Maximizing a Monotone Submodular Function Under an Unknown Knapsack Capacity

Sven de Vries, Sabine Münch, Stephen Raach

Trier University, 54286 Trier, Germany
{devries / muench / raach}@uni-trier.de

Consider the problem of maximizing a nondecreasing submodular function defined on a set of weighted items under an unknown knapsack capacity. Assume items are packed sequentially into the knapsack, and the knapsack capacity is accessed through an oracle that answers whether an item fits into the currently packed knapsack. If an item is tried to be added and fits, it is packed irrevocably; if the addition exceeds the knapsack capacity, it is removed from consideration. We consider *nonadaptive* packing according to a *predetermined* sequence, denoted universal policy. We present an algorithm to compute universal policies that perform for any unknown but fixed capacity (which is assumed to be greater or equal to the heaviest item), at least as good as the classic algorithm due to Wolsey, which packs the knapsack according to steepest ascent and outputs the better of the currently packed knapsack and the first item that exceeds the knapsack capacity, applied to the same capacity.

As a byproduct, we obtain an adaptive algorithm for the problem of maximizing a monotone-increasing submodular function and a universal policy for maximizing a modular function, respectively, under an unknown knapsack capacity. Our algorithms perform at least as well as the best known algorithms for the respective problem but are far more intuitive and allow for simpler proofs of correctness than the ones provided in the literature. Furthermore, submodular instances exist where our adaptive algorithm performs strictly better than the best known adaptive algorithm from the literature.

Keywords : *Combinatorial optimization, submodular maximization, knapsack, unknown capacity, nonadaptive.*

1. Introduction.

Optimization problems involving submodular objective functions are a central issue in combinatorial optimization. The importance lies in their natural appearance across

various applications and their relevance to classical optimization problems, e.g., graph cuts [see e.g., 7], set cover problems [see e.g., 6], and facility location problems [see e.g., 3] and [see e.g., 1] .

Definition 1. *In the following, let $(I, \prec)$ be a strictly totally ordered finite set of cardinality $n \in \mathbb{N}$, and $f : 2^I \to \mathbb{R}_{\geq 0}$ a* ***normalized*** *($f(\emptyset) = 0$),* ***monotone-increasing*** *($f(X) \leq f(Y)$ for any $X, Y \subseteq I$ with $X \subseteq Y$) and* ***submodular*** *($f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$ for $X, Y \subseteq I$) function. For ease of notation, let $f(i) \coloneqq f(\{i\})$ for $i \in I$. Let every item $i \in I$ be associated with a weight $w(i) \in \mathbb{R}_{\geq 0}$. We denote $w(T) \coloneqq \sum_{i \in T} w(i)$ for the weight of a set $T \in 2^I$.*

Consider the classic problem of maximizing a submodular function under a knapsack constraint (SMKC), thus,

$$\max\{f(X) \colon \sum_{i \in X} w(i) \leq B,\, X \subseteq I\}.$$

Even though this problem is known to be an NP-hard problem, a polynomial time approximation algorithm exists due to Sviridenko [13], combining partial enumeration with discrete steepest ascent. This algorithm yields an approximation factor of $1 - \frac{1}{e} \approx 0.63$, where the approximation factor is the worst-case ratio between the objective value of the solution obtained by the algorithm and the objective value of the optimal solution. Furthermore, Wolsey [14] showed that the more simple algorithm of taking the better of the discrete steepest ascent solution and the first item exceeding the current, by discrete steepest ascent packed knapsack, has an approximation factor of at least $1 - e^{-\beta} \approx 0.357$, where β is the unique solution of $e^x = 2 - x$, $x \in \mathbb{R}$.

Definition 2. *In the following, let β denote the solution of $e^x = 2 - x$, $x \in \mathbb{R}$ and $\omega \coloneqq 1 - e^{-\beta}$.*

Both approximation results Sviridenko [13] and Wolsey [14] require complete knowledge of the instance. However, in several applications, there might be uncertainty about the knapsack capacity, the item weights, or the submodular function.

Consider the situation in which the government seeks to provide mobile internet coverage and has tasked a company to construct transmission towers. There is a set of possible locations I to build towers, each associated with construction costs $w(i) \geq 0$. The value of a set of transmission towers $T \subseteq I$, i.e., their coverage of the region with mobile internet, can be measured by a submodular function $f \colon 2^I \to \mathbb{R}_{\geq 0}$. The project manager determines the order in which these transmission towers will be constructed and delegates the construction process in that order to the construction team. The government desires a high level of mobile internet coverage, reflected by the performance of a set T, with a large $f(T)$. However, the government does not disclose the exact budget B for the project to the project manager as the construction begins before the

financing is finalized. At some point, the budget will run out, and the government will inquire about the results. The project manager then presents the set of completed transmission towers and evaluates their region coverage with mobile internet. Consequently, the project manager's objective is to organize the construction of transmission towers so that, for any budget B, the set T of built towers delivers good coverage $f(T)$, in comparison to the optimal selection of towers if the budget were known beforehand. Thus, in this application, the knapsack capacity B is an unknown parameter.

An instance of the submodular maximization problem under an unknown knapsack capacity (denoted in the following by SMUC) is a tuple (I, f, w) with I being a set of items i with weights $w(i) \geq 0$, and with f being a monotone-increasing submodular set function, defined on I. Thus, any instance of SMUC is independent of a knapsack capacity B.

Definition 3. *Let I a set of items i with weights $w(i)$, then any knapsack capacity $B \geq \max_{i \in I} w(i)$ is called **reasonable**.*

There are two possible approaches to SMUC: maximizing the submodular function *nonadaptively* or *adaptively*. In the nonadaptive approach an instance of SMUC is given and the items shall be ordered.

Definition 4. *Given a finite set of items I, any permutation $N = (N_1, N_2, \ldots, N_n) \in I^n$ with $N_i \neq N_j$ for $1 \leq i < j \leq n$ is called a **universal policy**.*

After ordering the items, a knapsack capacity is revealed, and items are added one by one to the knapsack according to the universal policy. If the addition of an item exceeds the knapsack capacity, there are two ways to proceed:

- In *packing without discarding*, packing the knapsack is stopped, and the current solution is returned.
- In *packing with discarding*, the item that exceeds the knapsack capacity is discarded. Then, packing continues with the next item in the universal policy until all items are packed or discarded.

In contrast, in the adaptive approach, items are packed one after another, and the choice of which item to pack next may depend on the observations made while deciding on the previous items.

Nonadaptive maximization of a submodular function due to an unknown knapsack capacity is related to the following leader-follower problem: The leader decides on a universal policy aiming to maximize the *ratio* between the value of the set of items packed into the knapsack according to the policy and the value of an optimally packed knapsack. This ratio is contingent upon the knapsack capacity, determined by the follower aiming to minimize the ratio while knowing the universal policy chosen by the leader. The optimal

solution to this leader-follower problem returns an *optimal* universal policy. However, even knowing an optimal solution to the leader-follower problem does not indicate how well the optimal policy performs compared to optimal packing with knowledge of the knapsack's capacity.

1.1. Our Result.

We provide results for nonadaptive and adaptive maximizing a submodular function under an unknown knapsack capacity. Our main contribution is an algorithm that computes for any instance of SMUC a universal policy such that packing for any fixed *reasonable* knapsack capacity according to that policy yields a set of items that is at least as valuable as the output of Wolsey's algorithm [14], for the same (known) knapsack capacity.

Therefore, in terms of the aforementioned leader-follower problem, the present paper not only finds an approximately optimal solution to the leader-follower problem, but also demonstrates that packing according to the approximately optimal solution of the leader-follower problem constantly approximates the optimal packing for any knapsack capacity.

