On Packing a Submodular Knapsack of Unknown Capacity

Sabine Münch, Stephen Raach, Sven de Vries

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

Consider the problem of maximizing a monotone-increasing submodular function defined on a set of weighted items under an unknown knapsack capacity. Assume that items are packed sequentially into the knapsack and that the capacity of the knapsack 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 of an item exceeds the knapsack capacity, there are two possible approaches: in packing without discarding, packing stops directly, while in packing with discarding, the item is removed from consideration, and packing continues.

Our main result considers *non-adaptive* packing without discarding according to a *predetermined* sequence, called universal policy, and presents the first algorithm to compute universal policies that perform for any unknown but fixed knapsack capacity (which is assumed to be greater than or equal to the heaviest item), at least as good as the classic greedy algorithm due to Wolsey (1982) applied to the same known capacity.

Besides, we obtain results for packing with discarding. Specifically, we develop an adaptive algorithm for maximizing a monotone increasing submodular function, and a universal policy for maximizing a modular function under an unknown, arbitrary knapsack capacity. Both are much simpler than those presented in previous literature, while providing the same approximation guarantees.

Keywords : *Combinatorial optimization, submodular maximization, knapsack, unknown capacity, non-adaptive.*

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., Goemans and Williamson 1995), set cover problems (see, e.g., Feige 1998), and facility location problems (see, e.g., Cornuejols et al. 1977; Ageev and Sviridenko 1999).

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

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

$$\max\{f(X)\colon w(X) \leq B,\, X \subseteq I\}. \tag{1}$$

Notice that, by the assumption $f(i) > 0$ for all $i \in I$, this maximum is automatically greater than zero for all B with $B \geq \min_{i \in I} w_i$.

Even though this problem is known to be NP-hard, a simple greedy algorithm, analyzed by Wolsey (1982), achieves a constant approximation factor, where the approximation factor is the worst-case ratio between the objective value of the algorithm's return and that of an optimal solution. This greedy algorithm starts with an empty solution and iteratively constructs a feasible solution by selecting items with maximum relative increase for addition to the current solution, until the addition of the next selected item would cause the total weight of the current solution to exceed the knapsack capacity. Then the greedy algorithm compares the current solution with the last selected item and returns the one with the greater objective value.

Definition 2. *Let $\beta = 0.4428\ldots$ denote the unique real solution of $e^x = 2 - x$, and denote $\omega \coloneqq 1 - e^{-\beta}$.*

Wolsey (1982) demonstrated that this greedy algorithm has an approximation factor of at least ω. Furthermore, Sviridenko (2004) showed that combining this greedy-algorithm with partial enumeration achieves an approximation factor of $1 - \frac{1}{e} = 0.63\ldots$. This approximation factor is the best possible achievable in polynomial time, unless $P = NP$, as demonstrated by Feige (1998).

Both approximation results, Wolsey (1982) and Sviridenko (2004), 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), where I is a set of items with weights $w_i \geq 0$ for $i \in I$, and f is a monotone-increasing submodular set function defined on I. By definition, any instance of SMUC is independent of a specific knapsack capacity B.

We focus on knapsack capacities that are large enough to allow any single item to fit.

Definition 3. *Given an* SMUC*-instance* (I, f, w)*, any knapsack capacity* $B \geq \max_{i \in I} w_i$ *is called* ***reasonable*** *for* (I, f, w)*.*

There are two possible approaches to SMUC: maximizing the submodular function either *non-adaptively* or *adaptively*.

In the non-adaptive approach, an instance of SMUC is given, and the items have to be pre-ordered. After ordering the items, the knapsack's capacity is revealed, and the items are added one by one according to the order.

Definition 4. *Given* I*, we denote the set of all permutations of* I *by* $\Pi(I)$ *and we call any permutation* $N = (N_1, N_2, \ldots, N_n) \in \Pi(I)$ *a* ***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 (and thereby discarding may implicitly happen).

Non-adaptive 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.

1.1. Results.

Our main contribution is for *non-adaptive* packing without discarding. We provide the first algorithm that computes for any instance of SMUC a universal policy such that packing without discarding according to that policy yields, for any fixed *reasonable* knapsack capacity, a set of items that is at least as valuable as the output of the greedy algorithm studied by Wolsey (1982), for the same (known) knapsack capacity.

