Packing a Knapsack of Unknown Capacity

Yann Disser[*], **Max Klimm**, **Nicole Megow**[†], **and Sebastian Stiller**

Department of Mathematics, Technische Universität Berlin, Germany
`{disser,klimm,nmegow,stiller}@math.tu-berlin.de`

Abstract

We study the problem of packing a knapsack without knowing its capacity. Whenever we attempt to pack an item that does not fit, the item is discarded; if the item fits, we have to include it in the packing. We show that there is always a policy that packs a value within factor 2 of the optimum packing, irrespective of the actual capacity. If all items have unit density, we achieve a factor equal to the golden ratio $\varphi \approx 1.618$. Both factors are shown to be best possible.

In fact, we obtain the above factors using packing policies that are *universal* in the sense that they fix a particular order of the items and try to pack the items in this order, independent of the observations made while packing. We give efficient algorithms computing these policies. On the other hand, we show that, for any $\alpha > 1$, the problem of deciding whether a given universal policy achieves a factor of α is coNP-complete. If α is part of the input, the same problem is shown to be coNP-complete for items with unit densities. Finally, we show that it is coNP-hard to decide, for given α, whether a set of items admits a universal policy with factor α, even if all items have unit densities.

1998 ACM Subject Classification F.2.2 Sequencing and Scheduling

Keywords and phrases Knapsack, unknown capacity, robustness, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.276

1 Introduction

In the standard knapsack problem we are given a set of items, each associated with a size and a value, and a capacity of the knapsack. The goal is to find a subset of the items with maximum value whose size does not exceed the capacity. In this paper, we study the *oblivious* knapsack problem where the capacity of the knapsack is not given. Whenever we try to pack an item, we observe whether or not it fits the knapsack. If it does, the item is packed into the knapsack and cannot be removed later. If it does not fit, we discard it and continue packing with the remaining items. The central question of this paper is how much we loose by not knowing the capacity, in the worst case. The oblivious variant of the knapsack problem naturally arises whenever items are prioritized by a different entity or at a different time than the actual packing of the knapsack. For example, waiting lists for stand-by capacities and services, e.g., on an airplane, have to be fixed before the exact amount of free seats due to no-shows is known. The order of the list must be determined based only on the size of the items (e.g., the sizes of groups of travelers) and their value (e.g., compensation for service denial).

A solution to the oblivious knapsack problem is a policy that governs the order in which we attempt to pack the items, depending only on the observation which of the previously attempted items did fit into the knapsack and which did not. In other words, a policy is a

[*] Supported by the Alexander von Humboldt Foundation.
[†] Supported by the German Science Foundation (DFG) under contract ME 3825/1.

binary decision tree with the item that is tried first at its root. The two children of the root are the items that are tried next, which of the two depends on whether or not the first item fits the knapsack, and so on. We aim for a solution that is good for *every* possible capacity, compared to the best solution of the standard knapsack problem for this capacity. Formally, a policy has *robustness factor* α if, for any capacity, packing according to the policy results in a value that is at least a $1/\alpha$-fraction of the optimum value for this capacity.

We show that the oblivious knapsack problem always admits a robustness factor of 2. In fact, this robustness factor can be achieved with a policy that packs the items according to a fixed order, irrespective of the observations made while packing. Such a policy is called *universal.* We provide an algorithm that computes a 2-robust, universal policy in time $\Theta(n \log n)$ for a given set of n items. We complement this result by showing that no robustness factor better than 2 can be achieved in general, even by policies that are not universal. In other words, the cost of not knowing the capacity is exactly 2.

We give a different efficient algorithm for the case that all items have unit density, i.e., size and value of each item coincide. This algorithm produces a universal policy with a robustness factor of at most the golden ratio $\varphi \approx 1.618$. Again, we show that no better robustness factor can be achieved in general, even by policies that are not universal.

While good universal policies can be found efficiently, it is intractable to compute the robustness factor of a *given* universal policy and it is intractable to compute the best robustness factor an instance admits. Specifically, we show that, for any fixed $\alpha \in (1, \infty)$, it is coNP-complete to decide whether a given universal policy is α-robust. For unit densities we establish a slightly weaker hardness result by showing that it is coNP-complete to decide whether a given universal policy achieves a *given* robustness factor α. Finally, we show that, for given α, it is coNP-hard to decide whether an instance of the oblivious knapsack problem admits a universal policy with robustness factor α, even when all items have unit density.

Related work

The knapsack problem has been studied for different models of imperfect information. In the *stochastic* knapsack problem, sizes and values of the items are random variables. It is known that a policy maximizing the expected value is PSPACE-hard to compute, see Dean et al. [7]. The authors assume that the packing stops when the first item does not fit the knapsack, and give a universal policy that approximates the value obtained by an optimal, not necessarily universal, policy by a factor of 2. They also provide a non-universal policy within a factor of $3 + \epsilon$ of the optimal policy. Bhalgat et al. [3] give an algorithm with an improved approximation guarantee of $8/3 + \epsilon$. They also give a PTAS for the case that it is allowed to violate the capacity of the knapsack by a factor of $1 + \epsilon$.

In *robust* knapsack problems, a set of possible scenarios for the sizes and values of the items is given. Yu [23], Bertsimas and Sim [2], Goetzmann et al. [11], and Monaci and Pferschy [18] study the problem of maximizing the worst-case value of a knapsack under various models. Büsing et al. [5] and Bouman et al. [4] study the problem from a computational point of view. Both allow for an adjustment of the solution after the realization of the scenario. Similar to our model, Bouman et al. consider uncertainty in the capacity.