As a byproduct of our findings, we obtain an adaptive algorithm for SMUC that performs for any *arbitrary* unknown knapsack capacity at least as well as Wolsey's algorithm [14] does with knowledge about the capacity. Thus, the presented adaptive algorithm performs at least as well as the algorithm proposed by Klimm and Knaack [9] that matches the same approximation factor of 0.357. For some instances of SMUC, the items packed by our adaptive algorithm are even more valuable than those packed by the adaptive algorithm due to Klimm and Knaack [9].

Regarding nonadaptive maximization of a modular function, we show that one can pack an *arbitrary* unknown but fixed knapsack nonadaptively at least as well as Wolsey's algorithm [14] does for the same known knapsack capacity. Thereby, we give a more straightforward proof for the $\frac{1}{2}$-approximation factor by Disser et al. [4].

1.2. Related Work.

Submodular maximization with various additional constraints is a central topic in combinatorial optimization. Nemhauser et al. [12] initiated research on this subject and considered the problem of maximizing a monotone-increasing submodular function under a knapsack constraint with unit weights. For this problem, Nemhauser et al. [12] showed that a greedy algorithm achieves an approximation factor of $(1 - \frac{1}{e}) \approx 0.63$. Furthermore, Feige [6] established that even for the maximum coverage problem with unit weights under a knapsack constraint, which is a special case of maximizing a monotone-increasing submodular function under a knapsack constraint, no better approximation factor is possible in polynomial time, unless $P = NP$.

For the problem of maximizing a monotone-increasing submodular function under a knapsack constraint with arbitrary nonnegative weights, Wolsey [14] presented a special version of a steepest ascent algorithm with an approximation factor of ω, and Sviridenko [13] showed that combining a partial enumeration procedure with steepest ascent achieves the best possible approximation factor of $1 - \frac{1}{e}$.

Navarra and Pinotti [11] and Disser et al. [4] considered maximization of a modular function under an unknown knapsack capacity *nonadaptively*. They both present an algorithm that computes for any instance a universal policy such that the knapsack *packed without discarding*, respectively, *with discarding*, approximates the optimal solution by a factor of at least $\frac{1}{2}$, where Navarra and Pinotti [11] assume a reasonable knapsack capacity.

Disser et al. [5] consider the more general problem of nonadaptively maximizing a fractionally subadditive function under an unknown knapsack capacity. However, their policy fails to achieve a constant robustness factor but relies on the greatest item value. Nonadaptive maximization of a strict superset of submodular functions under an unknown cardinality constraint is considered by Bernstein et al. [2].

Kawase et al. [8] considered *adaptively* maximizing a monotone submodular function under an unknown knapsack capacity. They demonstrated that an adaptive packing algorithm can approximate the value of the optimal packing by a factor of at least $\frac{2(1-1/e)}{21} > 0.06$. As mentioned, this result was improved by Klimm and Knaack [9], achieving ω.

1.3. Outline.

In Section 2, we review Wolsey's algorithm [14] for SMKC. In Section 3, we nonadaptively approach SMUC without discarding under the assumption of a reasonable knapsack capacity. In Section 4, we consider SMUC nonadaptively and adaptively, *without* the reasonable knapsack capacity assumption. In Section 5, we nonadaptively approach the special case of maximizing a modular function under an unknown arbitrary knapsack capacity.

2. Maximization of a Submodular Function with a Known Knapsack Constraint.

We start by recapitulating Wolsey's algorithm [14] for SMKC.

Algorithm 1:

Input: Set of items I, each $i \in I$ associated with weight $w(i) \geq 0$, submodular function f, knapsack capacity B.

Output: Subset of I.

1 $I \leftarrow I \setminus \{i \in I \colon w(i) > B\}$
2 $R \leftarrow \emptyset$
3 **while** $I \setminus R \neq \emptyset$ **do**
4 $\quad i^* \in \arg\max_{i \in I \setminus R} \left\{ \frac{f(R \cup \{i\}) - f(R)}{w(i)} \right\}$
5 $\quad$ **if** $w(R \cup \{i^*\}) \leq B$ **then**
6 $\quad\quad R \leftarrow R \cup \{i^*\}$
7 $\quad$ **else**
8 $\quad\quad$ **break**
9 **return** $\arg\max\{f(R), f(i^*)\}$

Algorithm 1 starts by deleting all items with a weight greater than the knapsack capacity from I, since these items can never be part of a feasible solution of SMKC. Algorithm 1 then initializes $R = \emptyset$ and chooses in every iteration some item i^* from the set of the remaining items $I \setminus R$ that maximizes the relative increase $\frac{f(R \cup \{i\}) - f(R)}{w(i)}$ of the objective function. If $w(R \cup \{i^*\}) \leq B$, thus, if item i^* fits into the currently packed knapsack, item i^* is added to the set R. If item i^* does not fit into the currently packed knapsack, the while loop in Line 3 breaks. If all items fit into the knapsack, Algorithm 1 returns the entire set in Line 9. Otherwise, Algorithm 1 compares the present solution R with the first item i^*, which did not fit into the currently packed knapsack, and returns the better of R and i^*. Informally, the basic idea of returning the better of R and i^* is to avoid getting stuck in the first local optimum.

Definition 5. *Let (I, f, w) an instance of* SMUC *and B a knapsack capacity, then denote the objective function value of the output of Algorithm 1 as $\Phi(I, f, w, B)$. The* ***approximation factor*** *of Algorithm 1 is defined as*

$$\min_{\substack{(I,f,w) \text{ is an instance of SMUC,} \\ B \geq 0}} \frac{\Phi(I, f, w, B)}{f(\mathrm{Opt}^B)},$$

where Opt^B denotes the set of items included in an optimal solution of the submodular maximization problem with knapsack capacity B.

Thus, the approximation factor of Algorithm 1 is the worst-case ratio between the objective value of the output of Algorithm 1 and the optimal solution over all instances of SMUC and all knapsack capacities.

Proposition 1. *[14] Algorithm 1 has an approximation factor of at least ω.*

The proof of Proposition 1 requires the following Proposition, which will also be used later in our proof of Theorem 2.

Proposition 2. *[14, Theorem 2] Let (I, f, w) an instance of* SMUC*, let B a knapsack capacity, and R^B be the set constructed in the while loop in Line 3 of Algorithm 1 applied to the instance (I, f, w) with the knapsack capacity B. If $w(R^B) = B$, then $\frac{f(R^B)}{f(\mathrm{Opt}^D)} \geq 1 - e^{-\frac{B}{D}}$ for any capacity $D \geq B$.*

Wolsey [14] showed that the approximation factor of Algorithm 1 is at least ω. For completeness, we demonstrate that no better bound is possible.