Therefore, in terms of the aforementioned leader-follower problem, the present paper finds an approximately optimal solution, i.e., an approximately optimal policy, and demonstrates that packing according to it approximates the optimal packing for any knapsack capacity by a factor of at least ω.

As a consequence of our result, we obtain an adaptive algorithm (adaptivity naturally allows for discarding) for SMUC that packs for any *arbitrary* (even unreasonable) unknown knapsack capacity a set whose value approximates the value of an optimal solution by at least ω. Thus, we provide a simpler adaptive algorithm than those presented in the literature while matching the best-known approximation guarantee. Furthermore, we present a curvature-dependent bound of the presented adaptive algorithm using a curvature-dependent bound for a greedy algorithm by Klimm and Knaack (2022).

As another implication of our main result, we show that for non-adaptive maximization of a *modular function* with discarding, one can pack an knapsack of *arbitrary* unknown but fixed capacity at least as well as the greedy algorithm analyzed by Wolsey (1982) does for the same known knapsack capacity. Thereby, we give a simpler proof of the best possible $\frac{1}{2}$-approximation by Disser et al. (2017). For a summary of our results, see Table 1 below.

1.2. Further related work.

Submodular maximization with various additional constraints is a central topic in combinatorial optimization. Nemhauser et al. (1978) 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. (1978) showed that a greedy algorithm achieves an approximation factor of $1 - \frac{1}{e} = 0.63\ldots$. Furthermore, Feige (1998) 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 (1982) introduced a variation of the greedy algorithm, based on selecting items according to their relative increase, with an approximation factor of ω. Later, Sviridenko (2004) demonstrated that combining a partial enumeration procedure with this greedy algorithm achieves the best possible approximation factor of $1 - \frac{1}{e}$.

Navarra and Pinotti (2017) and Disser et al. (2017) considered maximization of a modular function under an unknown knapsack capacity (MMUC) *non-adaptively*. Each

of them proposed an algorithm that computes a universal policy for any instance of MMUC. Assuming a reasonable knapsack capacity, the knapsack packed without discarding according to the universal policy, returned by the algorithm of Navarra and Pinotti (2017), approximates the value of an optimal solution by a factor of at least $\frac{1}{2}$. For arbitrary knapsack capacities, packing with discarding according to the universal policy returned by the algorithm of Disser et al. (2017) achieves the same approximation guarantee.

For the problem of maximizing a submodular function under an unknown arbitrary knapsack capacity, Kawase et al. (2019) presented a *randomized* universal policy such that the expected value of the knapsack packed non-adaptively with discarding, according to that policy, approximates the value of an optimal solution by at least $\frac{1}{2}(1-\frac{1}{\sqrt[4]{e}}) = 0.11\ldots$.

Disser et al. (2024) considered the more general problem of maximizing a fractionally subadditive function under an unknown knapsack capacity by non-adaptively packing without discarding. However, their policy fails to achieve a constant factor approximation; instead, it relies on the ratio between the value of the least and the most valuable item. Non-adaptive maximization of a strict superset of submodular functions under an unknown cardinality constraint without discarding items is considered by Bernstein et al. (2022).

Kawase et al. (2019) considered *adaptively* maximizing a monotone submodular function under an unknown knapsack capacity. They demonstrated that an adaptive packing algorithm approximates the value of an optimal solution by a factor of at least $\frac{2(1-1/e)}{21} = 0.06\ldots$. Klimm and Knaack (2022) improved this factor to ω.

		non-adaptive packing	adaptive packing
MMUC	reasonable knapsack without discarding	Navarra and Pinotti (2017): $\frac{1}{2}$	$\left(\frac{1}{2}\right)$
	arbitrary knapsack with discarding	Disser et al. (2017): $\frac{1}{2}$ This work (Thm. 6): $\frac{1}{2}$	$\left(\frac{1}{2}\right)$
SMUC	reasonable knapsack without discarding	This work (Thm. 4, 3): ω	(ω)
	arbitrary knapsack	unknown	Klimm and Knaack (2022): ω This work (Cor. 3): ω

Table 1: Summary of approximation guarantees for MMUC and SMUC achieved by this and previous work. (Notice that all approximation results for non-adaptive packing imply the same approximation results for adaptive packing.)