The notion of a *robustness factor* that we adopt in this work is due to Hassin and Rubinstein [12] and is defined as the worst-case ratio of solution and optimum, over all realizations. Kakimura et al. [14] analyze the complexity of deciding whether an α-robust solution exists for a knapsack instance with an unknown bound on the number of items that can be packed. Megow and Mestre [16] study a variant of the knapsack problem with unknown capacity closely related to ours. In contrast to our model, they assume that the packing stops once the first item does not fit the remaining capacity. In this model, a universal policy with a constant robustness factor may fail to exist, and, thus, Megow and Mestre resort to *instance-sensitive* performance guarantees. They provide a PTAS that constructs a universal policy with robustness factor arbitrarily close to the best possible robustness factor for every particular instance.

The concept of *obliviousness* is used in various other contexts (explicitly or implicitly), such as hashing (Carter and Wegman [6]), caching (Frigo et al. [10], Bender et al. [1]) routing (Valiant and Brebner [22], Räcke [20]), TSP (Papadimitriou [19], Deineko et al. [8], Jia et al, [13]), Steiner tree and set cover (Jia et al, [13]), and scheduling (Epstein et al. [9], Megow and Mestre [16]). In all of these works, the general idea is that specific parameters of a problem instance are unknown, e.g., the cache size or the set of vertices to visit in a TSP tour, and the goal is to find a *universal solution* that performs well for all realizations of the hidden parameters.

Universal policies for the oblivious knapsack problem play a role in the design of public key cryptosystems. One of the first such systems – the Merkle–Hellman knapsack cryptosystem [17] – is based on particular instances that allow for a 1-robust universal policy for the oblivious knapsack problem. The basic version of this cryptosystem can be attacked efficiently, e.g., by the famous attack of Shamir [21]. This attack uses the fact that the underlying knapsack instance has exponentially increasing item sizes. A better understanding of universal policies may help to develop knapsack-based cryptosystems that avoid the weaknesses of Merkle and Hellman's.

2 Preliminaries

An instance of the *oblivious knapsack problem* is given by a set of n items $\mathcal{I}$, where each item $i \in \mathcal{I}$ has a non-negative *value* $v(i) \in \mathbb{Q}_{\geq 0}$ and a strictly positive *size* $l(i) \in \mathbb{Q}_{>0}$. For a subset $S \subseteq \mathcal{I}$ of items, we write $v(S) = \sum_{i\in S} v(i)$ and $l(S) = \sum_{i\in S} l(i)$ to denote its total value and total size, respectively, of the items in S. A *solution* for instance $\mathcal{I}$ is a policy $\mathcal{P}$ that governs the order in which the items are considered for packing into the knapsack. The policy must be independent of the capacity of the knapsack, but the choice which item to try next may depend on the observations which items did and which items did not fit the knapsack so far. Formally, a solution policy is a binary decision tree that contains every item exactly once along each path from the root to a leaf. The *packing* $\mathcal{P}(C) \subseteq \mathcal{I}$ of $\mathcal{P}$ for a fixed capacity C is obtained as follows: We start with $\mathcal{P}(C) = \emptyset$ and check whether the item r at the root of $\mathcal{P}$ fits the knapsack, i.e., whether $l(r) + l(\mathcal{P}(C)) \leq C$. If the item fits, we add r to $\mathcal{P}(C)$ and continue packing recursively with the left subtree of r. Otherwise, we discard r and continue packing recursively with the right subtree of r.

A *universal policy* Π for instance $\mathcal{I}$ is a policy that does not depend on observations made while packing, i.e., the decision tree for a universal policy has a fixed permutation of the items along every path from the root to a leaf. We identify a universal policy with this fixed permutation and write $\Pi = (\Pi_1, \Pi_2, \dots, \Pi_n)$. Analogously to general policies, the packing $\Pi(C) \subseteq \mathcal{I}$ of a universal policy Π for capacity $C \leq l(\mathcal{I})$ is obtained by considering the items in the order given by the permutation Π and adding every item if it does not exceed the remaining capacity. We measure the quality of a policy for the oblivious knapsack problem by comparing its packing with the optimal packing for each capacity. More precisely, a policy $\mathcal{P}$ for instance $\mathcal{I}$ is called *α-robust for capacity* C, $\alpha \geq 1$, if it holds that $v(\textsc{Opt}(\mathcal{I}, C)) \leq \alpha \cdot v(\mathcal{P}(C))$, where $\textsc{Opt}(\mathcal{I}, C)$ denotes an optimal packing for capacity C. We say $\mathcal{P}$ is *α-robust* if it is α-robust for all capacities. In this case, we call α the *robustness factor* of policy $\mathcal{P}$.

3 Solving the Oblivious Knapsack Problem

In this section, we describe an efficient algorithm that constructs a universal policy for a given instance of the oblivious knapsack problem. The solution produced by our algorithm is guaranteed to pack at least half the value of the optimal solution for any capacity C. We show that this is the best possible robustness factor.

The analysis of our algorithm relies on the classical *modified greedy* algorithm (cf. [15]). We compare the packing of our policy, for each capacity, to the packing obtained by the modified greedy algorithm instead of the actual optimum. As the modified greedy is a 2-approximation, to show that our policy is 2-robust it is sufficient to show that its packing is never worse than the one obtained by the modified greedy algorithm. We briefly review the modified greedy algorithm.