Theorem 1. *Algorithm 1 cannot generally provide a better approximation than ω.*

Proof. Let $n \in N$, and (I, f, w) an instance of SMUC with $I = \{a_1, a_2, \ldots, a_{\lceil \beta n \rceil - 1}, a_{\lceil \beta n \rceil}, x\}$ [1], $w(a_i) = \frac{1}{n}$ for all $i \in \{1, 2, \ldots, \lceil \beta n \rceil - 1\}$, $w(a_{\lceil \beta n \rceil}) = 1 - \beta + \frac{2}{n}$, and $w(x) = 1$, $f \colon 2^I \to \mathbb{R}_{\geq 0}, f(T) \mapsto \min\{\sum_{t \in T} v(t), 1\}$ with $v(a_i) = \frac{1}{n}\left(1 - \frac{1}{n}\right)^{i-1}$ for all $i \in \{1, 2, \ldots, \lceil \beta n \rceil - 1\}$, $v(a_{\lceil \beta n \rceil}) = (1 - \beta + \frac{2}{n})\left(1 - \frac{1}{n}\right)^{\lceil \beta n \rceil - 1}$, and $v(x) = 1$. Let $B = 1$ the knapsack capacity. It holds

$$\frac{f(a_1) - f(\emptyset)}{w(a_1)} = 1 = \frac{f(x) - f(\emptyset)}{w(x)} > \frac{f(a_i) - f(\emptyset)}{w(a_i)} \text{ for all } i \in \{2, \ldots, \lceil \beta n \rceil\} \text{ and}$$

$$\frac{f\left(\bigcup_{j=1}^{i}\{a_j\}\right) - f\left(\bigcup_{j=1}^{i-1}\{a_j\}\right)}{w(a_i)} = \frac{f(a_i)}{w(a_i)} = \left(1 - \frac{1}{n}\right)^{i-1} = 1 - \frac{1}{n}\sum_{k=0}^{i-2}\left(1 - \frac{1}{n}\right)^{k}$$

$$= \frac{f\left(\bigcup_{j=1}^{i-1}\{a_j\} \cup \{x\}\right) - f\left(\bigcup_{j=1}^{i-1}\{a_j\}\right)}{w(x)} \geq \left(1 - \frac{1}{n}\right)^{k} = \frac{f\left(\bigcup_{j=1}^{i-1}\{a_j\} \cup \{a_k\}\right) - f\left(\bigcup_{j=1}^{i-1}\{a_j\}\right)}{w(a_k)}$$

for all $i \in \{2, \ldots, \lceil \beta n \rceil\}$ and $k > i$.

Therefore, Algorithm 1 chooses during the first $\lceil \beta n \rceil - 1$ iterations of the while loop in Line 3 the items $a_1, \ldots, a_{\lceil \beta n \rceil - 1}$. In the next iteration of the while loop Algorithm 1 chooses item $a_{\lceil \beta n \rceil}$. Then, it holds $\sum_{j=1}^{\lceil \beta n \rceil} w(a_j) \geq \beta - \frac{1}{n} + 1 - \beta + \frac{2}{n} \geq 1$ and the while loop ends in Line 8. Therefore, Algorithm 1 outputs $\arg\max\{f(\bigcup_{j=1}^{\lceil \beta n \rceil - 1}\{a_j\}), f(a_{\lceil \beta n \rceil})\}$. However, the optimal solution to the instance (I, f, w) with knapsack capacity $B = 1$ is $\{x\}$ with $f(x) = 1$. It holds $\lim_{n \to \infty} \frac{\Phi(I,f,w,B)}{f(x)} = \lim_{n \to \infty} \frac{\max\{f(\bigcup_{j=1}^{\lceil \beta n \rceil - 1}\{a_j\}), f(a_{\lceil \beta n \rceil})\}}{f(x)} = \frac{\max\{1 - e^{-\beta}, 1 - e^{-\beta}\}}{1} = \omega$ and the claim follows. □

It follows directly by Proposition 1 and Theorem 1.

Corollary 1. *Algorithm 1 has an approximation factor of ω.*

[1] where β was defined as the solution of $e^x = 2 - x,\ x \in \mathbb{R}$.

3. Maximization of a Submodular Function under an Unknown Knapsack Capacity Nonadaptively.

This section considers the nonadaptive approach to SMUC, assuming reasonable knapsack capacities. Our goal is to develop an algorithm that determines for any instance of SMUC a universal policy such that, for any reasonable knapsack capacity, packing *without discarding* according to the universal policy is at least as good as the result of Algorithm 1. We formally define:

Definition 6. *Given an instance (I, f, w) of* SMUC*, a universal policy N and a knapsack capacity B, the* ***packed set*** *(packed without discarding) $K(N, B)$ is defined as*

$$K(N,B) \coloneqq \left\{ N_j \colon j \leq k,\ k = \max\left\{ l \colon 1 \leq l \leq n,\ \sum_{i=1}^{l} w(N_i) \leq B \right\} \right\},$$

The ***value function*** *g^N of the universal policy N assigns to any knapsack capacity B the value of the corresponding packed set $K(N, B)$, and is denoted by*

$$g^N \colon \mathbb{R}_+ \to \mathbb{R}_+,\ g^N(B) \coloneqq f(K(N,B)).$$

The ***robustness factor*** *of a universal policy N is defined as*

$$\min_{B \geq \max_{i \in I} w(i)} \frac{g^N(B)}{f(\mathrm{Opt}^B)}.$$

A universal policy N is called ***better or equal*** *than Algorithm 1 if $g^N(B) \geq \Phi(I, f, w, B)$ for any $B \geq \max_{i \in I} w(i)$.*

Clearly, given an instance of SMUC, any universal policy better or equal than Algorithm 1 has a robustness factor of at least ω, by Proposition 1.

Notice that a reasonable knapsack capacity is necessary to provide a constant robustness factor for *packing without discarding* by any universal policy at all:

Example 1. *[11] Let $I = \{a, b\}$, with $w(a) = 1$ and $w(b) = 1 + \varepsilon$, $\varepsilon \geq 0$, and $f : 2^I \to \mathbb{R}_{\geq 0}$ given by $f(\emptyset) = 0$, $f(a) = 1$, $f(b) = f(\{a, b\}) = M$ for $M \geq 1$. We demonstrate that for this instance of* SMUC *no universal policy can achieve a constant robustness factor as $M \to \infty$.*

Let $N = (b, a)$. Then, for $B = 1$ it holds $g^N(B) = 0$, since $w(b) > 1$. However the optimal solution to $\max\{f(X) \colon \sum_{i \in X} w(i) \leq 1, X \subseteq I\}$ is $\mathrm{Opt}^B = \{a\}$, and therefore $\frac{g^N(B)}{f(\mathrm{Opt}^B)} = \frac{0}{1} = 0$.

Let $\bar{N} = (a, b)$. Then, for $\bar{B} = 1 + \varepsilon$ it holds $g^{\bar{N}}(\bar{B}) = 1$, since $w(a) < 1 + \varepsilon$ and $w(a) + w(b) > 1 + \varepsilon$. However, the optimal solution to $\max\{f(X) \colon \sum_{i \in X} w(i) \leq 1 + \varepsilon, X \subseteq I\}$ is $\mathrm{Opt}^{\bar{B}} = \{b\}$, and therefore $\frac{g^{\bar{N}}(\bar{B})}{f(\mathrm{Opt}^{\bar{B}})} = \frac{1}{M}$ and it holds $\lim_{M \to \infty} \frac{g^{\bar{N}}(\bar{B})}{f(\mathrm{Opt}^{\bar{B}})} = 0$.

However, under the reasonable knapsack capacity assumption, which rules out $B < 1+\varepsilon$, in Example 1 it holds for the policy for $N = (b, a)$ that

$$\min_{B \geq \max_{i \in I} w(i)} \frac{g^N(B)}{f(\mathrm{Opt}^B)} = 1,$$

since $g^N(B) = f(\{b\}) = f(\mathrm{Opt}^B)$ for $1+\varepsilon \leq B < 2+\varepsilon$, and $g^N(B) = f(\{a,b\}) = f(\mathrm{Opt}^B)$ for $B \geq 2+\varepsilon$.

