^{-1}}{1} = \varphi$$

where we exploit that all items from 1 to a have at most a weight of φ^{-1} by construction and $\mathcal{C}_k \geq 1$ since the solution contains item n.

For $k = b$, the competitive factor is

$$\frac{\mathrm{w}(OPT)}{\mathrm{w}(\mathcal{C})} \leq \frac{\mathrm{w}(\mathcal{C}_{k+1})}{\mathrm{w}(\mathcal{C}_k)} = \frac{\mathrm{w}(\mathcal{C}_k) + w_b}{\mathrm{w}(\mathcal{C}_k)} \leq 1 + \frac{w_b}{1 + \sum_{i=1}^{a} w_i} \leq 1 + \varphi^{-1} = \varphi$$

where we used the inequality known from the case distinction.

For $k > b$, the lower bound for $\mathcal{C}_k$ is

$$\mathrm{w}(\mathcal{C}_k) = 1 + \sum_{i=1}^{a} w_i + \sum_{i=b}^{k-1} w_i > 1 + w_b \geq 1 + \varphi^{-1} = \varphi$$

and thus, the competitive factor is

$$\frac{\mathrm{w}(OPT)}{\mathrm{w}(\mathcal{C})} \leq \frac{\mathrm{w}(\mathcal{C}_{k+1})}{\mathrm{w}(\mathcal{C}_k)} \leq \frac{\mathrm{w}(\mathcal{C}_k) + 1}{\mathrm{w}(\mathcal{C}_k)} \leq 1 + \frac{1}{\varphi} = \varphi.$$

The run time of $\mathcal{O}(n \log n)$ can be shown by inspecting each line of the algorithm individually. The first line can be implemented in $\mathcal{O}(n \log n)$ with any fast sorting algorithm. Line 2 can be implemented in logarithmic time using binary search. Line 3 is a constant time operation and the evaluation of the condition in line 4 as well. If we assume the structure of the chain from algorithm RandIKP, the lines 5 and 7 can be computed in linear time. The total run time is $\mathcal{O}(n \log n)$ and therefore as fast as the algorithm DetIKP. □

2.4 Lower Bounds

In this section, we discuss lower bounds for the competitive factor of incremental knapsack problems. First, we start with instances of three items and give a lower bound for deterministic and randomized algorithms. Then, an instance with five items shows a lower bound for unit densities of φ. Then, we show a lower bound of the competitive factor of 2 for any deterministic algorithm in the incremental knapsack problem.

2.4.1 Deterministic Instance with Three Items

First, we show that a competitive factor better than $\sqrt{2}$ is not achievable. After that we show how to actually achieve a competitive factor of $\sqrt{2}$.

Lemma 2.7. *There exists an instance of the incremental knapsack problem where no deterministic algorithm achieves a competitive factor better than* $\sqrt{2}$.

Proof. Consider the instance $\mathcal{I} = \{1, 2, 3\}$ with $p_1 = w_1 = \sqrt{2}$ and $p_2 = w_2 = p_3 = w_3 = 1$. For OPT on capacity $\sqrt{2}$ the solution $\{1\}$ gives us a profit of $\sqrt{2}$. On capacity 2, the solution $\{2, 3\}$ gives us a profit of 2.

Let $\mathcal{C}$ be an arbitrary maximal chain. If $\mathcal{C}_1$ is the solution $\{1\}$, then at capacity 2 the solution $\mathcal{C}_2$ has weight $\sqrt{2}+1 > 2$ and will not be used. This leads to the competitive factor of

$$\frac{\mathrm{p}(OPT(2))}{\mathrm{p}(\mathcal{C}(2))} = \frac{2}{\sqrt{2}} = \sqrt{2}.$$

If $\mathcal{C}_1$ is either solution $\{2\}$ or $\{3\}$, then at capacity $\sqrt{2}$ the solution $\mathcal{C}_2$ has at least weight $1+1 = 2 > \sqrt{2}$ and thus

$$\frac{\mathrm{p}(OPT(\sqrt{2}))}{\mathrm{p}(\mathcal{C}(\sqrt{2}))} = \frac{\sqrt{2}}{1} = \sqrt{2}$$

completes the lemma. □

The $\sqrt{2}$ lower bound is tight on instances with three items. This can be shown by arguing how to achieve a competitive factor of $\sqrt{2}$ on any instance. The proof reduces to a big case analysis in which an incremental chain is explicitly stated for each case.

Lemma 2.8. *For every instance with three items, there exists a chain achieving a competitive factor of* $\sqrt{2}$.

Proof. Consider the instance $\mathcal{I} = \{1, 2, 3\}$ with arbitrary item profits and weights. The items are ordered such that $p_1 \geq p_2 \geq p_3 > 0$. We will now show that there exists a chain, such that the competitive factor is $\sqrt{2}$. We examine the weight and relative profits of the given items more closely.

The optimal solution is chosen from the set $2^{\mathcal{I}}$. For the calculation of the competitive factor we can assume an arbitrary capacity and show that the competitive factor of the optimal solution and the solution of the chain is at most $\sqrt{2}$. The equivalent alternative is to look at all optimal solutions and determine their weights. Each weight is then regarded as a capacity and the chain is compared to the optimal solution at that capacity. We show that the competitive ratio is at most $\sqrt{2}$.

The set of solutions $2^{\mathcal{I}}$ is not entirely necessary and can be reduced to a much smaller set. The solutions $\emptyset$, $\{2\}$ and $\{3\}$ have at most the profit of solution $\{1\}$ and by the definition of the competitive factor those solutions are not interesting calculation of for the competitive factor. The solution $\mathcal{I}$ uses every item. Any maximal chain includes this solution and thus the competitive factor is 1.

The solutions which are left over are $\{1\}, \{2, 3\}, \{1, 3\}$ and $\{1, 2\}$. If any of these are solutions in the chain, then the competitive factor is 1 for that capacity and the solution does not matter in determining a lower bound.

Before we go into the analysis, we introduce the notation of a maximal chain $\mathcal{C}(a,b,c)$ of solutions $\mathcal{C}_0 = \emptyset$, $\mathcal{C}_1 = \{a\}$, $\mathcal{C}_2 = \{a,b\}$ and $\mathcal{C}_3 = \mathcal{I}$ for simplicity reasons.

Case 1: $w_1 \geq w_2 \geq w_3$

This case is split into three sub-cases.