Let $d(i) = v(i)/l(i)$ denote the *density* of item i. The modified greedy algorithm (MGreedy) for a set of items $\mathcal{I}$ and known knapsack capacity C first discards all items that are larger than C from $\mathcal{I}$. The remaining items are sorted in non-increasing order of their densities, breaking ties arbitrarily. The algorithm then either takes the longest prefix P of the resulting sequence that still fits into capacity C, or the first item s that does not fit anymore, depending on which of the two has a greater value. In the latter case, we say that s is a *swap item* (for capacity C) and that C is a *swap capacity.* In both cases, we refer to P as the *greedy set* for capacity C. See Algorithm 1 for a formal description.

For our analysis, it is helpful to fix the tie-breaking rule under which MGreedy initially sorts the items. To this end, we assume that there is a bijection $t : \mathcal{I} \to \{1, 2, \ldots, n\}$, that maps every item $i \in \mathcal{I}$ to a *tie-breaking index* $t(i)$, and that the modified greedy algorithm initially sorts the items decreasingly with respect to the tuple $\tilde{d}(\cdot) = (d(\cdot), t(\cdot))$, i.e., the items are sorted non-increasingly by density and whenever two items have the same density, they are sorted by decreasing tie-breaking index. In the following, for two items i, j, we write $\tilde{d}(i) \succ \tilde{d}(j)$ if and only if $d(i) > d(j)$, or $d(i) = d(j)$ and $t(i) > t(j)$, and say that i has higher density than j.

We evaluate the quality of our universal policy by comparing it for every capacity with the solution of MGreedy. This analysis suffices because of the following well-known property of the modified greedy algorithm.

▶ **Theorem 1** (cf. [15]). *For every instance $(\mathcal{I}, C)$ of the standard knapsack problem with known capacity, $v(\textsc{Opt}(\mathcal{I}, C)) \le 2 \cdot v(\textsc{MGreedy}(\mathcal{I}, C))$.*

We are now ready to describe our algorithm Universal (Algorithm 2) that produces a universal policy tailored to imitate the behavior of MGreedy without knowing the capacity.

Algorithm 1: MGreedy($\mathcal{I}$, C)

```
Input: set of items $\mathcal{I}$, capacity $C$
Output: subset $S \subseteq \mathcal{I}$ such that $l(S) \le C$ and $v(S) \ge v(\textsc{Opt}(\mathcal{I}, C))/2$
$D \leftarrow \langle$items in $\{i \in \mathcal{I} \mid l(i) \le C\}$ sorted decreasingly by density $\tilde{d}\rangle$
$k \leftarrow \max\{j \mid l(\{D_1, \ldots, D_j\}) \le C\}$
$P \leftarrow (D_1, \ldots, D_k)$, $s \leftarrow D_{k+1}$
if $v(P) \ge v(s)$ then
    return $P$
else
    return $\{s\}$
```

```
Algorithm 2: UNIVERSAL($\mathcal{I}$)
  Input: set of items $\mathcal{I}$
  Output: sequence of items $\Pi$
  $L \leftarrow \langle$items in $\mathcal{I}$ sorted by non-decreasing size$\rangle$
  $\Pi^{(0)} \leftarrow \emptyset$
  for $r \leftarrow 1, \dots, n$ do
      if $L_r$ is a swap item then
          $\Pi^{(r)} \leftarrow (L_r, \Pi^{(r-1)})$
      else
          $j \leftarrow 1$
          while $j \leq |\Pi^{(r-1)}|$ and $\tilde{d}(\Pi_j^{(r-1)}) \succ \tilde{d}(L_r)$ do
              $j \leftarrow j + 1$
          $\Pi^{(r)} \leftarrow (\Pi_1^{(r-1)}, \dots, \Pi_{j-1}^{(r-1)}, L_r, \Pi_j^{(r-1)}, \dots)$
  return $\Pi^{(n)}$
```

First, UNIVERSAL determines which items are swap items. It then starts with an empty permutation, and considers the items in order of non-decreasing sizes, inserting each item into the permutation. Swap items are always placed in front of all items already in the permutation, and all other items are inserted in front of the first item in the permutation that has a lower density.

We prove the following result.

▶ **Theorem 2.** *The algorithm* UNIVERSAL *constructs a universal policy of robustness factor* 2*.*

Before we prove this theorem, we first analyze the structure of the permutation output by UNIVERSAL in terms of density, size, and value. First, we prove that every item following a non-swap item has lower density.

▶ **Lemma 3.** *For a sequence* Π *returned by* UNIVERSAL*, we have* $\tilde{d}(\Pi_k) \succ \tilde{d}(\Pi_{k+1})$ *for every non-swap item* Π_k*,* $1 \leq k < n$*.*

Proof. For $j \in \{k, k+1\}$, let $r(j) \in \{1, \dots, n\}$ be the index of the iteration in which UNIVERSAL inserts Π_j into Π. We distinguish two cases.

If $r(k) < r(k+1)$, then the item Π_{k+1} cannot be a swap item, since it would appear in front of the item Π_k if it was. As each non-swap item is inserted into Π such that all items left of it are larger with respect to $\tilde{d}$, the claim follows.

If $r(k) > r(k+1)$, since it is not a swap item, Π_k is put in front of Π_{k+1} because it has a higher density. ◀

We prove that no item preceding a swap item has smaller size.