1.3. Outline.

In Section 2, we review the greedy algorithm analyzed by Wolsey (1982) for SMKC. Next, in Section 3, we present our main result: an algorithm that returns for any SMUC-instance a universal policy such that, for any reasonable knapsack capacity, non-adaptive packing according to that policy yields a set at least as valuable as the solution returned by the greedy algorithm applied to the same instance and known knapsack capacity.

Section 4 presents an easy consequence of the main result for adaptive packing for SMUC, *without* the reasonable knapsack capacity assumption. In Section 5, we address non-adaptive packing with discarding for the special case of maximizing a modular function under an unknown arbitrary knapsack capacity. In the Appendix, we give the omitted proofs.

2. Maximization of a submodular function with a known knapsack constraint.

We start by recapitulating the greedy algorithm analyzed by Wolsey (1982) for SMKC. To convert this greedy algorithm into a deterministic algorithm, we usually *fix an arbitrary tie-breaking rule for comparing items by their relative increase* by choosing a permutation.

Algorithm 1:

Input: Set of items I, weights $w_i \geq 0$ for $i \in I$, submodular function f, knapsack capacity B.

Output: Approximately optimal solution to SMKC.

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 Choose $i^* \in \arg\max_{i \in I \setminus R} \left\{ \frac{f(R \cup \{i\}) - f(R)}{w_i} \right\}$ with tie-breaking according to a fixed rule
5 **if** $w(R \cup \{i^*\}) \leq B$ **then** $R \leftarrow R \cup \{i^*\}$
6 **else break**
7 **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 to SMKC. Then it initializes $R = \emptyset$ and chooses in each 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 $w(R \cup \{i^*\}) \leq B$, i.e., 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 is left. If all items fit into the knapsack, Algorithm 1 returns the entire set in Line 7. 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. *For an instance (I, f, w) of* SMUC *and a knapsack capacity B, denote the* ***objective function value of the output of Algorithm 1*** *as $\Phi(I, f, w, B)$. Further, we use Opt^B to denote an arbitrary* ***optimal solution****, i.e., a set $X \subseteq I$ with $w(X) \leq B$ and $f(X)$ attaining the value of (1), and $f(\mathrm{Opt}^B)$ to denote the* ***value of an optimal solution*** *to the submodular maximization problem with knapsack capacity B.*

*The **approximation factor** of Algorithm 1 is defined as*

$$\inf_{\substack{\text{SMUC-}instance\ (I,f,w),\\ B\colon B\geq \min_{i\in I} w_i}} \frac{\Phi(I,f,w,B)}{f(\mathrm{Opt}^B)}.$$

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 value over all instances of SMUC and all knapsack capacities, provided that the value of an optimal solution is greater than zero.

Proposition 1. *(Wolsey 1982, Theorem 3) Algorithm 1 has an approximation factor of at least ω.*

Wolsey (1982) 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 provide a better approximation than ω in general.*

Proof. For $n \in \mathbb{N},\ n \geq \frac{2}{\beta}$, let (I, f, w) be an instance of SMUC given by $I = \{a_0, a_1, a_2, \ldots, a_{\lceil \beta n\rceil -1}, a_{\lceil \beta n\rceil}\}$, $w_{a_0} = 1$, $w_{a_i} = \frac{1}{n}$ for all $i \in \{1, 2, \ldots, \lceil \beta n\rceil - 1\}$ and $w_{a_{\lceil \beta n\rceil}} = 1 - \beta + \frac{2}{n}$, and $f\colon 2^I \to \mathbb{R}_{\geq 0}$ defined by

$$f(\emptyset) \coloneqq 0,$$

$$f(a_i) \coloneqq \begin{cases} 1 & \text{if } i = 0, \\ \frac{1}{n} & \text{if } i \in \{1, \ldots, \lceil \beta n\rceil - 1\}, \\ \left(1 - \beta + \frac{2}{n}\right)\left(1 - \frac{1}{n}\right)^{\lceil \beta n\rceil - 1} & \text{if } i = \lceil \beta n\rceil, \end{cases}$$