Case 1.1: $p_2 + p_3 \geq \sqrt{2}p_1$

We see that

$$p_1 = \frac{\sqrt{2}p_1}{\sqrt{2}} \leq \frac{p_2+p_3}{\sqrt{2}} \leq \frac{p_2+p_2}{\sqrt{2}} = \sqrt{2}p_2$$

and thus

$$\text{ratio}(\mathcal{C}(2,3,1)) = \max\left\{\frac{p_1}{p_2}, \frac{p_1+p_3}{p_2+p_3}, \frac{p_1+p_2}{p_2+p_3}\right\} = \max\left\{\frac{p_1}{p_2}, \frac{p_1+p_2}{p_2+p_3}\right\} \leq \sqrt{2}.$$

The last inequality holds true since $\frac{p_1}{p_2} \leq \frac{\sqrt{2}p_2}{p_2} = \sqrt{2}$ and $\frac{p_1+p_2}{p_2+p_3} \leq \frac{2p_1}{\sqrt{2}p_1} = \sqrt{2}$.

Case 1.2: $p_2 + p_3 \leq \sqrt{2}p_1$ and $p_3 \geq (\sqrt{2}-1)p_1$

We see that

$$\frac{p_2+p_3}{p_1} \leq \frac{\sqrt{2}p_1}{p_1} = \sqrt{2}$$

and

$$\frac{p_1+p_2}{p_1+p_3} \leq \frac{p_1+p_1}{p_1+(\sqrt{2}-1)p_1} = \frac{2p_1}{\sqrt{2}p_1} = \sqrt{2}$$

which results to

$$\text{ratio}(\mathcal{C}(1,3,2)) = \max\left\{\frac{p_2+p_3}{p_1}, \frac{p_1+p_2}{p_1+p_3}\right\} \leq \sqrt{2}.$$

Case 1.3: $p_2 + p_3 \leq \sqrt{2}p_1$ and $p_3 \leq (\sqrt{2}-1)p_1$

We see that

$$\frac{p_2+p_3}{p_1} \leq \frac{\sqrt{2}p_1}{p_1} = \sqrt{2}$$

and

$$\frac{p_1+p_3}{p_1} \leq \frac{p_1+(\sqrt{2}-1)p_1}{p_1} = \frac{\sqrt{2}p_1}{p_1} = \sqrt{2}$$

which results to

$$\text{ratio}(\mathcal{C}(1,2,3)) = \max\left\{\frac{p_2+p_3}{p_1}, \frac{p_1+p_3}{p_1}\right\} \leq \sqrt{2}.$$

Case 2: $w_1 \geq w_3 \geq w_2$

Case 2.1: $p_2 + p_3 \geq \sqrt{2}p_1$

Analogous to Case 1.1.

Case 2.2: $p_2 + p_3 \leq \sqrt{2}p_1$

$$\text{ratio}(\mathcal{C}(1,2,3)) = \max\left\{\frac{p_2+p_3}{p_1}, \frac{p_1+p_3}{p_1+p_2}\right\} \leq \max\left\{\frac{\sqrt{2}p_1}{p_1}, 1\right\} = \sqrt{2}$$

Case 3: $w_2 \geq \max\{w_1, w_3\}$

Case 3.1: $p_3 \geq (\sqrt{2}-1)p_1$

$$\text{ratio}(\mathcal{C}(1,3,2)) = \max\left\{\frac{p_2+p_3}{p_1+p_3}, \frac{p_1+p_2}{p_1+p_3}\right\} = \frac{p_1+p_2}{p_1+p_3} \leq \frac{2p_1}{p_1+(\sqrt{2}-1)p_1} = \sqrt{2}$$

Case 3.2: $p_3 \leq (\sqrt{2}-1)p_1$

$$\text{ratio}(\mathcal{C}(1,2,3)) = \max\left\{\frac{p_2+p_3}{p_1}, \frac{p_1+p_3}{p_1}\right\} = \frac{p_1+p_3}{p_1} \leq \frac{p_1+(\sqrt{2}-1)a_p}{p_1} = \sqrt{2}$$

Case 4: $w_3 \geq \max\{w_1, w_2\}$

$$\text{ratio}(\mathcal{C}(1,2,3)) = \max\left\{\frac{p_2+p_3}{p_1+p_2}, \frac{p_1+p_3}{p_1+p_2}\right\} \leq \max\left\{\frac{p_2+p_1}{p_1+p_2}, \frac{p_1+p_2}{p_1+p_2}\right\} = 1$$

□

2.4.2 Randomized Instance with Three Items

The lower bound for the competitive factor of randomized algorithms on instances with three items uses the same instance as in the deterministic setting which is $\mathcal{I} = \{1,2,3\}$ with $p_1 = w_1 = \sqrt{2}$ and $p_2 = w_2 = p_3 = w_3 = 1$.

Lemma 2.9. *There is no randomized algorithm which achieves a competitive factor of* 1.17 *on the above mentioned instance.*

Proof. Without loss of generality the outcome of the randomized algorithm is a maximal chain $\mathcal{C}$. For $\mathcal{C}_1$, we differentiate between two outcomes. These are profit and weight equals either 1 or $\sqrt{2}$. Let $q = \Pr[\mathcal{C}_1 = \{1\}]$ be the probability that $\mathcal{C}_1$ is the item set consisting of item 1. The complementary probability $1 - q$ states that the solution $\mathcal{C}_1$ consists of either item 2 or item 3.

The expected value of the randomized chain at capacity $\sqrt{2}$ is

$$\mathrm{p}(\mathbb{E}[C(\sqrt{2})]) \leq q \cdot \sqrt{2} + (1-q) \cdot 1 = 1 + q \cdot (\sqrt{2} - 1).$$

At capacity 2 we yield

$$\mathrm{p}(\mathbb{E}[C(2)]) \leq q \cdot \sqrt{2} + (1-q) \cdot 2 = 2 - q \cdot (2 - \sqrt{2})$$

with the same arguments as in the deterministic case.

We now analyze how good the chain is dependent on q.

Case 1: $q \leq 0.5$
The competitive factor at the capacity of $\sqrt{2}$ is

$$\frac{\mathrm{p}(OPT(\sqrt{2}))}{\mathrm{p}(\mathbb{E}[C(\sqrt{2})])} \geq \frac{\sqrt{2}}{1 + q \cdot (\sqrt{2} - 1)} \geq \frac{\sqrt{2}}{1 + 0.5 \cdot (\sqrt{2} - 1)} > 1.17.$$

Case 2: $q \geq 0.5$
The competitive factor at the capacity of 2 is

$$\frac{\mathrm{p}(OPT(2))}{\mathrm{p}(\mathbb{E}[C(2)])} \geq \frac{2}{2 - q \cdot (2 - \sqrt{2})} \geq \frac{2}{2 - 0.5 \cdot (2 - \sqrt{2})} > 1.17$$

which completes the proof. □

2.4.3 Unit Density Instance

For unit densities the previous results apply, since the instances only consist of items with unit density. The previous lower bound is $\sqrt{2}$ for the deterministic case and 1.17 for the randomized case. We show a lower bound of φ which means that no competitive factor of an algorithm is better than that. That lower bound matches the competitive factor of the already presented algorithm for unit densities.