▶ **Lemma 4.** *For a permutation* Π *returned by* UNIVERSAL*, we have* $l(\Pi_j) \geq l(\Pi_k)$ *for every swap item* $\Pi_k, 1 < k \leq n$*, and every other item* $\Pi_j, 1 \leq j < k$*.*

Proof. Since Π_k is a swap item, it stands in front of all items inserted earlier into Π. Hence, all items that appear in front of Π_k in Π have been inserted in a later iteration of UNIVERSAL. Since UNIVERSAL processes items in order of non-decreasing sizes, we have $l(\Pi_j) \geq l(\Pi_k)$. ◀

We prove that no item preceding a swap item has smaller value.

▶ **Lemma 5.** *For a permutation Π returned by* Universal*, we have $v(\Pi_j) \geq v(\Pi_k)$ for every swap item $\Pi_k, 1 < k \leq n$, and every other item $\Pi_j, 1 \leq j < k$.*

Proof. We distinguish three cases.

First case: Π_j is a swap item and $\tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)$. By Lemma 4, we have $l(\Pi_j) \geq l(\Pi_k)$, and the claim trivially holds.

Second case: Π_j is a swap item and $\tilde{d}(\Pi_j) \prec \tilde{d}(\Pi_k)$. Since Π_j is a swap item, there is a capacity $C \geq l(\Pi_j)$ such that

$$v(\Pi_j) > v(\{i \in \mathcal{I} \mid l(i) \leq C \text{ and } \tilde{d}(i) \succ \tilde{d}(\Pi_j)\}).$$

In particular, for $C = l(\Pi_j)$ we obtain

$$v(\Pi_j) > v(\{i \in \mathcal{I} \mid l(i) \leq l(\Pi_j) \text{ and } \tilde{d}(i) \succ \tilde{d}(\Pi_j)\}). \tag{1}$$

Since, by Lemma 4, $l(\Pi_j) \geq l(\Pi_k)$, the item Π_k is included in the set on the right hand side of (1). We conclude that $v(\Pi_j) > v(\Pi_k)$.

Third case: Π_j is not a swap item. Let $\Pi_{j'}$ be the first swap item after Π_j in Π, i.e.,

$$j' = \min\{i \in \{j+1, \dots, k\} \mid \Pi_i \text{ is a swap item }\}.$$

Note that the minimum is attained as Π_k is a swap item. The analysis of the first two cases implies that $v(\Pi_{j'}) \geq v(\Pi_k)$. By Lemma 3 we have $\tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_{j+1}) \succ \cdots \succ \tilde{d}(\Pi_{j'})$, and by Lemma 4 we have $l(\Pi_j) \geq l(\Pi_{j'})$. Hence, $v(\Pi_j) \geq v(\Pi_{j'}) \geq v(\Pi_k)$. ◀

Finally, the next lemma gives a legitimation for the violation of the density order in the output permutation. Essentially, whenever an item precedes denser items, we guarantee that it is worth at least as much as all of them combined.

▶ **Lemma 6.** *For a permutation Π returned by* Universal*, we have*

$$v(\Pi_k) \geq v\big(\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\}\big)$$

for every item $\Pi_k, 1 \leq k < n$.

Proof. We distinguish whether Π_k is a swap item, or not.

If Π_k is a swap item, by definition, Π_k is worth more than the greedy set for some capacity $C \geq l(\Pi_k)$. Thus,

$$\begin{aligned} v(\Pi_k) &> v\big(\{\Pi_j \mid l(\Pi_j) \leq C \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\}\big) \\ &\geq v\big(\{\Pi_j \mid l(\Pi_j) \leq l(\Pi_k) \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\}\big). \end{aligned}$$

Since items whose size is strictly larger than $l(\Pi_k)$ are inserted into Π at a later iteration of Universal, they can only end up behind Π_k if they are smaller with respect to $\tilde{d}$. Hence,

$$\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\} \subseteq \{\Pi_j \mid l(\Pi_j) \leq l(\Pi_k) \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\},$$

and thus $v(\Pi_k) > v(\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\})$, as claimed.

If, on the other hand, Π_k is not a swap item, let $\Pi_{k'}$ be the first swap item after it in Π. If no such item exists, the claim holds by Lemma 3, since

$$\big\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\big\} = \emptyset.$$

Otherwise, by Lemma 3, we obtain $\tilde{d}(\Pi_k) \succ \tilde{d}(\Pi_{k+1}) \succ \cdots \succ \tilde{d}(\Pi_{k'})$ and hence

$$\begin{aligned}\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\} &= \{\Pi_j \mid j > k' \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\} \\ &\subseteq \{\Pi_j \mid j > k' \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_{k'})\}.\end{aligned}$$

Consequently, and by the argument above for swap items,

$$v(\Pi_{k'}) > v(\{\Pi_j \mid j > k' \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_{k'})\}) \geq v(\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) > \tilde{d}(\Pi_k)\})).$$

Finally, by Lemma 5, we have $v(\Pi_k) \geq v(\Pi_{k'}) \geq v(\{\Pi_j \,|\, j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\})$. ◀

We now prove Theorem 2.

Proof of Theorem 2. We show that for every item set $\mathcal{I}$, the permutation $\Pi = \textsc{Universal}(\mathcal{I})$ satisfies $v(\textsc{Opt}(\mathcal{I}, C)) \leq 2v(\Pi(C))$ for every capacity $C \leq l(\mathcal{I})$. By Theorem 1, it suffices to show $v(\Pi(C)) \geq v(\text{M}\textsc{Greedy}(\mathcal{I}, C))$ for all capacities. We distinguish between swap capacities and capacities where $\text{M}\textsc{Greedy}$ outputs a greedy set.