$$f(A) \coloneqq \frac{1}{n}\sum_{i=1}^{|A|}\left(1 - \frac{1}{n}\right)^{i-1} \text{ for } A \subseteq \{a_1, \ldots, a_{\lceil \beta n\rceil -1}\},$$

$$f(A \cup \{a_{\lceil \beta n\rceil}\}) \coloneqq f(A) + f(a_{\lceil \beta n\rceil}) \text{ for } A \subseteq \{a_1, \ldots, a_{\lceil \beta n\rceil -1}\}, \text{ and}$$

$$f(X) \coloneqq 1 \text{ for } X \subseteq I,\ a_0 \in X.$$

It is straightforward to check that f is submodular and monotone increasing. Further, for $1 \leq j \leq \lceil \beta n\rceil$, we have

$$\begin{aligned} &\frac{f\left(\{a_1, \ldots, a_{j-1}\} \cup \{a_0\}\right) - f\left(\{a_1, \ldots, a_{j-1}\}\right)}{w_{a_0}} \\ &= 1 - \frac{1}{n}\sum_{i=1}^{j-1}\left(1 - \frac{1}{n}\right)^{i-1} = 1 - \frac{1}{n}\frac{1 - (1 - \frac{1}{n})^{j-1}}{1 - (1 - \frac{1}{n})} \\ &= \left(1 - \frac{1}{n}\right)^{j-1} = \frac{\frac{1}{n}\sum_{i=1}^{j}(1 - \frac{1}{n})^{i-1} - \frac{1}{n}\sum_{i=1}^{j-1}(1 - \frac{1}{n})^{i-1}}{\frac{1}{n}} \\ &= \frac{f\left(\{a_1, \ldots, a_j\}\right) - f\left(\{a_1, \ldots, a_{j-1}\}\right)}{w_{a_j}} \end{aligned}$$

$$\geq \frac{f\left(\{a_1,\dots,a_{\lceil\beta n\rceil}\}\right) - f\left(\{a_1,\dots,a_{\lceil\beta n\rceil-1}\}\right)}{w_{a_{\lceil\beta n\rceil}}},$$

where the second equality follows by the geometric sum, and the last inequality holds as an equality when $j = \lceil\beta n\rceil$. Assume that Algorithm 1 breaks ties in Line 4 by favoring a_i over a_{i+1} for $1 \leq i < \lceil\beta n\rceil - 1$, and any item $a_j \in \{a_1,\dots,a_{\lceil\beta n\rceil}\}$ over the item a_0.

Let $B = 1$ be the knapsack capacity. Then, Algorithm 1 constructs in the while loop in Line 3 the set $\bigcup_{i=1}^{\lceil\beta n\rceil-1}\{a_i\}$ and returns $\arg\max\{f\left(\bigcup_{i=1}^{\lceil\beta n\rceil-1}\{a_i\}\right), f(a_{\lceil\beta n\rceil})\}$, since $w(\{a_1,\dots,a_{\lceil\beta n\rceil-1}\}) = (\lceil\beta n\rceil - 1)\frac{1}{n} < \frac{\beta n}{n} = \beta < B$ and $w(\{a_1,\dots,a_{\lceil\beta n\rceil}\}) = (\lceil\beta n\rceil - 1)\frac{1}{n} + 1 - \beta + \frac{2}{n} \geq \beta - \frac{1}{n} + 1 - \beta + \frac{2}{n} > B$.

In the limit $n \to \infty$ we have

$$\begin{aligned}
\lim_{n\to\infty} f\left(\{a_1,\dots,a_{\lceil\beta n\rceil-1}\}\right) &= \lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{\lceil\beta n\rceil-1}\left(1-\frac{1}{n}\right)^{i-1} = \lim_{n\to\infty} 1-\left(1-\frac{1}{n}\right)^{\lceil\beta n\rceil-1}\\
&= \lim_{n\to\infty} 1-\left(\left(1-\frac{1}{n}\right)^{n}\right)^{\beta} = 1-e^{-\beta} = \omega = 1-e^{-\beta}\\
&\overset{\text{(I)}}{=} 1-\frac{1}{2-\beta} = \frac{1-\beta}{2-\beta} \overset{\text{(II)}}{=} (1-\beta)e^{-\beta} = \lim_{n\to\infty}(1-\beta)\left(1-\frac{1}{n}\right)^{\beta n}\\
&= \lim_{n\to\infty}\left(1-\beta+\frac{2}{n}\right)\left(1-\frac{1}{n}\right)^{\lceil\beta n\rceil-1} = \lim_{n\to\infty} f(a_{\lceil\beta n\rceil})
\end{aligned}$$

where (I) and (II) follow by Definition 2.