A frequently used equality which is also needed in the upcoming proof is

$$\varphi^i + \varphi^{i-1} = \varphi^{i+1}.$$

Lemma 2.10. *For every $\delta > 0$, there exists an instance of the incremental unit density knapsack problem where no algorithm achieves a competitive factor of $\varphi - \delta$.*

The proof is based on an instance with five items. For any starting item, there exists a capacity at which the solution performs badly compared to the optimal solution.

Proof. Let ε be such that $0 < \varepsilon < \min\{0.1, \frac{\delta}{\varphi}\}$. We base our proof on the solution $\mathcal{C}_1$.

The instance consists of five items $w_1 = w_2 = 1 + \varepsilon$ and $w_3 = \frac{2}{\varphi}$, $w_4 = 1 + \frac{1}{\varphi^2}$ and $w_5 = \varphi$. Then, $\mathcal{C}$ is the chain which is the result of any arbitrary algorithm.

Case 1: $\mathcal{C}_1 = \{w_1\}$ or $\mathcal{C}_1 = \{w_2\}$
Let capacity $t = \varphi$. We see that $\mathcal{C}_2$ does not fit into the knapsack since

$$\mathrm{w}(\mathcal{C}_2) \geq w_1 + w_2 = 2 + 2\varepsilon > \varphi = t$$

which leads to a competitive factor of

$$\frac{\mathrm{w}(OPT(t))}{\mathrm{w}(\mathcal{C}(t))} = \frac{\mathrm{w}(OPT(t))}{\mathrm{w}(\mathcal{C}_1)} = \frac{\varphi}{1+\varepsilon} = \varphi - \frac{\varphi\varepsilon}{1+\varepsilon} > \varphi - \delta.$$

Case 2: $\mathcal{C}_1 = \{w_3\}$
Let capacity $t = 2 + 2\varepsilon$. We see that $\mathcal{C}_2$ does not fit into the knapsack since

$$\mathrm{w}(\mathcal{C}_2) \geq w_3 + w_1 = \frac{2}{\varphi} + 1 + \varepsilon > 2 + 2\varepsilon = t$$

which leads to a competitive factor of

$$\frac{\mathrm{w}(OPT(t))}{\mathrm{w}(\mathcal{C}(t))} = \frac{\mathrm{w}(OPT(t))}{\mathrm{w}(\mathcal{C}_1)} = \frac{w_1 + w_2}{w_3} = \frac{2+2\varepsilon}{\frac{2}{\varphi}} = \varphi \cdot (1+\varepsilon) > \varphi > \varphi - \delta.$$

Case 3: $\mathcal{C}_1 = \{w_4\}$
We choose capacity $t = 1 + \varepsilon + \frac{2}{\varphi}$. We see that $\mathcal{C}_2$ does not fit into the knapsack since

$$\mathrm{w}(\mathcal{C}_2) \geq w_4 + w_1 = 1 + \frac{1}{\varphi^2} + 1 + \varepsilon > 1 + \varepsilon + \frac{2}{\varphi} = t.$$

So at the chosen capacity we have a competitive factor greater than

$$\frac{\mathrm{w}(OPT(t))}{\mathrm{w}(\mathcal{C}_1)} = \frac{w_2 + w_3}{w_4} = \frac{1 + \varepsilon + \frac{2}{\varphi}}{1 + \frac{1}{\varphi^2}} > \frac{1 + \frac{2}{\varphi}}{1 + \frac{1}{\varphi^2}} = \frac{\frac{\varphi+2}{\varphi}}{\frac{\varphi^2+1}{\varphi^2}} = \varphi \cdot \frac{\varphi+2}{\varphi^2+1} = \varphi > \varphi - \delta.$$

We use that $\varphi + 2 = \varphi^1 + \varphi^0 + 1 = \varphi^2 + 1$.

Case 4: $\mathcal{C}_1 = \{w_5\}$
We choose capacity $t = \varphi + 1$. We see that $\mathcal{C}_2$ does not fit into the knapsack since

$$\mathrm{w}(\mathcal{C}_2) \geq w_5 + w_1 = \varphi + 1 + \varepsilon > \varphi + 1 = t.$$

So, at the chosen capacity we have a competitive factor greater than

$$\frac{\mathrm{w}(OPT(t))}{\mathrm{w}(\mathcal{C}_1)} = \frac{w_3 + w_4}{w_5} = \frac{\varphi + 1}{\varphi} = \frac{\varphi^1 + \varphi^0}{\varphi} = \frac{\varphi^2}{\varphi} = \varphi > \varphi - \delta.$$

In the above calculate we use

$$w_3 + w_4 = \frac{2}{\varphi} + 1 + \frac{1}{\varphi^2} = \left(\varphi^0 + \varphi^{-1}\right) + \left(\varphi^{-1} + \varphi^{-2}\right) = \varphi^1 + \varphi^0 = \varphi + 1$$

and that calculation completes the proof. □

2.4.4 Improved Deterministic Instance

In this section, the lower bound for the competitive factor of incremental knapsack instances follows ideas very similar to the ones we have already seen. This one uses more than three items. With an increasing number of items in the instance, the lower bound approaches a value of 2.

Utilizing three items, the idea is to have two small items and one big item. An algorithm has to choose if they start with a small item or the big item. If the algorithm chooses the big one, there is a capacity at which OPT packs two slightly smaller items and the big item is everything the chain offers. If the algorithm chooses a small item, the ratio is not too good compared to OPT using the biggest item accessible.

Lemma 2.11. *For every $\delta > 0$, there exists an instance of the incremental knapsack problem where no algorithm achieves a competitive factor of $2 - \delta$.*

Proof. For $\delta > 1$, we have $2 - \delta < 1$, which is not possible according to our definitions. Let $0 < \delta \leq 1$ be arbitrary and let $n = \lceil \frac{4}{\delta} \rceil$ be the number of items. We will use n items in this instance where item i has profit $p_i = \frac{n+i}{n}$ and weight $w_i = \mathcal{F}_n + \mathcal{F}_i - 1$ where $\mathcal{F}_i$ is the i-th Fibonacci number. The Fibonacci sequence starts with $\mathcal{F}_1 = \mathcal{F}_2 = 1$ and the following elements are defined by the recurrence relation $\mathcal{F}_i = \mathcal{F}_{i-1} + \mathcal{F}_{i-2}$. This instance is used to show a result on the unknown capacity problem in [5].