First, assume that C is a swap capacity, and let $\{\Pi_k\} = \text{M}\textsc{Greedy}(\mathcal{I}, C)$ be the swap item returned by the modified greedy algorithm. Then, $\Pi(C)$ contains at least one item Π_j with $j \leq k$. By Lemma 5, we have $v(\Pi(C)) \geq v(\Pi_j) \geq v(\Pi_k) = v(\text{M}\textsc{Greedy}(\mathcal{I}, C))$.

Now assume that C is not a swap capacity. Let $G^+ = \text{M}\textsc{Greedy}(\mathcal{I}, C) \setminus \Pi(C)$ be the set of items in the greedy set for capacity C that are not packed by the permutation Π. Similarly, let $U^+ = \Pi(C) \setminus \text{M}\textsc{Greedy}(\mathcal{I}, C)$. If $G^+ = \emptyset$, then $v(\Pi(C)) \geq v(\text{M}\textsc{Greedy}(\mathcal{I}, C))$ and we are done. Suppose now that $G^+ \neq \emptyset$. Then, also $U^+ \neq \emptyset$, since $\Pi(C)$ is inclusion maximal. For all items $i \in U^+$, we have $l(i) \leq C$ and $i \notin \text{M}\textsc{Greedy}(\mathcal{I}, C)$. Since C is not a swap capacity, $\text{M}\textsc{Greedy}(\mathcal{I}, C)$ is the greedy set for capacity C, and thus $\tilde{d}(i) \prec \tilde{d}(i')$ for all $i \in U^+$ and $i' \in G^+$. By definition of $\Pi(C)$ and since $U^+ \neq \emptyset$, we also have $k = \min\{j \mid \Pi_j \in U^+\} < \min\{k' \mid \Pi_{k'} \in G^+\}$, i.e., the first item $\Pi_k \in U^+$ in Π is encountered before every item from G^+. It follows that

$$G^+ \subseteq \big\{\Pi_j \mid j > k \text{ and } \tilde{d}(\Pi_j) \succ \tilde{d}(\Pi_k)\big)\big\}.$$

By Lemma 6, $v(\Pi_k) \geq v(G^+)$, and hence we obtain

$$\begin{aligned}v(\Pi(C)) &= v\big(\Pi(C) \cap \text{M}\textsc{Greedy}(\mathcal{I}, C)\big) + v(U^+) \\ &\geq v\big(\Pi(C) \cap \text{M}\textsc{Greedy}(\mathcal{I}, C)\big) + v(\Pi_k) \\ &\geq v\big(\Pi(C) \cap \text{M}\textsc{Greedy}(\mathcal{I}, C)\big) + v(G^+) = v(\text{M}\textsc{Greedy}(\mathcal{I}, C)).\end{aligned}$$ ◀

While it is obvious that Universal runs in polynomial time, we show that it can be modified to run in time $\Theta(n \log n)$. The proof is omitted due to space constraints.

▶ **Theorem 7.** *The algorithm* Universal *can be implemented to run in time* $\Theta(n \log n)$.

We now give a general lower bound on the robustness factor of any policy for the oblivious knapsack problem. This shows that Universal is best possible.

▶ **Theorem 8.** *For every* $\delta > 0$*, there are instances of the oblivious knapsack problem where no policy achieves a robustness factor of* $2 - \delta$.

Proof. We give a family of instances, one for each size $n \geq 3$. We ensure that for every item i of the instance of size n, there is a capacity C, such that packing item i first can only lead to a solution that is worse than $\textsc{Opt}(\mathcal{I}, C)$ by a factor of at least $(2 - 4/n)$. This completes the proof, as the factor approaches 2 for increasing values of n. The instance of size n is given by $\mathcal{I} = \{1, 2, \dots, n\}$ with $l(i) = F_n + F_i - 1$, and $v(i) = 1 + \frac{i}{n}$, where F_i denotes the i-th Fibonacci number ($F_1 = 1, F_2 = 1, F_3 = 2, \dots$).

We need to show that, no matter which item is tried first (i.e., no matter which item is the root of the policy), there is a capacity for which this choice ruins the solution. Observe that both values and sizes of the items are strictly increasing. Assume that item $i \geq 3$ is packed first. Since the smallest item has size $l(1) = F_n$, for capacity $C_i = 2F_n + F_i - 2 < 2F_n + F_i - 1 = l(1) + l(i)$, no additional item fits the knapsack. However, the unique optimum solution in this case is $\textsc{Opt}(\mathcal{I}, C_i) = \{i-1, i-2\}$. These two items fit the knapsack, as $l(i-1) + l(i-2) = 2F_n + F_{i-1} + F_{i-2} - 2 = 2F_n + F_i - 2 = C_i$. By definition,

$$\frac{v(i-1) + v(i-2)}{v(i)} = \frac{2n + 2i - 3}{n + i} = 2 - \frac{3}{n+i} \geq 2 - \frac{3}{n}\,.$$

Thus, policies that first pack item $i \geq 3$ cannot attain a robustness factor better than $2 - 3/n$.