However, the optimal solution is $\{a_0\}$ with $f(a_0) = 1$ and the claim follows by $\lim_{n\to\infty}\frac{\Phi(I,f,w,B)}{f(a_0)} = \lim_{n\to\infty}\frac{\max\{f(\{a_1,\dots,a_{\lceil\beta n\rceil-1}\}),f(a_{\lceil\beta n\rceil})\}}{f(a_0)} = \omega$. □

It follows directly by Proposition 1 and Theorem 1:

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

3. Non-adaptive maximization of a submodular function under an unknown knapsack capacity.

In this section, our main result for the non-adaptive approach to SMUC, assuming reasonable knapsack capacities, is presented. 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, we define the* ***set packed without discarding*** *$K(N, B)$ by:*

$$K(N,B) \coloneqq \begin{cases} \left\{N_j \colon j \leq k,\ k = \max\left\{l \colon 1 \leq l \leq n,\ \sum_{i=1}^{l} w_{N_i} \leq B\right\}\right\} & \text{if } w_{N_1} \leq B,\\ \emptyset & \text{otherwise.}\end{cases}$$

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

$$g^N\colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0},\ g^N(B) \coloneqq f(K(N,B)).$$

*The **robustness factor** of a universal policy N for an instance (I,f,w) is defined as*

$$\inf_{B\colon\, B \geq \max_{i\in I} w_i} \frac{g^N(B)}{f(\mathrm{Opt}^B)}.$$

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

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

Notice that, in general, the assumption of a reasonable knapsack capacity is necessary for packing without discarding according to a universal policy to approximate the optimal solution value within a constant factor, as the following example shows.

Example 1. *(Navarra and Pinotti 2017) Let $I = \{a,b\}$, with $w_a = 1$ and $w_b = 1+\varepsilon$, $\varepsilon \geq 0$, and $f : 2^I \to \mathbb{R}_{\geq 0}$ be 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*, there exists no universal policy such that, for any arbitrary knapsack capacity, packing without discarding approximates an optimal solution by a constant factor as $M \to \infty$.*

If $N = (b,a)$, then for $B = 1$, we have $g^N(B) = 0$, since $w_b > 1$. However, the optimal solution to $\max\{f(X)\colon w(X) \leq 1,\ X \subseteq I\}$ is $\{a\}$, and therefore $\frac{g^N(B)}{f(\mathrm{Opt}^B)} = \frac{0}{1} = 0$.

If, on the other hand, $\bar{N} = (a,b)$, then for $\bar{B} = 1+\varepsilon$, we have $g^{\bar{N}}(\bar{B}) = 1$, since $w_a < 1+\varepsilon$ and $w_a + w_b > 1+\varepsilon$. However, the optimal solution to $\max\{f(X)\colon w(X) \leq 1+\varepsilon,\ X \subseteq I\}$ is $\{b\}$, and therefore $\frac{g^{\bar{N}}(\bar{B})}{f(\mathrm{Opt}^{\bar{B}})} = \frac{1}{M}$ and $\lim_{M\to\infty} \frac{g^{\bar{N}}(\bar{B})}{f(\mathrm{Opt}^{\bar{B}})} = 0$.

In contrast to the inapproximability shown in Example 1, we have, for the same instance, any *reasonable* knapsack capacity B, and the universal policy $N = (b,a)$, that

$$\min_{B\geq 1+\varepsilon} \frac{g^N(B)}{f(\mathrm{Opt}^B)} = 1,$$

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 (2017) presented an algorithm, described below, which returns a universal policy with a robustness factor of at least $\frac{1}{2}$.