We formally proof the idea explained before. Let $\mathcal{C}$ be the maximal chain given by the algorithm after solving that instance. The solution $\mathcal{C}_1$ consists of an item k, formally

speaking $\mathcal{C}_1 = \{k\}$. We distinguish two cases. If $k \leq 2$, then item k is a small item and there are no two items which are smaller. If $k > 2$, then item k is a big item and there are at least two items with smaller weight and profit.

Case 1: $k \leq 2$

This case has a chain which starts with packing a single item which is either item 1 or item 2. Our capacity of choice is $t = w_n = 2\mathcal{F}_n - 1$. Here, OPT chooses only item n. The first observation is that there is no way two items could fit into the knapsack at capacity t and the chain has at most profit p_2. This is due to $\mathcal{C}_2$ not being feasible which can be seen by

$$\mathrm{w}(\mathcal{C}_2) \geq w_1 + w_2 = \mathcal{F}_n + \mathcal{F}_1 - 1 + \mathcal{F}_n + \mathcal{F}_2 - 1 = 2\mathcal{F}_n > 2\mathcal{F}_n - 1 = t.$$

Our competitive factor here is at least

$$\frac{\mathrm{p}(OPT)}{\mathrm{p}(\mathcal{C}_1)} \geq \frac{p_n}{p_2} = \frac{2n}{n+2} = \frac{2n+4}{n+2} - \frac{4}{n+2} > 2 - \frac{4}{\frac{4}{\delta}} = 2 - \delta.$$

Case 2: $k > 2$

In this case, we choose capacity $t = w_{k-1} + w_{k-2}$ and therefore the optimal solution contains two smaller items, namely $k-1$ and $k-2$. The weight of solution $\mathcal{C}_2$ will exceed the capacity since

$$\begin{aligned}
\mathrm{w}(\mathcal{C}_2) &\geq w_k + w_1 \\
&= \mathcal{F}_n + \mathcal{F}_k - 1 + \mathcal{F}_n + \mathcal{F}_1 - 1 \\
&> \mathcal{F}_n + \mathcal{F}_k - 1 + \mathcal{F}_n - 1 \\
&= \mathcal{F}_n + \mathcal{F}_{k-1} - 1 + \mathcal{F}_n + \mathcal{F}_{k-2} - 1 \\
&= w_{k-1} + w_{k-2} \\
&= t.
\end{aligned}$$

So, the solution at the chosen capacity has to be $\mathcal{C}_1$ and thus we get

$$\frac{\mathrm{p}(OPT)}{\mathrm{p}(\mathcal{C}_1)} = \frac{p_{k-1} + p_{k-2}}{p_k} = \frac{n+k-1+n+k-2}{n+k} = \frac{2n+2k}{n+k} - \frac{3}{n+k} > 2 - \frac{3}{\frac{4}{\delta}} > 2 - \delta$$

as the lower bound for the competitive factor which concludes the proof. □

Chapter 3

Incremental Multiple Knapsacks Problem

In this chapter, we will discuss the incremental multiple knapsacks problem (IKMP). In contrast to what we have already seen, we extend the setting from a single knapsack to multiple knapsacks. We denote the number of knapsacks by m. The capacity threshold is equal for all knapsacks. So far, only the order in which the items are packed was important and now the assignment to the knapsack is an additional task.

In Chapter 2, we already showed that even with m knapsacks the problem without minimum capacity restriction is not solvable. For $m = 1$, the problem reduces to the already seen incremental knapsack problem. Thus, the lower bound of the incremental knapsack problem transfers to the incremental multiple knapsacks problem. This gives us a rough outline on the problem.

There are two objective functions to measure the competitive factor for multiple knapsacks. One is to take the maximum of all knapsack profits and one is to take the sum of the knapsacks profits. For the maximum, the optimal solution would not need m knapsacks since one would be enough. The optimal solution is obtained by fixing a knapsack and using the optimal solution of the offline problem. An algorithm for m knapsacks could simply fix a single knapsacks and pack it according to the algorithm DetIKP. Independent of the number of knapsacks a competitive factor of 2 is achieved. Although there is a way to maintain the competitive factor, there is no guarantee that it is tight. The lower bound of 2 does not apply when there are multiple knapsacks. The objective function maximum is solvable with an algorithm for the single knapsack problem.

This is not viable for the sum objective function. The optimal solution uses all knapsacks to gain more profit. An algorithm for the problem also has to make use of every knapsack.

This objective function needs a different approach and is different in many ways. This makes the sum objective function interesting and our object of research in this chapter. Other objective functions are not included in this work. In the chapter, the incremental multiple knapsacks problem is always using the sum objective function.

3.1 Deterministic Algorithm

The algorithm DetIKP is already known and yields a competitive factor of 2 at the single knapsack problem. A first approach would be an extension to m knapsacks. For that to happen we have to maximize the number of items of DetIKP over multiple knapsacks. This is not possible due to the incremental constraint.

We now give a counter example that shows that in general there is no way to generalize algorithms from one to multiple knapsacks. Let us consider the set of weights $\{2, 3, 5, 6\}$ and two knapsacks. Then we want to maximize the weights using the items in ascending order. For the capacity 6, we can put weights 2 and 3 into one knapsack and weight 5 into the other. For capacity 8, we can put weights 2 and 6 into the first knapsack and weights 3 and 5 in the second knapsack. In an incremental setting, those solutions cannot be contained in the same chain. So, maximizing the number of items is not working for multiple knapsacks which was a critical property of the DetIKP. Similarly, problems also exist for UnitIKP when trying to generalize it for more than one knapsack.

We use a new approach for the incremental multiple knapsacks problem. There will be $\log n$ phases with capacities $t_i = 2^i$ for $0 \leq i \leq \lceil \log_2 (\sum_{i=1}^{n} w_i) \rceil$. The idea is that in phase i we find a 2-competitive approximation for capacity t_i. That solution should contain all items previously used. This is achieved by not simply running the approximation algorithm on the instance with an increased capacity but by modifying the instance and putting together the first result of the approximation with the second one.

So, at all capacities which are powers of 2, we have m different solutions such that the sum of profits is at most worse by a factor of 2 than the fractional optimal knapsack solution. The competitive factor is also measured at capacities, which are not necessarily powers of 2. For those the chain uses the solution with weight of the biggest power of 2 smaller than the capacity. Since the weight of solution is at least half of the capacity we obtain a competitive factor of 4.