3.1. From Modular Maximization to Submodular Maximization.

It is a natural idea to try to adapt existing results for *maximization of a modular* (i.e. linear) *function under an unknown knapsack capacity* (denoted in the following by MMUC) to SMUC. Thus, before we consider SMUC with a general monotone-increasing submodular function, we revisit the special case of a modular function. For MMUC, assuming a reasonable knapsack capacity and packing without discarding, Navarra and Pinotti [11] presented an algorithm, described below, which returns a universal policy with a robustness factor of at least $\frac{1}{2}$.

Algorithm 2:

Input: Set of items I, each $i \in I$ associated with weight $w(i)$, and value $f(i)$.
Output: Universal policy N.

$N \leftarrow (N_1, N_2, \ldots, N_{|I|})$ Items ordered non-increasingly by $\frac{f(i)}{w(i)}$.
$k \leftarrow 1$
while $\sum_{j=1}^{k} w(N_j) \leq \max_{i \in I} w(i)$ **and** $k \leq |I|$ **do**
  $k \leftarrow k+1$
if $f(N_k) > \sum_{j=1}^{k-1} f(N_j)$ **then**
  $N \leftarrow (N_k, N_1, N_2, \ldots, N_{k-1}, N_{k+1}, \ldots, N_{|I|})$
return N

Before generalizing Algorithm 2 to arbitrary submodular objective functions, we briefly describe it. In Algorithm 2, the initial order N is given by sorting the items according to their ratio $f(i)/w(i)$. Then, Algorithm 2 identifies the first item N_k in the initial ratio-order N for which the set $\{N_1, \ldots, N_k\}$ is heavier than the heaviest item. If the objective value of N_k is better than the sum of the values of the previous items $N_1, \ldots, N_{k-1}$, the order N is updated by moving N_k to the front. Subsequently, the updated order is returned. Notice that the final order is generated by updating the initial order at most once.

Definition 7. *Let (I, f, w) an instance of* SMUC. *We call the tuple $S = (S_1, \ldots, S_n) \in I^n$* ***steepest ascent*** *ordered, if*

$$S_j \in \arg\max \left\{ \frac{f(\{S_1, \ldots, S_{j-1}, x\}) - f(\{S_1, \ldots, S_{j-1}\})}{w(x)} : x \in I \setminus \{S_1, \ldots, S_{j-1}\} \right\} \text{ for all}$$

$1 \leq j \leq n$ *and* $S_j \prec S_k$ *for all* $S_k \in \arg\max \left\{ \frac{f(\{S_1, \ldots, S_{j-1}, x\}) - f(\{S_1, \ldots, S_{j-1}\})}{w(x)} : x \in I \setminus \{S_1, \ldots, S_{j-1}\} \right\}$.

Notice that the condition $S_j \prec S_k$ for all $S_k \in \arg\max \left\{ \frac{f(\{S_1,\ldots,S_{j-1},x\}) - f(\{S_1,\ldots,S_{j-1}\})}{w(x)} : x \in I \setminus \{S_1, \ldots, S_{j-1}\} \right\}$ serves only as a tie-breaking rule in order to make the steepest ascent order unique. Although Algorithm 2 intended for MMUC, a slightly *modified* version can be applied to SMUC. First, we replace the sorting according to the ratio $f(i)/w(i)$ in Line 1 by

---

1' $N \leftarrow (N_1, \ldots, N_{|I|})$ steepest ascent ordered.

Observe that for modular objective functions both sortings coincide. Second, the condition $f(N_k) > \sum_{j=1}^{k-1} f(N_j)$ in Line 5, which determines whether to move N_k to the first position of N or not, is replaced by the condition

---

5' **if** $f(N_k) > f(\{N_1, \ldots, N_{k-1}\})$ **then**

Again, for modular objective functions, these conditions coincide.

The following example presents an instance of SMUC for which the *modified* Algorithm 2 returns a universal policy that is not better or equal to Algorithm 1, even though the value function provides an approximation to the value of an optimal solution of at least ω.

Example 2. *Let* $I = \{a, b, c\}$ *be a set of items with weights* $w(a) = 1$, $w(b) = 1.2$ *and* $w(c) = 2.1$, *and* $f : 2^I \to \mathbb{R}_{\geq 0}, T \mapsto \min\{\sum_{t \in T} v(t), 2\}$, *with* $v(a) = 1$, $v(b) = 0.6$, *and* $v(c) = 2$. *The* modified *Algorithm 2 starts with the steepest ascent order* $N = S = (a, b, c)$. *Since* $w(\{a, b\}) = 2.2 > 2.1 = \max_{i \in I} w(i)$ *the while loop in the* modified *Algorithm 2 ends with* $k = 2$ *and because of* $f(N_k) = f(b) = 1 = f(a)$ *the unchanged order* N *is returned. Let* $B = 3$ *be the capacity of the knapsack. Then,* $g^N(3) = 1.6$, *since* $w(\{a, b\}) = 2.2 < 3$ *and* $w(\{a, b, c\}) = 4.3 > 3$. *In contrast, Algorithm 1 returns the single item* c *in Line 9, since* $f(c) = 2 > f(\{a, b\}) = 1.6$, *which is an optimal solution. Therefore* $f(\mathrm{Opt}^3) = \Phi(I, f, w, 3) > g^N(3) \geq \omega f(\mathrm{Opt}^3)$.

Although the universal policy returned by the *modified* Algorithm 2 does not compare favorably to Algorithm 1, it achieves the same approximation factor.

Theorem 2. *Let* (I, f, w) *an instance of* SMUC, *then any universal policy returned by the* modified *Algorithm 2 has a robustness factor of at least* ω.

The proof of Theorem 2 is deferred to the Appendix.

3.2. Matching the Approximation Factor of Algorithm 1 Nonadaptively.

Recall that we aim for every instance of SMUC to construct a universal policy better or equal to Algorithm 1. As a simple consequence of Example 2, any universal policy that wants to imitate Algorithm 1 has to treat a single item i^* returned by Algorithm 1, especially since it might be necessary to place i^* at the beginning of the universal policy.

Definition 8. *Let (I, f, w) be an instance of* SMUC *and $S = (i_1, \dots, i_{|I|})$ the steepest ascent order of I. For $2 \leq j \leq |I|$ call i_j a* ***swap item*** *if $f(i_j) > f(\{i_1, \dots, i_{j-1}\})$.*

We modify the steepest ascent order by identifying the swap items and moving them to the front of the universal policy. This is formalized in the following algorithm.

Algorithm 3:

Input: An instance (I, f, w) of SMUC.
Output: Universal policy N.

```
1 Determine the steepest ascent order S
2 T ← {i ∈ I : i is a swap item}
3 N ← S
4 for j = 2, ..., n do
5     if N_j ∈ T then
6         N ← (N_j, N_1, ..., N_{j-1}, N_{j+1}, ..., N_n)
7 return N
```

Algorithm 3 starts by ordering the items of I according to the steepest ascent order S and determining the set T of all swap items. Then, each iteration of the for loop in Line 4 checks whether N_j is a swap item. If N_j is a swap item, the current order N is updated by placing N_j in front of all other items. Otherwise, the current order remains the same. Note that whenever item N_j is identified as a swap item in Line 5 of Algorithm 3 and moved to the front of N, any item N_m with $m > j$ remains in its position in the updated order N in Line 6 and the relative position for any pair of items (i_k, i_l) with $1 \leq k < l < j$ remains constant.

Theorem 3. *Let (I, f, w) an instance of* SMUC*, then, any universal policy N, returned by Algorithm 3, yields $g^N(B) \geq \Phi(I, f, w, B)$ for any knapsack capacity $B \geq \max_{i \in I} w(i)$.*

The proof of Theorem 3 uses the following observation, which follows directly by the Lines 5 and 6 of Algorithm 3.