Now, assume that one of the two smallest items is packed first. For capacity $C_{1,2} = l(n) = 2F_n - 1 < 2F_n = l(1) + l(2)$, no additional item fits the knapsack. The unique optimum solution, however, is to pack item n. It remains to compute the ratios

$$\frac{v(n)}{v(1)} > \frac{v(n)}{v(2)} = \frac{2n}{n+2} = 2 - \frac{4}{n+2} > 2 - \frac{4}{n}\,.$$

Hence, policies that first pack item 1 or item 2 do not achieve a robustness factor better than $2 - 4/n$. ◀

4 Unit Densities

In this section we restrict ourselves to instances of the oblivious knapsack problem, where all items have unit density, i.e., $v(i) = l(i)$ for all items $i \in \mathcal{I}$. For two items $i, j \in \mathcal{I}$ we say that i is smaller than j and write $i \prec j$ if $v(i) < v(j)$, or $v(i) = v(j)$ and $t(i) < t(j)$, where t is the tiebreaking index introduced in Section 3. We give an algorithm UniversalUD (cf. Algorithm 3) that produces a universal policy tailored to achieve the best possible robustness factor equal to the golden ratio $\varphi \approx 1.618$. The algorithm considers the items from smallest to largest, and inserts each item into the output sequence as far to the right as possible, such that the item is not preceded on its left by other items that are more than a factor φ smaller. Intuitively, the algorithm tries as much as possible to keep the resulting order sorted increasingly by size; only when an item dominates another item by a factor of at least φ the algorithm ensures that it precedes this item in the final sequence. Note that, even though φ is irrational, for rationals a, b the condition $a < \varphi b$ can be tested efficiently by testing the equivalent condition $a/b < 1 + b/a$.

▶ **Theorem 9.** *The algorithm* UniversalUD *constructs a universal policy of robustness factor φ when all items have unit density.*

Proof. Given an instance $\mathcal{I}$ of the oblivious knapsack problem with unit densities and any capacity $C \leq v(\mathcal{I})$, we compare the packing $\Pi(C)$ that results from the solution $\Pi = \textsc{UniversalUD}(\mathcal{I})$ with an optimal packing $\textsc{Opt}(\mathcal{I}, C)$. We define the set M of items in $\Pi(C)$ for which at least one smaller item is not in $\Pi(C)$, i.e., more precisely, let $M = \{i \in \Pi(C) \mid \exists j \in \mathcal{I} \backslash \Pi(C) : j \prec i\}$.

We first consider the case that $M \neq \emptyset$ and set $i = \min_{\prec} M$ to be the smallest item in M with respect to '$\prec$'. Consider the iteration r of UniversalUD in which i is inserted into Π, i.e., $i = L_r$. By definition of M, there is an item $j \prec i$ with $j \notin \Pi(C)$. Let j be

```
Algorithm 3: UniversalUD($\mathcal{I}$)
Input: set of items $\mathcal{I}$
Output: sequence of items $\Pi$
$L \leftarrow \langle$items in $\mathcal{I}$ sorted such that $L_1 \prec \cdots \prec L_n\rangle$
$\Pi^{(0)} \leftarrow \emptyset$
for $r \leftarrow 1, \ldots, n$ do
    $j \leftarrow 1$
    while $j \leq |\Pi^{(r-1)}|$ and $v(L_r) < \varphi v(\Pi_j^{(r-1)})$ do
        $j \leftarrow j + 1$
    $\Pi^{(r)} \leftarrow (\Pi_1^{(r-1)}, \ldots, \Pi_{j-1}^{(r-1)}, L_r, \Pi_j^{(r-1)}, \ldots)$
return $\Pi^{(n)}$
```

the first such item in Π. Since $j \prec i$, we have $j \in \Pi^{(r)}$. From $i \in \Pi(C)$ and $j \notin \Pi(C)$, it follows that i precedes j in Π (and thus in $\Pi^{(r)}$). Let i' be the item directly preceding j in $\Pi^{(r)}$. If $i' = i$, i was compared with j when it was inserted into $\Pi^{(r)}$, with the result that $v(i) \geq \varphi v(j)$ and thus $v(\Pi(C)) \geq \varphi v(j)$. If $i' \neq i$, by definition of j, we still have $i' \in \Pi(C)$. Also, either $i' \succ j$ and thus $v(i') \geq v(j)$, or j was compared with i' when it was inserted into Π in an earlier iteration of UniversalUD, with the result that $v(i') > \frac{1}{\varphi}v(j)$. Again, $v(\Pi(C)) \geq v(i) + v(i') > v(j) + \frac{1}{\varphi}v(j) = \varphi v(j)$.

In both cases it follows from $j \notin \Pi(C)$ that $v(\textsc{Opt}(\mathcal{I}, C)) \leq C < v(\Pi(C)) + v(j)$, and using $v(j) \leq \frac{1}{\varphi}v(\Pi(C))$ we get

$$\frac{v(\textsc{Opt}(\mathcal{I}, C))}{v(\Pi(C))} < \frac{v(\Pi(C)) + v(j)}{v(\Pi(C))} < 1 + \frac{1}{\varphi} = \varphi.$$