In the pseudo-code, the variable I keeps track of all items which are not used in previous solutions and can therefore be used in later solutions. $\mathcal{C}_i$ is the i-th solution of the chain and $\mathcal{C}_i^l$ is the set of items used in the knapsack l by the i-th solution. Altogether the

Algorithm 6 DETIMKP$(\mathcal{I}, m)$

$\mathcal{C}_0 \leftarrow \emptyset$
$I \leftarrow \mathcal{I}$
for $i = 0$ **to** $\lceil \log_2 (\sum_{i=1}^{n} w_i) \rceil$ **do**
    for $l = 1$ **to** m **do**
        $t_i^l \leftarrow 2^i - w(\mathcal{C}_i^l)$
        $A_i^l \leftarrow$ APPROX(I, t_i^l)
        $I \leftarrow I / A_i^l$
        $\mathcal{C}_{i+1}^l \leftarrow \mathcal{C}_i^l \cup A_i^l$
    end for
end for
return $\mathcal{C}$

exponent $l \in [m]$ says that this variable refers to the l-th knapsack. t_i^l is the capacity which is available to extend the solution in knapsack l to the solution $i + 1$ of the chain.

We show two lemmas which will be useful in proving Theorem 3.3. Lemma 3.1 shows that for every two optimal fractional solutions with different weights the solution with lower weight has the higher density.

Lemma 3.1. *Let F_a be an optimal fractional solution with weight t_a and F_b an optimal fractional solution on the same items with weight t_b. If $\sum_{i=1}^{n} w_i \geq t_a \geq t_b > 0$ then $p(F_a) \leq t_a/t_b \cdot p(F_b)$.*

Proof. Let $r = \frac{t_a}{t_b}$. The first observation is

$$\mathrm{w}(F_a) = t_a = \frac{t_a}{t_b} \cdot t_b = r \cdot \mathrm{w}(F_b).$$

One can split up F_a into F_a/F_b and F_b such that

$$\mathrm{w}(F_a/F_b) = \mathrm{w}(F_a) - \mathrm{w}(F_b) = r \cdot \mathrm{w}(F_b) - \mathrm{w}(F_b) = (r-1) \cdot \mathrm{w}(F_b).$$

Further we can see that from the optimality of F_b it follows that

$$\forall i \in F_b, j \in \mathcal{I}/F_b : \frac{p_i}{w_i} \geq \frac{p_j}{w_j}$$

since the optimal fractional solution consists of the most dense items. Items which are partially used are considered to be contained in both F_b and $\mathcal{I}/F_b$.

We separate F_a into F_a/F_b and F_b. Then, we fix a least efficient item $k \in F_b$ yielding

$$\forall i \in F_b, j \in F_a/F_b : \frac{p_i}{w_i} \geq \frac{p_k}{w_k} \geq \frac{p_j}{w_j}.$$

Finally we obtain

$$\begin{aligned}
\mathrm{p}(F_a) &= \mathrm{p}(F_a/F_b) + \mathrm{p}(F_b) \\
&\leq \frac{p_k}{w_k} \cdot \mathrm{w}(F_a/F_b) + \mathrm{p}(F_b) \\
&= \frac{p_k}{w_k} \cdot (r-1) \cdot \mathrm{w}(F_b) + \mathrm{p}(F_b) \\
&\leq r \cdot \mathrm{p}(F_b) \\
&\leq \frac{t_a}{t_b} \cdot \mathrm{p}(F_b).
\end{aligned}$$

□

We show a second property which is used in the proof of Theorem 3.3.

Let the term $FOPT(m \cdot 2^k)$ refer to the fractional optimal solution on the item set $\mathcal{I}$ and consider a capacity of $m \cdot 2^k$ with a single knapsack. We know m knapsacks with capacity 2^k in the fractional multiple knapsacks problem have an equivalent solution. Since items can be split into small pieces, there is no difference in having one knapsack with capacity $m \cdot 2^k$ or m knapsacks with capacity 2^k. This is not the case in the integer problem setting. Let A_i^l refer to the solutions of ApproxKP computed in DetIMKP.

Lemma 3.2. $\mathrm{p}(FOPT(m \cdot 2^k)) \leq 2 \cdot \sum_{l=1}^{m} \sum_{i=0}^{k} \mathrm{p}(A_i^l)$

Proof. The solution $FOPT(m \cdot 2^k)$ is partitioned into a number of fractional solutions F_i^l where i and l are parameters. We can order the tuples (i, l) lexicographically. That means that $(i', l') < (i, l)$ if and only if $i' < i$ or both $i' = i$ and $l' < l$. Each F_i^l has a weight of t_i^l.

The partition will be done by using the most dense items and put them into the first F_i^l. Using the lexicographical sorting on the index of the solutions, the lower the index the better the density. Let $I_{i,l}$ be the set of items which is not used by the algorithm at the time $A_{i,l}$ is constructed. $FOPT(I_i^l, t_i^l)$ is the fractional solution on item set I_i^l with capacity t_i^l. We want to show that

$$\mathrm{p}(F_i^l) \leq \mathrm{p}(FOPT(I_i^l, t_i^l)).$$

The first thing to note is, that the capacity for both solutions is the same. Thus, the profit of the solutions only depend on the density of the items. Since the preceding solutions of

F_i^l greedily use the most dense items, F_i^l is left with the least good items. For $FOPT(I_i^l, t_i^l)$ the items which are left over are those used up by the approximation solutions $A_{i'}^{l'}$ where (i', l') is smaller than (i, l). $A_{i'}^{l'}$ has at most as much weight as $F_{i'}^{l'}$ and cannot consist of more items. If the two solutions differ in items, then $F_{i'}^{l'}$ used the item with higher density.

The algorithm ApproxKP from the Chapter 1 yields a 2-competitive solution. Since A_i^l works on the same items and has the same capacity as $FOPT(I_i^l, t_i^l)$, we can conclude that

$$\mathrm{p}(FOPT(I_i^l, t_i^l)) \leq \mathrm{p}(A_i^l),$$

which gives us

$$\mathrm{p}(FOPT(m \cdot 2^k)) = \sum_{l=1}^{m} \sum_{i=0}^{k} \mathrm{p}(F_i^l) \leq \sum_{l=1}^{m} \sum_{i=0}^{k} \mathrm{p}(FOPT(I_i^l, t_i^l)) \leq 2 \sum_{l=1}^{m} \sum_{i=0}^{k} \mathrm{p}(A_i^l).$$

□

We can now proof our main theorem.

Theorem 3.3. DetIMKP *is 4-competitive and has a run time of* $\mathcal{O}(n \log n)$.