Lemma 1. *Let (I, f, w) an instance of* SMUC *and N the universal policy returned by Algorithm 3. If N_k is a swap item (in the steepest ascent order S), then every N_j with $j < k$ is a swap item (in S), and it holds $f(N_j) > f(N_k)$.*

Now, we prove Theorem 3.

Proof of Theorem 3. Let N be the universal policy returned by Algorithm 3, S the steepest ascent order in Line 1 of Algorithm 3, and $B \geq \max_{i \in I} w(i)$. Assume without loss of generality that Algorithm 1 inspects the items in Line 4 according to S. In order to compare $g^N(B)$ with the objective value of the output of Algorithm 1, we distinguish two cases.

Case 1: Algorithm 1 outputs in Line 9 the single item i^* with $i^* = S_k$, $k > 1$. Then, when the while loop in Line 3 of Algorithm 1 ends, it holds $R = \{S_1, \ldots, S_{k-1}\}$ and $f(S_k) > f(R)$. It follows directly $g^N(B) \geq f(N_1) \geq f(S_k) = \Phi(I, f, w, B)$, where the second inequality follows by Lemma 1.

Case 2: Algorithm 1 returns the set $R = \{S_1, \ldots, S_{k-1}\}$ with $2 \leq k \leq n$. Clearly, S_k is not a swap item, since otherwise Algorithm 1 would have returned S_k. If there exists no swap item S_j with $j > k$, then, $\{S_1, \ldots, S_{k-1}\} = \{N_1, \ldots, N_{k-1}\}$ and hence $g^N(B) = f(R) = \Phi(I, f, w, B)$. If on the other hand there exists a swap item S_j with $j > k$, then, $f(S_j) > f(\{S_1, \ldots, S_{j-1}\}) \geq f(\{S_1, \ldots, S_{k-1}\}) = f(R)$ and as before follows $g^N(B) \geq f(N_1) \geq f(S_j) \geq f(R) = \Phi(I, f, w, B)$. □

It follows directly by Theorem 3 and Proposition 1 that the universal policy returned by Algorithm 3 has a robustness factor of at least ω.

3.2.1. Simplifying Algorithm 3.

Recall that the universal policy computed by Algorithm 3 is better or equal than Algorithm 1, in contrast to the universal policy computed by the *modified* Algorithm 2. However, in contrast to the *modified* Algorithm 2, it might be necessary to swap arbitrary many items in Line 6 of Algorithm 3. Now, we simplify Algorithm 3 to Algorithm 4, described below, by starting with the steepest ascent order and swapping *only* the swap item with maximum objective value. Clearly, according to the notion of swap items, the maximum-value swap item is precisely the swap item located at the position with the highest index in the steepest ascent order among all swap items.

Algorithm 4:

Input: An instance (I, f, w) of SMUC.
Output: Universal policy N.

1 Determine the steepest ascent order S
2 $N \leftarrow S$
3 $j \leftarrow n$
4 **while** N_j *is not a swap item and* $j \neq 1$ **do**
5 $\quad j \leftarrow j - 1$
6 **if** $j \neq 1$ **then**
7 $\quad N \leftarrow (N_j, N_1, \ldots, N_{j-1}, N_{j+1}, \ldots, N_n)$
8 **return** N

For any instance of SMUC, Algorithm 4 outputs a universal policy better or equal than Algorithm 1.

Theorem 4. *Let (I, f, w) an instance of* SMUC *and $B \geq \max_{i \in I} w(i)$. Then, $g^N(B) \geq \Phi(I, f, w, B)$ for the universal policy N returned by Algorithm 4.*

The proof of Theorem 4 is deferred to the Appendix.

4. Maximizing a Submodular Function under an Unknown Knapsack Capacity without Assuming a Reasonable Knapsack Capacity.

This section considers SMUC *without* assuming a reasonable knapsack capacity *non-adaptively* and *adaptively*.

4.1. Nonadaptively Maximizing a Submodular Function under an Unknown Knapsack Capacity.

Recall that, by Example 1, the assumption of a reasonable knapsack capacity is necessary for guaranteeing a constant robustness factor by any universal policy if the knapsack is packed *without discarding*. Therefore, the *nonadaptive* approach to SMUC with an *arbitrary* unknown knapsack capacity might yield an arbitrarily bad solution if packing the knapsack is stopped as soon as the first item does not fit into the knapsack. In order to circumvent the inapproximability in this setting, we permit discarding during the packing.

Definition 9. *Let (I, f, w) an instance of* SMUC *and N a universal policy of I. The* ***exhausted packed set*** *$E(N, B)$ packed with discarding according to the order N, given a knapsack capacity B, is the unique set $U \subseteq I$ with*

(i) $\sum_{i \in U} w(i) \leq B$,

(ii) $\sum_{i \in U \cap \{N_k \colon k<j\}} w(i) + w(N_j) > B$ *for all* $N_j \in I \setminus U$.

The ***exhausting value function*** *of N assigns to any knapsack capacity B the function value of the corresponding exhausted packed set $E(N, B)$ and is denoted by*

$$h^N \colon \mathbb{R}_+ \to \mathbb{R}_+,\ h^N(B) \coloneqq f(E(N, B)).$$

A universal policy is called ***better or equal*** *than Algorithm 1 if the exhausted value function has, for any capacity B, a value $h^N(B) \geq \Phi(I, f, w, B)$.*

Our goal would be to construct for each instance of SMUC a universal policy better or equal than Algorithm 1. However, the following example shows that this is impossible.

Example 3. *Let $I = \{a, b, c\}$ a set of items with weights $w(a) = 2.9$, $w(b) = 2$ and $w(c) = 1$. Let $v(a) = 3$, $v(b) = 2$, and $v(c) = 1$. Let $f : 2^I \to \mathbb{R}_{\geq 0}$ be defined by $f(T) = \sum_{t \in T} v(t) - v(b)$ for all $T \in 2^I$ with $\{a, b\} \subseteq T$ and $f(T) = \sum_{t \in T} v(t)$ for all $T \in 2^I$ with $\{a, b\} \nsubseteq T$.*

Let $B = 2$. Then, Algorithm 1 first removes item a, because $w(a) > 2$, and then returns item b because $w(\{b, c\}) = 3 > 2$ and $f(b) > f(c)$. Therefore, in every universal policy that imitates Algorithm 1 item b must be ordered earlier than item c.

Let $\bar{B} = 5$. Then, Algorithm 1 outputs $\{a, c\}$ with $f(\{a, c\}) = 4$. We show that for any universal policy N in which item b appears earlier than item c, it necessarily holds $h^N(5) \leq 3$. To see this, observe that $w(\{a, b, c\}) > 5$ and therefore either $h^N(5) = f(\{a, b\}) = 3$ or $h^N(5) = f(\{b, c\}) = 3$. Then, it holds $\Phi(I, f, w, 5) = f(\{a, c\}) = 4 > h^N(5)$ for any universal policy in which b is ordered earlier than c.

Notice that the order in which Algorithm 1 considers the items depends on their relative increase and therefore depends on items already excluded at the beginning.

4.2. Adaptively Maximizing a Submodular Function under an Unknown Knapsack Capacity.

Example 3 shows an instance of SMUC and knapsack capacities that do not permit a nonadaptive approximation by ω. To avoid this impossibility, we now consider adaptive algorithms. In this setting, we present a new adaptive algorithm that performs for any knapsack capacity at least as well as Algorithm 1, and therefore at least as well as the algorithm proposed by Klimm and Knaack [9], with matching worst-case approximation guarantees. However, our adaptive algorithm is far more intuitive and allows for a far simpler proof, following almost directly by Algorithm 4. Furthermore, there exist instances where our algorithm strictly outperforms the algorithm of Klimm and Knaack [9].