Algorithm 2:

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

1 $N \leftarrow (N_1, N_2, \ldots, N_n)$ Items ordered non-increasingly by $\frac{f(i)}{w_i}$.
2 $k \leftarrow 1$
3 **while** $\sum_{j=1}^{k} w_{N_j} \leq \max_{i \in I} w_i$ **and** $k \leq n$ **do**
4 $\quad k \leftarrow k + 1$
5 **if** $f(N_k) > \sum_{j=1}^{k-1} f(N_j)$ **then** $N \leftarrow (N_k, N_1, N_2, \ldots, N_{k-1}, N_{k+1}, \ldots, N_n)$
6 **return** N

Before generalizing Algorithm 2 to arbitrary submodular objective functions, we briefly describe it. In Algorithm 2, the initial policy N is given by sorting the items according to their ratio $\frac{f(i)}{w_i}$. 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 greater than the sum of the values of the previous items $N_1, \ldots, N_{k-1}$, the policy N is updated by moving N_k to the front, and the updated policy is returned. Notice that the final policy is generated by updating the initial policy at most once.

Definition 7. *Let (I, f, w) be an instance of* SMUC. *We call $S = (S_1, \ldots, S_n) \in \Pi(I)$ the* ***greedy policy*** *if*

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

for all $1 \leq j \leq n$ and ties (for the choice of S_j) are resolved using the same tie-breaking rule as in Algorithm 1.

Notice that the greedy policy of an instance coincides with the order in which Algorithm 1 inspects the items, since we use the same tie-breaking rule.

Although Algorithm 2 is intended for MMUC, a slightly *modified* version can be applied to SMUC. We change Lines 1 and 5 in the following way:

1' $N \leftarrow (N_1, \ldots, N_n)$ Items in greedy policy
5' **if** $f(N_k) > f(\{N_1, \ldots, N_{k-1}\})$ **then**

For modular objective functions, the modifications of Algorithm 2 correspond to the original algorithm, and therefore neither the course nor the result of the computation is changed in this case.

Unfortunately, the universal policy returned by the *modified* Algorithm 2 does not always compare favorably to Algorithm 1, since there are instances where this policy is worse than Algorithm 1, as the following example demonstrates.

Example 2. *Let $I = \{a, b, c\}$ 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 greedy policy $N = (a, b, c)$. *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* $f(N_k) = f(b) = 0.6 < 1 = f(a)$ *the unchanged policy* 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 7, since* $f(c) = 2 > f(\{a, b\}) = 1.6$. *Therefore* $\Phi(I, f, w, 3) > g^N(3)$.

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

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

3.2. Matching the approximation factor of Algorithm 1 non-adaptively.

As a simple consequence of Example 2, any universal policy that wants to imitate Algorithm 1 has to treat any single item i^* that could be returned by Algorithm 1 with special care since it might be necessary to place i^* at the front of the universal policy.

Definition 8. *Let* (I, f, w) *be an instance of* SMUC *and let* $S = (S_1, \ldots, S_n)$ *be the greedy policy of* I. *For* $2 \leq j \leq n$, *call* S_j *a* ***swap item*** *in* S *if* $f(S_j) > f(\{S_1, \ldots, S_{j-1}\})$.

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

Algorithm 3 starts by ordering the items of I according to the greedy policy 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 policy N is updated by moving N_j to the front. Otherwise, the current policy 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 policy N in Line 5 and the relative ordering of any pair of items (N_k, N_l) with $1 \leq k < l < j$ remains unchanged.

Theorem 3. *Let* (I, f, w) *be an instance of* SMUC. *Then, for any universal policy* N *returned by Algorithm 3, we have* $g^N(B) \geq \Phi(I, f, w, B)$ *for any reasonable knapsack capacity* B.

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

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

Now, we prove Theorem 3.