Proof. Let t be arbitrary and k such that $2^k \leq t < 2^{k+1}$. Let $OPT(t)$ be the optimal solution of the incremental multiple knapsacks problem with capacity t and $FOPT$ be the fractional optimal solution in the single knapsack problem.

$$\begin{aligned}
\mathrm{p}(OPT(t)) &\leq \mathrm{p}(FOPT(m \cdot t)) && (3.1)\\
&\leq \mathrm{p}(FOPT(m \cdot 2^{k+1})) && (3.2)\\
&\leq 2 \cdot \mathrm{p}(FOPT(m \cdot 2^k)) && \text{(Lemma 3.1)}\\
&\leq 4 \cdot \sum_{l=1}^{m} \sum_{i=0}^{k} \mathrm{p}(A_{i,l}) && \text{(Lemma 3.2)}\\
&= 4 \cdot \sum_{l=1}^{m} \mathrm{p}(\mathcal{C}_{k+1}^l) && (3.3)\\
&= 4 \cdot \mathrm{p}(\mathcal{C}_{k+1}) && (3.4)
\end{aligned}$$

Inequality 3.1 uses that $OPT(t)$ is the optimal solution to incremental multiple knapsacks problem and its profit is at most as good as the one for the fractional optimal solution

of the same problem. For fractional solutions, it does not matter if they have a single knapsack with capacity $m \cdot t$ or m knapsacks with capacity t.

Inequality 3.2 holds since raising the capacity will not decrease the profit of an optimal solution. The next two inequalities are proven in Lemma 3.1 and 3.2.

To complete the theorem, equality 3.3 uses definitions of the algorithm. If we apply the definition of $\mathcal{C}_{k+1}^l$ k times, we are left with a union of A_i^l and $\mathcal{C}_0^l$ which is the empty set.

$$\mathcal{C}_{k+1}^l = \mathcal{C}_k^l \cup A_k^l = \mathcal{C}_{k-1}^l \cup A_{k-1}^l \cup A_k^l = ... = \mathcal{C}_0^l \cup A_1^l \cup ... \cup A_k^l = \cup_{i=1}^k A_i^l$$

The profit of $\mathcal{C}_{k+1}^l$ is therefore

$$\mathrm{p}(\mathcal{C}_{k+1}^l) = \mathrm{p}(\cup_{i=1}^k A_i^l) = \sum_{i=0}^{k} \mathrm{p}(A_{i,l})$$

and the union of all knapsacks $l \in [m]$ gives us the complete solution.

It is left to show that $\mathcal{C}_{k+1}$ is a feasible solution. This is the case since for each knapsack l it holds

$$\mathrm{w}(\mathcal{C}_{k+1}^l) = \mathrm{w}(\mathcal{C}_k^l) + \mathrm{w}(A_k^l) \leq \mathrm{w}(\mathcal{C}_k^l) + t_k^l = \mathrm{w}(\mathcal{C}_k^l) + 2^k - \mathrm{w}(\mathcal{C}_k^l) = 2^k \leq t.$$

Our algorithm can be implemented with a run time of $\mathcal{O}(n \log n)$ using algorithm ApproxKP.

The first part is to sort all items by descending density in time $\mathcal{O}(n \log n)$. We then initialize an array of length n which can store a tuple of two values which tells the chain where the item belongs, more exactly one value determines the knapsack (l) and one determines the number of solutions in the chain (i). This marks the solution in the chain where the item appears first. This we call chain information of an item.

The approximation algorithm calls examine all items of the item set and put them into the chain according to DetIMKP. We have two variables to mark start and stop of the greedy solution and variables to remember the weight and profit of the greedy solution. After the computation of greedy solution and swap item solution is done and the better is determined, we adjust these variables to avoid unnecessary computations in the next call. When the capacity is increased, the greedy solution will contain all the previous items and additional new items. So the greedy solution is just extended and not computed from scratch.

If the greedy solution is returned, all items between start and stop of the greedy solution are saved with the current chain information. Then we reset all variables. If any of those items was already marked with chain information, it is a swap item from an earlier call of the algorithm and the information is left unchanged. If the swap item solution is returned, the stop marker moves by 1 and the item is get its chain information. The start of the greedy solution, the weight and the profit are not reseted. In the next call of the approximation algorithm the profit and weight is just updated and the stop of the greedy solution as well.

So the total run time of all approximation algorithm calls is linear. The nested for loops are limited to $n \log n$ iterations since $m \leq n$. Each iteration is constant time by the above argumentation about the run time of the approximation algorithm.

Altogether, we obtain a run time of $\mathcal{O}(n \log n)$. □

We now discuss some limitations of the algorithm. More specifically we argue that the competitive factor of 4 cannot be improved when using the general idea of approximating many different instances. The competitive factor has two issues. One is the approximation algorithm ApproxKP and the other is the factor of 2 of the phase capacities t_i.

The approximation algorithm has to have the special property of being good not only compared to the optimal solution but also compared to the fractional solution. In Chapter 1 there was an example which shows that the profit factor between an integral and a fractional solution can be arbitrary close to 2. So the competitive ratio of ApproxKP is tight and cannot be improved. The choice of the phase capacities is made such that for any $i > 1$ the extra capacity is at least

$$t^{i+1} - t^i = 2^{i+1} - 2^i = 2^i \cdot (2 - 1) = 2^i \geq 2^0 = 1$$

and $t_0 = 1$.

The property is important since it gives an approximation algorithm the option to pack any available item. If $t^{i+1} - t^i < 1$ the algorithm could not use an item with weight 1 and very high profit and is restricted to use other items. This can lead to arbitrary bad approximation factors.

The approach of $\log n$ phases of item packing has its limits and the competitive factor cannot be improved.

3.2 Unit Densities

The incremental multiple knapsacks problem can be limited to unit density instances. Analogous to the single knapsack variant, we only use the weights of the item set and ignore the profits. For a single knapsack, we achieve a competitive factor of φ. The algorithm UnitIKP from Section 2.3 does not carry over to multiple knapsacks, since we argue that a greedy approach, which uses the least weight item first, maximizes the number of items. This does not hold for multiple knapsacks settings, as we have already seen from a counter example for the general setting with multiple knapsacks at the beginning of the chapter.

The algorithm UnitIMKP for the incremental multiple knapsacks problem is a greedy algorithm. The algorithm keeps track of the current weight of every knapsack. Then, the knapsack with the least weight is always chosen. The items are inserted from highest weight to lowest weight. The goal is to show that every knapsack is at least half filled for all capacity thresholds.

Algorithm 7 UnitIMKP$(\mathcal{I}, m)$

```
1: Sort items by ascending weight
2: C_0 ← ∅
3: for i from 1 to n do
4:     s ← knapsack with smallest weight
5:     C_i ← C_{i−1}
6:     C_i^s ← C_i^s ∪ {n − i + 1}
7: end for
8: return C
```

Theorem 3.4. UnitIMKP *is 2-competitive and has* $\mathcal{O}(n \log n)$ *run time.*

Proof. Let $t < \sum_{i=1}^{n} w_i$ be an arbitrary capacity.