Theorem 5. *There exists an adaptive algorithm that, for any instance of* SMUC *and any fixed but unknown knapsack capacity, performs at least as well as Algorithm 1 for the same knapsack capacity, known in advance.*

To prove Theorem 5, we require a simple observation directly following the notion of the universal policy returned by Algorithm 4.

Lemma 2. *Let (I, f, w) an instance of* SMUC*, S the steepest ascent order, N the universal policy returned by Algorithm 4 applied to (I, f, w) and $N_1 = S_j$. Then, it holds $w(S_i) < w(N_1)$ for $1 \leq i < j$.*

Now, we prove Theorem 5.

Proof of Theorem 5. Let (I, f, w) be an instance of SMUC. For any fixed knapsack capacity B let $I^B \coloneqq \{i \in I \colon w(i) \leq B\}$ and let S^B denote the steepest ascent order of $(I^B, f|_{I^B}, w|_{I^B})$ and N^B the output of Algorithm 4 applied to $(I^B, f|_{I^B}, w|_{I^B})$. We gradually develop an adaptive algorithm that satisfies the claim. To this end, define $A \coloneqq \{i \in I \colon$ there exists B such that $i = N_1^B\}$ and let the adaptive algorithm try to add the items of A in non-increasing weight order into the knapsack, until the first item fits into the knapsack.

Let B be an arbitrary but fixed knapsack capacity. It follows directly by the notion of A that the first item packed into the knapsack by the adaptive algorithm is the item N_1^B. Furthermore, let $x \in A \cup \{\emptyset\}$ represent the item that was tried to pack into the knapsack directly before N_1^B and $I^{<w(x)} \coloneqq \{i \in I \colon w(i) < w(x)\}$, with $I^{<w(\emptyset)} \coloneqq I$. Then, for $S^{<w(x)}$ being the steepest ascent order of $(I^{<w(x)}, f|_{I^{<w(x)}}, w|_{I^{<w(x)}})$ it holds $S_i^B = S_i^{<w(x)}$ for $1 \leq i \leq j$ with $S_j^B = N_1^B$, by Lemma 2, implying $\{S_1^{<w(x)} \ldots, S_j^{<w(x)}\} = \{N_1^B, \ldots, N_j^B\}$. Thus, let the adaptive algorithm continuing packing the items $S_1^{<w(x)}, \ldots, S_{j-1}^{<w(x)}$. If it holds $B \geq w(\{N_1^B, \ldots, N_j^B\})$, let the adaptive algorithm continue packing the item that maximizes the relative increase, while any item that does not fit into the knapsack gets discarded, and packing continues until every item either gets packed or discarded. Then, this adaptive algorithm exactly packs according to N^B; thus, it follows directly by Theorem 4 that it packs a value of at least $\Phi(I, f, w, B)$. □

Due to the extensive nature of the adaptive algorithm in Klimm and Knaack [9], we refrain from presenting it here but only provide a simple example to illustrate the superiority of our algorithm. For more in-depth information about their algorithm, we refer to Klimm and Knaack [9].

Example 4. *Let* $(\{a, b, c, d\}, f, w)$ *be an instance of* SMUC *with* $f(T) \coloneqq |T|$ *for all* $T \subseteq \{a, b, c\}$ *and* $f(T) = 5$ *for all* $T \supseteq \{d\}$*, and* $w(a) = w(b) = w(c) = 1$*, and* $w(d) = 6$*.*

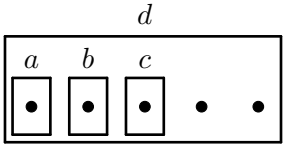

Figure 1: Graphical representation of the function f

As shown in Figure 1, the function f assigns to each set of items the number of dots covered by the corresponding rectangles and, therefore, is a maximum coverage function.

For the knapsack capacity $B = 6$, the adaptive algorithm described in the proof of Theorem 5 as well as Algorithm 1 pack the set $\{d\}$, for a value of 5. However, in contrast, for this particular instance, the adaptive algorithm of Klimm and Knaack [9] packs according to steepest ascent, thus, the set $\{a, b, c\}$ for a value of 3.

5. Nonadaptively Maximizing a Modular Function Under an Arbitrary Unknown Knapsack Capacity.

In this section, we revise MMUC with an *arbitrary* unknown knapsack capacity non-adaptively with discarding. We present an algorithm that computes a universal policy for every instance of MMUC better or equal to Algorithm 1. It is well-known that for every instance of MMUC Algorithm 1 approximates the optimal solution by a factor of at least $\frac{1}{2}$.

Lemma 3. *[10, Proposition 17.6.] Let (I, f, w) an instance of* MMUC*, then it holds, for any $B \geq 0$, that $\Phi(I, f, w, B) \geq \frac{1}{2} \cdot f(\mathrm{Opt}^B)$.*

It follows by Lemma 3 that a universal policy better or equal than Algorithm 1 has a robustness factor of at least $\frac{1}{2}$. Thus, in this setting, we match the result of Disser et al. [4]. However, we will present an algorithm that is more intuitive than theirs and allows for an uncomplicated and straightforward proof of correctness.

Algorithm 4 is not suitable for generating a universal policy better or equal than Algorithm 1 because if Algorithm 1 outputs for some fixed knapsack capacity a single item and the first item in the universal policy generated by Algorithm 4 is a swap item heavier than the fixed knapsack capacity, the exhausting value function may yield a smaller value than the function value of the single item. Another obvious idea to obtain for any instance of MMUC a universal policy better or equal than Algorithm 1 would be to apply Algorithm 3. However the following example illustrates that this is futile.

Example 5. *Let $I = \{a, b, c\}$ be a set of items with weights $w(a) = 5$, $w(b) = 1$, and $w(c) = 7$, and let $f \colon 2^I \to \mathbb{R}_{\geq 0}$ be a modular function defined by $f(\emptyset) = 0$, $f(a) = 5$, $f(b) = 2$, and $f(c) = 8$. The steepest ascent order of I is $S = (b, c, a)$. Since $f(c) = 8 > 2 = f(b)$, item c is a swap item and since $f(a) = 5 < 10 = f(c, b)$, item a is not a swap item, and therefore Algorithm 3 returns the universal policy $N = (c, b, a)$.*

Let $B = 5$. It holds $h^N(5) = f(b) = 2 < 5 = f(a) = \Phi(I, f, w, 5)$.

Example 5 demonstrates that there exist instances of MMUC such that the universal policy returned by Algorithm 3 is not better or equal than Algorithm 1. Therefore, moving only the swap items to the front may not generally be sufficient to obtain a universal policy better or equal to Algorithm 1. We generalize the notion of swap items.

Definition 10. *Let (I, f, w) be an instance of* SMUC *and $N = (i_1, \ldots, i_{|I|})$ an arbitrary order of I. For $2 \leq j \leq |I|$ we call i_j a **generalized swap item** in N, if there exist $k \in \{1, \ldots, j-1\}$ with $f(i_j) > f(\{i_k, \ldots, i_{j-1}\})$.*

Now, we modify Algorithm 3 by starting with the steepest ascent order S and moving any generalized swap item N_j to the position $k < j$, where $\{N_k, \ldots, N_{j-1}\}$ is the largest set directly in front of N_j with a function value smaller than the function value of N_j.

Algorithm 5:

Input: An instance (I, f, w) of MMUC.

Output: Universal policy N.