Now, assume that $M = \emptyset$. Intuitively, this means that $\Pi(C)$ consists of a prefix of L (the smallest items). Let $i_1 \succ \cdots \succ i_k$ be the items in $\Pi(C) \setminus \textsc{Opt}(\mathcal{I}, C)$, and let $j_1 \succ \cdots \succ j_l$ be the items in $\textsc{Opt}(\mathcal{I}, C) \setminus \Pi(C)$. As $\Pi(C)$ consists of a prefix of L, we have $|\Pi(C)| \geq |\textsc{Opt}(\mathcal{I}, C)|$ and thus $k \geq l$. If $k = 0$, the claim trivially holds. Otherwise, since M is empty, we have $j_l \succ i_1$. Is suffices to show $v(j_h) \leq \varphi v(i_h)$ for all $h \leq l$. To this end, we consider any fixed $h \leq l$. From $v(\{i_1, \ldots, i_{h-1}\}) \leq v(\{j_1, \ldots, j_{h-1}\})$ it follows that

$$v(j_h) \leq v(\textsc{Opt}(\mathcal{I}, C)) - v(\{j_1, \ldots, j_{h-1}\}) \leq C - v(\{i_1, \ldots, i_{h-1}\}).$$

This implies that j_h cannot precede all items of $\{i_h, \ldots, i_k\}$ in Π, as $j_h \notin \Pi(C)$. Hence, there is an item $i'' \in \{i_h, \ldots, i_k\}$ that precedes j_h in Π. Since $j_h \succ i''$, in the iteration when UniversalUD inserted j_h into Π, i'' was already present. From the fact that i'' ended up preceding j_h it follows that j_k was compared with i'' and thus $v(j_h) < \varphi v(i'') \leq \varphi v(i_h)$. We obtain

$$\frac{v(\textsc{Opt}(\mathcal{I}, C))}{v(\Pi(C))} \leq \frac{v(\textsc{Opt}(\mathcal{I}, C) \setminus \Pi(C))}{v(\Pi(C) \setminus \textsc{Opt}(\mathcal{I}, C))} = \frac{\sum_{h=1}^{l} v(j_h)}{\sum_{h=1}^{k} v(i_h)} \leq \frac{\sum_{h=1}^{l} \varphi v(i_h)}{\sum_{h=1}^{l} v(i_h)} = \varphi. \qquad ◀$$

A naïve implementation of UniversalUD runs in time $\Theta(n^2)$. We improve this running time to $\Theta(n \log n)$. The proof of the following theorem is omitted due to space constraints.

▶ **Theorem 10.** *The algorithm* UniversalUD *can be implemented to run in time* $\Theta(n \log n)$.

We now establish that UniversalUD is best possible, even if we permit non-universal policies (the proof is omitted due to space constraints).

▶ **Theorem 11.** *There are instances of the oblivious knapsack problem where no policy achieves a robustness factor of $\varphi - \delta$, for any $\delta > 0$, even when all items have unit density.*

5 Hardness

Although we can always find a 2-robust universal policy in polynomial time, we show in this section that, for any fixed $\alpha \in (1, \infty)$, it is intractable to decide whether a given universal policy is α-robust. This hardness result also holds for instances with unit densities when α is part of the input. As the final – and arguably the most interesting – result of this section, we establish coNP-hardness of the the problem to decide for a given instance and given $\alpha > 1$, whether the instance admits a universal policy with robustness factor α. All proofs rely on the hardness of the following version of SubsetSum.

▶ **Lemma 12.** *Let $W = \{w_1, w_2, \ldots, w_n\}$ be a set of positive integer weights and $T \leq \sum_{k=1}^{n} w_k$ be a target sum. The problem of deciding whether there is a subset $U \subseteq W$ with $\sum_{w \in U} w = T$ is* NP*-complete, even when*
1. *$T = 2^k$ for some integer $k \geq 3$,*
2. *all weights are in the interval $[2, T/2)$,*
3. *all weights have a difference of at least 2 to the closest power of 2.*

We first show that it is intractable to determine the robustness factor of a given universal policy.

▶ **Theorem 13.** *For any fixed and polynomially representable $\alpha > 1$ it is* coNP*-complete to decide whether a given universal policy for the oblivious knapsack problem is α-robust.*

Sketch of proof. For the proof of the coNP-hardness, we reduce from the variant of SubsetSum specified in Lemma 12. An instance of this problem is given by a set $W = \{w_1, w_2, \ldots, w_n\}$ of positive integer weights in the range $[2, T/2)$ and a target sum $T = 2^k$ for some integer $k \geq 3$. Let $\alpha > 1$ be polynomially representable. We construct an instance $\mathcal{I}$ and a sequence Π such that Π is an α-robust universal policy for $\mathcal{I}$ if and only if the instance of SubsetSum has no solution. First, we introduce for each weight $w \in W$ a *regular item* with $l(i_w) = v(i_w) = w$. This ensures that the optimal knapsack solution for capacity T is at least T if the instance of SubsetSum has a solution. We further introduce a set of additional items that ensure that the robustness factor for all capacities except T is at least α, regardless whether the instance of SubsetSum has a solution, or not. Specifically, let $m = \log_2 T - 1$. Then, for each $k \in \{0, 1, \ldots, m\}$ we introduce an *auxiliary item* j_k with size $l(j_k) = 2^k$ and value $v(j_k) = 2^k(1 - \epsilon)$, where $\epsilon > 0$ is suitably chosen depending on α. Intuitively, the set of auxiliary items ensures that all capacities less than T can be packed well enough. Finally, we introduce a *dummy item* with size $T + 1$ and large value. Intuitively, the dummy item ensures that all capacities larger than T can be packed well enough. The hardness is then established by showing that the sequence that contains first the dummy item, then the auxiliary items in decreasing order, and then the regular items is an α-robust universal solution if and only if the instance of SubsetSum has no solution. ◀

We give a result similar to Theorem 13 for unit densities. Note that this time we require α to be part of the input. The proof is technically more involved, and we only give a sketch due to space constraints.