Then, the chain $\mathcal{C}$ yields the biggest solution of the maximal chain $\mathcal{C}_i$ such that all knapsacks have weight at most t. We know for maximal chains that the number of items in $\mathcal{C}_i$ is i. Since every knapsack can pack at least one item, all knapsacks are non-empty. In line 4 the knapsack with the least weight is chosen and in the first m iterations one of the empty knapsacks is returned.

The item $i+1$ did not fit the knapsack with the least weight and thus, it would not have to fit into any knapsack, but the solution $\mathcal{C}_i$ is feasible, i.e. for all l

$$\mathrm{w}(\mathcal{C}_i^l) \leq t \quad \text{and} \quad \mathrm{w}(\mathcal{C}_i^l) + w_{i+1} \geq \mathrm{w}(\mathcal{C}_i^s) + w_{i+1} > t.$$

Since w_{i+1} is smaller than the biggest m items and every knapsack has packed one of those, all knapsack have weight at least w_{i+1}.

$$\mathrm{w}(\mathcal{C}_i^l) = \frac{1}{2}\left(\mathrm{w}(\mathcal{C}_i^l) + \mathrm{w}(\mathcal{C}_i^l)\right) \geq \frac{1}{2}\left(\mathrm{w}(\mathcal{C}_i^l) + w_{i+1}\right) \geq \frac{1}{2} \cdot t$$

The optimal solution has no more weight than $m \cdot t$ and $\mathcal{C}_i$ has at least weight $\frac{1}{2} \cdot m \cdot t$.

$$\frac{\mathrm{w}(OPT(t))}{w(\mathcal{C}(t))} \leq \frac{m \cdot t}{\sum_{l=1}^{m} \mathrm{w}(\mathcal{C}^l(t))} \leq \frac{m \cdot t}{\sum_{l=1}^{m} \frac{1}{2} \cdot t} = \frac{m \cdot t}{m \cdot \frac{1}{2} \cdot t} = 2$$

Therefore the competitive factor is 2.

The run-time of the algorithm is $\mathcal{O}(n \log n)$. The sorting of items takes $\mathcal{O}(n \log n)$ time. The loop is called a total of n times. Each iteration has to determine the knapsack with the least weight. The easiest way to do this fast is managing a min queue with all knapsack weights. Once a knapsack has been removed, it is inserted with a new weight. Removing the minimum is a constant time operation and inserting a new weight is a logarithmic time operation. This yields

$$\mathcal{O}(1 + \log m) = \mathcal{O}(\log m) = \mathcal{O}(\log n)$$

run time for one iteration. In total, we obtain $\mathcal{O}(n \log n)$ and therefore the algorithm's run time is also $\mathcal{O}(n \log n)$. □

3.3 Lower Bounds

In this section, we discuss lower bounds for the incremental multiple knapsacks problem. For general instances with $m = 1$, we showed a lower bound of $2 - \delta$ for all $\delta > 0$ in Section 2.4.4. For unit density instances with $m = 1$, we showed a lower bound of $\varphi - \delta$ for all $\delta > 0$ in Section 2.4.3. Since we are interested in constant lower bounds, these are also results in the variants with multiple knapsacks.

For individual lower bounds on a fixed number $m > 1$ of knapsacks, the lower bounds are not as good as the ones for a single knapsack. We show a lower bound of $1 + \frac{1}{2m} - \delta$ for all $\delta > 0$ for general instances. For unit density instances, it remains open what lower bounds dependent on m there are.

For the general case, we use the same instance as before (2.4.4) in the single knapsack problem and extend the analysis to an arbitrary number of knapsacks. For that, we will

determine the m items which are packed first and look at the smallest item. Then, we will choose the capacity such that one knapsack in the optimal solution is almost twice as profitable and every other knapsack at least as good as the chain. This will give us a lower bound of more than 1 for every possible number of knapsacks.

Lemma 3.5. *For every $\delta > 0$, there exists an instance of the incremental multiple knapsacks problem where no algorithm achieves a competitive factor of $1 + \frac{1}{2m} - \delta$.*

Proof. Let $n = \max\{\lceil \frac{1}{\delta} \rceil, m + 1\}$. Then, the instance has n items where an item i has profit $p_i = \frac{n+i}{n}$ and weight $w_i = \mathcal{F}_n + \mathcal{F}_i - 1$ where $\mathcal{F}_i$ is the i-th Fibonacci number.

Let us denote a maximal incremental chain by $\mathcal{C}$ and the set of items which are packed first in any knapsack by M. Then, $k = \min\{i \mid i \in M\}$ is the item with the smallest index and, by construction, also the one with the smallest profit and weight.

We use $\varepsilon = \frac{(m-2)(m-1)}{2n}$ to simplify the notation in the following calculations. Then, the greatest $m - 1$ items have a total profit of

$$\begin{aligned}
\sum_{i=n-m+2}^{n} p_i &= \sum_{j=1}^{m-1} p_{n-j+1} \\
&= \sum_{j=1}^{m-1} \frac{2n - j + 1}{n} \\
&= 2\,(m-1) - \sum_{j=1}^{m-1} \frac{j-1}{n} \\
&= 2\,(m-1) - \sum_{j=1}^{m-2} \frac{j}{n} \\
&= 2\,(m-1) - \frac{(m-2)(m-1)}{2n} \\
&= 2\,(m-1) - \varepsilon.
\end{aligned}$$

Case 1: $k \leq 2$

Let the capacity $t = w_n$. Thus, each knapsack can hold exactly one item. Then, the optimal solution contains the m most profitable items and has a total profit of

$$\mathrm{p}(OPT(t)) = \sum_{i=n-m+1}^{n} p_i = p_{n-m+1} + 2\,(m-1) - \varepsilon = 2 - \frac{m-1}{n} + 2\,(m-1) - \varepsilon$$

and the chain has a total profit of at most

$$\mathrm{p}(\mathcal{C}(t)) \leq p_k + \sum_{i=n-m+2}^{n} p_i = \frac{n+k}{n} + 2\,(m-1) - \varepsilon.$$

The competitive ratio is

$$\begin{aligned}
\frac{\mathrm{p}(OPT(t))}{\mathrm{p}(\mathcal{C}(t))} &\geq \frac{2 - \frac{m-1}{n} + 2\,(m-1) - \varepsilon}{\frac{n+k}{n} + 2\,(m-1) - \varepsilon} \\
&= 1 + \frac{2 - \frac{m-1}{n} - \frac{n+k}{n}}{\frac{n+k}{n} + 2\,(m-1) - \varepsilon} \\
&> 1 + \frac{1}{2m} - \frac{\frac{m+1}{n}}{2m} && (m \in \mathbb{N}_{>0}) \\
&\geq 1 + \frac{1}{2m} - \frac{1}{n} \\
&\geq 1 + \frac{1}{2m} - \delta.
\end{aligned}$$

Case 2: $k > 2$

Let the capacity $t = w_{k-1} + w_{k-2}$, then the chain is forced to pack at most one item into each knapsack. Since the weight of the items packed first into a knapsack of the chain is at least w_k, a knapsack with two items has a weight of at least

$$w_k + w_1 > w_{k-1} + w_{k-2} = t$$

as discussed in the previous proof.