1 Determine the steepest ascent order S

2 $N \leftarrow S$

3 **for** $j = 2, \ldots, n$ **do**

4 **if** N_j *is a generalized swap item in* N **then**

5 $k \leftarrow \min\{l \colon l \in \{1, \ldots, j-1\}, f(\{N_l, \ldots, N_{j-1}\}) < f(N_j)\}$

6 $N \leftarrow (N_1, \ldots, N_{k-1}, N_j, N_k, \ldots, N_{j-1}, N_{j+1}, \ldots, N_n)$

7 **return** N

We demonstrate that for any instance of MMUC Algorithm 5 computes a universal policy better or equal than Algorithm 1.

Theorem 6. *Let (I, f, w) an instance of* MMUC *and N the universal policy returned by Algorithm 5, then $h^N(B) \geq \Phi(I, f, w, B)$ for any knapsack capacity B.*

In order to prove Theorem 6, we need a simple observation regarding the restriction of universal policies to the items smaller than some fixed knapsack capacity.

Definition 11. *Let I a set of items, N a universal policy of I, and $T \subseteq I$. The restriction of N to T is the universal policy $N|_T = (N_{k_1}, \ldots, N_{k_m})$ with $m \leq n$, and $1 \leq k_i < k_j \leq n$ for all $1 \leq i < j \leq m$, and $N_{k_i} \in T$ for all $1 \leq i \leq m$.*

The restriction of a universal policy, obtained by Algorithm 5 for some instance of MMUC, to the set of items with weight smaller than a fixed knapsack capacity equals the universal policy obtained by Algorithm 5 for the same instance but with all items heavier than the knapsack capacity removed before applying Algorithm 5.

Lemma 4. *Let (I, f, w) an instance of* MMUC *and N the universal policy returned by Algorithm 5 applied to the instance (I, f, w). Let $B \geq 0$ a knapsack capacity and N^B the universal policy returned by Algorithm 5 applied to the instance $(\{i \in I \colon w(i) \leq B\}, f|_{\{i \in I \colon w(i) \leq B\}}, w|_{\{i \in I \colon w(i) \leq B\}})$. Then, $N|_{\{i \in I \colon w(i) \leq B\}} = N^B$.*

The proof of Lemma 4 is deferred to the Appendix.

Now, we prove Theorem 6.

Proof of Theorem 6. Let N the universal policy returned by Algorithm 5 for the instance (I, f, w). We distinguish two cases.

Case 1: Assume $B \geq \max_{i \in I} w(i)$. Let S the steepest ascent order in Line 1 of Algorithm 5 and assume without loss of generality that Algorithm 1 inspects the items according to S. If Algorithm 1 outputs the single item S_j with $j > 1$, then S_j is a

swap item, thus a generalized swap item. Therefore, it follows directly by Lemma 1 that $h^N(B) \geq f(N_1) \geq f(S_j) = \Phi(I, f, w, B)$.

Now, assume that Algorithm 1 outputs the packed set $R = \{S_1, \ldots, S_{j-1}\}$ with $2 \leq j \leq n$. Observe that for any order $\bar{N}$ and any generalized swap item $\bar{N}_m$ with $f(\bar{N}_m) > f(\{\bar{N}_k, \ldots, \bar{N}_{m-1}\})$ and $k < m$ it holds $f(\{\bar{N}_1, \ldots, \bar{N}_{k-1}\} \cup \{\bar{N}_m\}) = f(\{\bar{N}_1, \ldots, \bar{N}_{k-1}\} + f(\bar{N}_m) > f(\{\bar{N}_1, \ldots, \bar{N}_{k-1}\} + (\{\bar{N}_k, \ldots, \bar{N}_{m-1}\}) = f(\{\bar{N}_1, \ldots, \bar{N}_{m-1}\})$. However, this directly implies $h^N(B) \geq h^S(B) \geq f(R) = \Phi(I, f, w, B)$.

Case 2: Assume $B < \max_{i \in I} w(i)$. Let N^B the universal policy returned by Algorithm 5 for the instance $(\{i \in I \colon w(i) \leq B\}, f|_{\{i \in I \colon w(i) \leq B\}}, w|_{\{i \in I \colon w(i) \leq B\}})$. It holds $N|_{\{i \in I \colon w(i) \leq B\}} = N^B$ by Lemma 4. Therefore, $h^{N|_{\{i \in I \colon w(i) \leq B\}}}(B) = h^{N^B}(B) \geq \Phi(I, f, w, B)$, in which the last inequality follows by Case 1. □

6. Conclusion.

In this paper, we considered the problem of maximizing a monotone-increasing submodular function under an unknown knapsack capacity nonadaptively and adaptively. We presented an algorithm that for any instance of SMUC returns a universal policy better or equal than Algorithm 1 (due to Wolsey [14]) for any reasonable knapsack capacity. Thus, we demonstrated that the optimal solution of maximizing a monotone-increasing submodular function under an unknown *reasonable* knapsack capacity can be approximated nonadaptively and without discarding items by a least 0.357.

Omitting the reasonable knapsack capacity assumption, we presented an adaptive algorithm that performs, for any capacity, at least as well as Algorithm 1; thus, it has been demonstrated that the optimal solution of maximizing a monotone-increasing submodular function under an unknown arbitrary knapsack capacity can be approximated adaptively by at least 0.357.

For the special case that the submodular function is modular, an algorithm that generates a universal policy for every instance of SMUC better or equal to Algorithm 1 has been presented. Therefore, the optimal solution for maximizing a modular function under an unknown arbitrary knapsack capacity can be approximated adaptively by at least 0.5.

It remains an interesting open question whether there is a constant factor approximation for arbitrary monotone-increasing, submodular functions under an unknown arbitrary knapsack capacity. As a first step, it seems worthwhile to investigate special subsets of submodular functions, e.g., weighted matroid or coverage functions.

A. Appendix.

A.1. Proof of Theorem 2.

Proof. Let N the universal policy returned by the *modified* Algorithm 2, S the steepest ascent order in the *modified* Algorithm 2 and $B \geq \max_{i \in I} w(i)$. Assume without loss of generality that Algorithm 1 inspects the items in Line 4 according to the steepest ascent order S. Let $R^B = \{S_1, \ldots, S_{k-1}\}$ with $2 \leq k \leq n$ be the set constructed in the while loop in Line 3 of Algorithm 1.

Case 1: Algorithm 1 outputs the set R^B. Then, $g^N(B) = f(\{S_1, \ldots, S_{k-1}\}) = f(R^B) \geq \omega f(\mathrm{Opt}^B)$, where the last inequality follows by Proposition 1.

Case 2: Algorithm 1 outputs the single item S_k with $k > 1$ and $w(R^B) \leq \max_{i \in I} w(i)$. When the while loop in Line 3 of Algorithm 1 ends, it holds $\sum_{j=1}^{k} w(S_j) > B \geq \sum_{j=1}^{k-1} w(S_j) = w(R^B)$. Combined with $w(R^B) \leq \max_{i \in I} w(i)$ follows that the while loop of the *modified* Algorithm 2 ends with the item S_k. Since Algorithm 1 outputs S_k, it holds $f(S_k) > f(\{S_1, \ldots, S_{k-1}\})$ and the *modified* Algorithm 2 returns the universal policy $N = (S_k, S_1, \ldots, S_{k-1}, S_{k+1}, \ldots, S_{|I|})$. Because $w(S_k) \leq B$, it follows directly $g^N(B) \geq f(S_k) \geq \omega f(\mathrm{Opt}^B)$.