▶ **Theorem 14.** *It is* coNP*-complete to decide whether, for given $\alpha > 1$, a given universal policy for the oblivious knapsack problem is α-robust, even when all items have unit density.*

Sketch of proof. We use a similar construction as in the proof of Theorem 13, with the additional complication that we are only permitted to use items of unit density. To meet this requirement, we modify the auxiliary items to have sizes and values equal to $(1-\varepsilon)2^k$, in such a way that all capacities up to $T-1$ are packed well (but not optimally) by our policy. We replace the dummy item with a series of items of exponentially growing sizes and values. These dummy items ensure that all capacities larger than T can be packed well. The crucial capacities are those close to T. For these capacities, we use that if the SubsetSum instance has no solution, a value of at most $T-\varepsilon$ can be packed, while if the instance has a solution, the optimum value is T. On the other hand, the policy we construct packs $(1-\varepsilon)(T-1)$. By setting $\alpha = \frac{T-\epsilon}{(1-\varepsilon)(T-1)}$, we ensure that our policy can only be α-robust if the SubsetSum instance has a solution. It remains to tune ε such that our policy is α-robust for all capacities (including fractional ones) except those close to T. Note that the proof relies on the fact that α is part of the input and may thus be set arbitrarily close to 1. ◀

Finally, we prove that it is hard to decide whether a given instance admits an α-robust universal policy when α is part of the input. The full proof is omitted due to space constraints.

▶ **Theorem 15.** *It is* coNP*-hard to decide whether, for given $\alpha > 1$, an instance of the oblivious knapsack problem admits an α-robust universal policy, even when all items have unit density.*

Sketch of proof. We consider the same set of items $\mathcal{I}$ as in the proof of Theorem 14. We show that, if an α-robust, universal policy exists, then it must be similar to the policy Π that we construct in the proof of Theorem 14. From this, we are able to infer that $\mathcal{I}$ admits an α-robust, universal policy if and only if Π is α-robust. As shown in the proof of Theorem 14, it follows that $\mathcal{I}$ admits an α-robust universal policy if and only if the underlying instance of SubsetSum has no solution. ◀

References

1 M. A. Bender, R. Cole, and E. D. Demaine. Scanning and traversing: maintaining data for traversals in a memory hierarchy. In *Proceedings of the 10th European Symposium on Algorithms (ESA)*, pages 139–151, 2002.

2 Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98:49–71, 2003.

3 A. Bhalgat, A. Goel, and S. Khanna. Improved approximation results for stochastic knapsack problems. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1647–1665, 2011.

4 P. C. Bouman, J. M. van den Akker, and J. A. Hoogeveen. Recoverable robustness by column generation. In *Proceedings of the 19th European Symposium on Algorithms (ESA)*, pages 215–226, 2011.

5 Christina Büsing, Arie M.C.A. Koster, and Manuel Kutschka. Recoverable robust knapsacks: the discrete scenario case. *Optimization Letters*, 5(3):379–392, 2011.

6 J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

7 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. In *Proceddings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 208–217, 2004.

8 Vladimir G. Deineko, Rüdiger Rudolf, and Gerhard J. Woeginger. Sometimes travelling is easy: The master tour problem. In *Proceedings of the 3rd European Symposium on Algorithms (ESA)*, pages 128–141. Springer, 1995.

9 Leah Epstein, Asaf Levin, Alberto Marchetti-Spaccamela, Nicole Megow, Julian Mestre, Martin Skutella, and Leen Stougie. Universal sequencing on an unreliable machine. *SIAM Journal on Computing*, 41(3):565–586, 2012.

10 M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS)*, pages 285–297, 1999.

11 Kai-Simon Goetzmann, Sebastian Stiller, and Claudio Telha. Optimization over integers with robustness in cost and few constraints. In *Proceedings of the 9th Workshop on Approximation and Online Algorithms (WAOA)*, pages 89–101, 2011.

12 Refael Hassin and Shlomi Rubinstein. Robust matchings. *SIAM Journal on Discrete Mathematics*, 15(4):530–537, 2002.

13 Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for TSP, Steiner tree, and set cover. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 386–395, 2005.

14 Naonori Kakimura, Kazuhisa Makino, and Kento Seimi. Computing knapsack solutions with cardinality robustness. In *Proceedings of the 22nd International Conference on Algorithms and Computation (ISAAC)*, pages 693–702, 2011.

15 Bernhard Korte and Jens Vygen. *Combinatorial Optimization. Theory and Algorithms.* Springer, 2nd edition, 2002.

16 Nicole Megow and Julián Mestre. Instance-sensitive robustness guarantees for sequencing with unknown packing and covering constraints. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 495–504, 2013.

17 Ralph Merkle and Martin E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978.

18 Michele Monaci and Ulrich Pferschy. On the robust knapsack problem. In *Proceedings of the 10th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW)*, pages 207–210, 2011.

19 Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

20 Harald Räcke. Survey on oblivious routing strategies. In *Proceedings of the 5th Conference on Computability in Europe: Mathematical Theory and Computational Practice (CiE)*, pages 419–429, 2009.

21 Adi Shamir. A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. In *Proceedings of the 23rd Symposium on Foundations of Computer Science (FOCS)*, pages 145–152, 1982.

22 L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC)*, pages 263–277, 1981.

23 Gang Yu. On the max-min 0-1 knapsack problem with robust optimization applications. *Operations Research*, 44(2):407–415, 1996.