The optimal solution could fit the items $k-1$ and $k-2$ in a single knapsack and fill the other $m-1$ knapsacks with the $m-1$ most profitable items. This is a lower bound for the profit of the optimal solution. The profit of $\mathcal{C}(t)$ is greatest if item k is in one knapsack and the $m-1$ most profitable items are in the other knapsacks. This yields

$$\mathrm{p}(OPT(t)) \geq p_{k-1} + p_{k-2} + \sum_{j=1}^{m-1} p_{n-j+1} = \frac{2n + 2k - 3}{n} + 2\,(m-1) - \varepsilon$$

and

$$\mathrm{p}(\mathcal{C}(t)) \leq p_k + \sum_{j=1}^{m-1} p_{n-j+1} = \frac{n+k}{n} + 2\,(m-1) - \varepsilon.$$

The competitive ratio is

$$\begin{aligned}
\frac{\mathrm{p}(OPT(t))}{\mathrm{p}(\mathcal{C}(t))} &\geq \frac{\frac{2n+2k-3}{n} + 2\,(m-1) - \varepsilon}{\frac{n+k}{n} + 2\,(m-1) - \varepsilon} \\
&= 1 + \frac{\frac{2n+2k-3}{n} - \frac{n+k}{n}}{\frac{n+k}{n} + 2\,(m-1) - \varepsilon} \\
&> 1 + \frac{\frac{2n+2k-3}{n} - \frac{n+k}{n}}{2m} \\
&= 1 + \frac{\frac{n+k-3}{n}}{2m} \\
&\geq 1 + \frac{1}{2m} \\
&> 1 + \frac{1}{2m} - \delta.
\end{aligned}$$

□

Chapter 4

Conclusion

We researched variants of the incremental knapsack problem and presented and analyzed several algorithms for these problems.

The first variant is the incremental knapsack problem without any constraints. For that, we showed that no deterministic algorithm can achieve any competitive factor. This also holds for the variant with multiple knapsacks. For randomized algorithms this result is similar though not exactly the same. For single knapsack and multiple knapsacks problems, a randomized algorithm does not achieve a constant competitive factor. However, for the incremental knapsack problem we presented an algorithm with competitive factor $\mathcal{O}(2^n)$ using randomization. This could be further examined in future work, since the lower bound of $\Omega(1)$ leaves a significant gap.

Furthermore, we introduced a constraint to incremental problems. We allowed the chain to always pack a single item. This was done by modifying the competitive factor definition and only use capacities such that every solution with a single item is feasible. For that constraint we looked at the same variants. We achieved a competitive factor of 2 for the incremental knapsack problem and a competitive factor of 4 in the setting with multiple knapsacks. For a single knapsack, the result is the best possible factor. For a fixed number $m \geq 2$ of knapsacks we showed the competitive factor cannot become much better than $\frac{2m+1}{2m}$ which is at best 1.2. In general, the problem is as hard as the single knapsack variant which results in a lower bound of 2 for the case $m = 1$. This leaves us with a gap between the lower bound on the competitive factor and the actual competitive factor of our algorithm. The general idea of the algorithm for multiple knapsacks has reached its limit, so further improvements will need a different approach for a new algorithm. On the other hand, the lower bound for the competitive factor is based on a construction for bad instances of one knapsack. Big improvements on the lower bound should be possible.

We also investigated instances of items with unit density. For the single knapsack we presented an algorithm with competitive factor of $\frac{1+\sqrt{5}}{2} \approx 1.62$. This number is also known as the golden ratio and is found very often in the context of Fibonacci numbers. Here, the lower bound of the competitive factor is the same as the competitive factor of the algorithm. The instance which shows the lower bound uses Fibonacci numbers as item weights. For the unit density variant of the incremental multiple knapsacks problem we presented a greedy algorithm with a competitive factor of 2. For one knapsack, the lower bound of ≈ 1.62 is obtained. Considering a given number of knapsacks, it remains an open question whether the competitive factor can be improved.

This leaves us with open questions for future research. The incremental multiple knapsacks problem has gaps between the presented algorithms and the lower bound for the competitive factor. The incremental knapsack problem can be regarded as solved in the deterministic case.

Another set of variants which include randomization has not been researched yet. For the incremental knapsack problem the algorithm using randomization is simpler than the deterministic one but achieves the same competitive factor. The lower bound for the competitive factor of randomized algorithms is 1.17. In the unit density variant and all multiple knapsacks variants no randomization is considered.

For now, it remains an open question whether randomization can improve the competitive factor of an algorithm for any incremental knapsack variant or not.

Bibliography

[1] Steven S. Skiena. "Who is Interested in Algorithms and Why?: Lessons from the Stony Brook Algorithms Repository". In: *SIGACT News* 30.3 (Sept. 1999), pp. 65–74. ISSN: 0163-5700. DOI: 10.1145/333623.333627. URL: http://doi.acm.org/10.1145/333623.333627.

[2] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004. ISBN: 978-3-540-40286-2.

[3] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. 5th. Springer Publishing Company, Incorporated, 2007. ISBN: 978-3-642-24488-9.

[4] A. Marchetti-Spaccamela and C. Vercellis. "Stochastic on-line knapsack problems". In: *Mathematical Programming* 68.1 (Jan. 1995), pp. 73–104. ISSN: 1436-4646. DOI: 10.1007/BF01585758. URL: https://doi.org/10.1007/BF01585758.

[5] Yann Disser et al. "Packing a Knapsack of Unknown Capacity". In: *CoRR* abs/1307.2806 (2013). URL: http://arxiv.org/abs/1307.2806.

[6] Guolong Lin et al. "A General Approach for Incremental Approximation and Hierarchical Clustering". In: *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*. SODA '06. Miami, Florida: Society for Industrial and Applied Mathematics, 2006, pp. 1147–1156. ISBN: 0-89871-605-5. URL: http://dl.acm.org/citation.cfm?id=1109557.1109684.

[7] Heiko Röglin. *Algorithmen und Berechnungskomplexitaet I*. 2017. URL: http://www.roeglin.org/teaching/Skripte/AlgoI.pdf (visited on 05/22/2017).