Case 3: Algorithm 1 outputs the single item S_k with $k > 1$ and $w(R^B) > \max_{i \in I} w(i)$. Then, the while loop of the *modified* Algorithm 2 breaks with some item S_m with $m < k$. Whether it holds $f(S_m) > f(\{S_1, \ldots, S_{m-1}\})$ or not, the *modified* Algorithm 2 either outputs $N = (S_m, S_1, \ldots, S_{m-1}, S_{m+1}, \ldots, S_{k-1}, S_k, \ldots, S_{|I|})$ or $N = S$. In either case, it holds $g^N(B) = f(\{S_m, S_1, \ldots, S_{m-1}, S_{m+1}, \ldots, S_{k-1}\}) = f(R^B)$ and $w(S_k) \leq \max_{i \in I} w(i) < w(R^B)$, which implies $2 \cdot w(R^B) > w(R^B) + w(S_k) > B$.

Applying Algorithm 1 to the instance (I, f, w) and the capacity $D = w(R^B)$, Algorithm 1 constructs the packed set R^B and it holds $w(R^B) = D$. It follows by Proposition 2 that

$$\begin{aligned} g^N(B) = f\left(R^B\right) &\geq \left(1 - e^{-\frac{D}{B}}\right) f\left(\mathrm{Opt}^B\right) \\ &\geq \left(1 - e^{-\frac{w\left(R^B\right)}{2w(R^B)}}\right) f\left(\mathrm{Opt}^B\right) = \left(1 - e^{-\frac{1}{2}}\right) f(\mathrm{Opt}^B) \geq \omega f\left(\mathrm{Opt}^B\right). \square \end{aligned}$$

A.2. Proof of Theorem 4

Proof. Let N be the universal policy returned by Algorithm 4, S the steepest ascent order in Line 1 of Algorithm 4, and $B \geq \max_{i \in I} w(i)$. Assume without loss of generality that Algorithm 1 inspects the items in Line 4 according to S. To prove the claim, we compare $g^N(B)$ with $\Phi(I, f, w, B)$. If Algorithm 1 outputs in Line 9 the single item i^* with $i^* = S_k$, $k > 1$, then it follows $g^N(B) \geq f(N_1) \geq f(S_k) = \Phi(I, f, w, B)$ completely analogous to Case 1 in the proof of Theorem 3.

Thus, assume that Algorithm 1 outputs $R = \{S_1, \dots, S_{k-1}\}$ with $2 \leq k \leq n$. Clearly, S_k is not a swap item, since otherwise Algorithm 1 would have output S_k, and therefore holds $N_1 \neq S_k$. Assume that $N_1 = S_1$. This implies the absence of swap items and therefore $N = S$ and $g^N(B) = f(R) = \Phi(I, f, w, B)$.

Now, assume that $N_1 = S_j$ with $j > k$. Then, S_j is a swap item and by the notion of swap items $f(S_j) > f(\{S_1, \dots, S_{j-1}\}) \geq f(R)$. It follows immediately that $g^N(B) \geq f(N_1) \geq f(R) = \Phi(I, f, w, B)$.

Last, assume that $N_1 = S_j$ with $j < k$. Then, Algorithm 4 outputs the order $N = (S_j, S_1, \dots, S_{j-1}, S_{j+1}, \dots, S_{k-1}, S_k, \dots, S_{|I|})$. Thus, $\{S_1, \dots, S_k\} = \{N_1, \dots, N_k\}$ and $g^N(B) = f(R) = \Phi(I, f, w, B)$. □

A.3. Proof of Lemma 4.

Proof. Let S the steepest ascent order of I and $S_l \in \arg\max_{i \in I} w(i)$, and $\bar{S}$ the steepest ascent order of $I \setminus S_l$. Notice that $S_{r+1} = \bar{S}_r$ for $r \geq l$. Furthermore, let Z^k denote the order in Line 6 in the k-th iteration of the for loop in Line 3 of Algorithm 5 applied to (I, f, w) and $\bar{Z}^k$ the order in Line 6 in the k-th iteration of the for loop in Line 3 of Algorithm 5 applied to $(I \setminus S_j, f|_{I \setminus S_j}, w|_{I \setminus S_j})$.

Notice that for any $j < l$ it holds $Z^j_k = \bar{Z}^j_k$ for $k < l$ and $Z^j_{k+1} = \bar{Z}^j_k$ for $k \geq l$. Moreover, $f(S_{l+1}) > f(\{Z^l_k, \dots, Z^l_l\})$ for $k \leq l$ is true if and only if $f(\bar{S}_l) > f(\{\bar{Z}^{l-1}_{k-1}, \dots, \bar{Z}^{l-1}_{l-1}\})$. It holds $f(S_l) > f(S_k)$ for any $k > l$, because of the modularity of f and $w(S_k) \leq \max_{i \in I} w(i)$. Let $Z^l_t = S_l$. Then, in the above equivalence, $k > t$ must hold. For any $m \geq l$, it follows directly that $f(S_{m+1}) > f(\{Z^m_k, \dots, Z^m_m\})$ for $t < k \leq m$ if and only if $f(\bar{S}_m) > f(\{\bar{Z}^{m-1}_{k-1}, \dots, \bar{Z}^{m-1}_{m-1}\})$.

Therefore, for any $j \geq l$ it holds $Z^j_{k+1} = \bar{Z}^j_k$ for $k \geq t$ and $Z^j_k = \bar{Z}^j_k$ for $k < t$. However, this directly implies that the order Z^n returned by Algorithm 5 applied to (I, f, w) with S_j removed equals the order $\bar{Z}^n$ returned by Algorithm 5 applied to $(I \setminus S_j, f|_{I \setminus S_j}, w|_{I \setminus S_j})$. Now, the claim follows by induction. □

References

[1] Alexander A Ageev and Maxim I Sviridenko. An 0.828-approximation algorithm for the uncapacitated facility location problem. *Discrete Applied Mathematics*, 93 (2-3):149–156, 1999.

[2] Aaron Bernstein, Yann Disser, Martin Groß, and Sandra Himburg. General bounds for incremental maximization. *Mathematical Programming*, 191(2):953–979, 2022.

[3] Gerard Cornuejols, Marshall L Fisher, and George L Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management science*, 23(8):789–810, 1977.

[4] Yann Disser, Max Klimm, Nicole Megow, and Sebastian Stiller. Packing a knapsack of unknown capacity. *SIAM Journal on Discrete Mathematics*, 31(3):1477–1497, 2017.

[5] Yann Disser, Max Klimm, Annette Lutz, and David Weckbecker. Fractionally subadditive maximization under an incremental knapsack constraint with applications to incremental flows. *SIAM Journal on Discrete Mathematics*, 38(1):764–789, 2024.

[6] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[7] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[8] Yasushi Kawase, Hanna Sumita, and Takuro Fukunaga. Submodular maximization with uncertain knapsack capacity. *SIAM Journal on Discrete Mathematics*, 33(3): 1121–1145, 2019.

[9] Max Klimm and Martin Knaack. Maximizing a submodular function with bounded curvature under an unknown knapsack constraint. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[10] Bernhard Korte and Jens Vygen. *Combinatorial optimization: theory and algorithms*. Springer, 5 edition, 2012.

[11] Alfredo Navarra and Cristina M. Pinotti. Online knapsack of unknown capacity: How to optimize energy consumption in smartphones. *Theoretical Computer Science*, 697:98–109, 2017.

[12] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14:265–294, 1978.

[13] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

[14] Laurence A. Wolsey. Maximising real-valued submodular functions: Primal and dual heuristics for location problems. *Mathematics of Operations Research*, 7(3): 410–425, 1982.