Proof of Theorem 3. Let N be the universal policy returned by Algorithm 3, S be the greedy policy, and $B \geq \max_{i \in I} w_i$. 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 7 the single item i^* with $i^* = S_k$, $k > 1$. Then, when the while loop in Line 3 of Algorithm 1 ends, we have $R = \{S_1, \ldots, S_{k-1}\}$ 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 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 and it follows $g^N(B) \geq f(N_1) \geq f(S_j) \geq f(R) = \Phi(I, f, w, B)$, analogously to Case 1. □

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 than or equal to 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 up to n items in Line 5 of Algorithm 3. Now, we simplify Algorithm 3 to Algorithm 4, described below, by starting with the greedy policy and swapping *only* the swap item with maximum objective value. 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 greedy policy among all swap items.

Algorithm 4:

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

Determine the greedy policy S
$N \leftarrow S$
for $j = n$ down to 2 **do**
    if N_j is a swap item **then**
        $N \leftarrow (N_j, N_1, \ldots, N_{j-1}, N_{j+1}, \ldots, N_n)$
        break
return N

Definition 9. *For any instance (I, f, w) of* SMUC*, we call the universal policy returned by Algorithm 4 the* ***improved greedy policy*** *of (I, f, w).*

For any instance of SMUC, the improved greedy policy is better than or equal to Algorithm 1.

Theorem 4. *Let (I, f, w) be an instance of* SMUC*, let N be the improved greedy policy, and B be a reasonable knapsack capacity. Then, $g^N(B) \geq \Phi(I, f, w, B)$.*

We briefly point out that the improved greedy policy has a better robustness factor than ω if the curvature of the submodular function f is strictly less than 1.

Definition 10. *Let (I, f, w) be an instance of* SMUC*. The* ***curvature*** *of f is defined as*

$$c = 1 - \min_{i \in I} \frac{f(I) - f(I \setminus \{i\})}{f(i)}.$$

Corollary 2. *Let (I, f, w) be an instance of* SMUC*, c be the curvature of f, N be the improved greedy policy and B be a reasonable knapsack capacity. Then,*

$$g^N(B) \geq \frac{1 - x}{2 - (2 - c)x} f(\mathrm{Opt}^B),$$

where x is the unique solution of $1 - e^{-cz} = c\frac{1-z}{2-(2-c)z}$, $z \in [0, 1]$.

Corollary 2 is a direct consequence of Corollary 4, which is stated below in Section 4, where we discuss in detail approximation results depending on the curvature of the submodular function.

4. Adaptively maximizing a submodular function under an unknown, possibly unreasonable, knapsack capacity.

It is unknown to us whether, for each instance of SMUC, there exists a universal policy such that, for any unknown and possibly unreasonable knapsack capacity, packing with

discarding according to this policy approximates the optimal solution value within a constant factor. However, the following example demonstrates that it is impossible to construct, for each instance of SMUC, a universal policy such that packing with discarding according to this policy yields, for any knapsack capacity, a set of items that is at least as valuable as the return of Algorithm 1.

Example 3. *Let $I = \{a, b, c\}$ be a set of items with weights $w_a = 2.9$, $w_b = 2$, $w_c = 1$, and values $v_a = 3$, $v_b = 2$, $v_c = 1$. 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 $f(\{a, c\}) = 4$. By packing a knapsack of capacity $\bar{B}$ with discarding, according to any universal policy in which item b is ordered earlier than item c, either the set $\{a, b\}$ or the set $\{b, c\}$ will be packed, as $w(\{a, b, c\}) > 5$. Since $f(\{a, b\}) = f(\{b, c\}) = 3$, Algorithm 1 returns a better solution for knapsack capacity $\bar{B}$ than any universal policy in which b is ordered earlier than c.

In contrast, Kawase et al. (2019) demonstrated that, for every instance of SMUC and any unknown but fixed knapsack capacity, an optimal solution can be constantly approximated by packing adaptively. Klimm and Knaack (2022) improved the constant approximation factor to ω, which is currently the best known.

We use the non-adaptive packing results of Section 3 to give a quite simple adaptive algorithm that matches the approximation guarantee ω for adaptive packing under arbitrary knapsack capacities. Our adaptive algorithm is more intuitive and allows for a simpler proof than the one by Klimm and Knaack (2022). Furthermore, our proof follows directly by a generalization of Theorem 4 to slightly weaker conditions on the knapsack capacity.

Definition 11. *Let (I, f, w) be an instance of* SMUC. *We define $I^B := \{i \in I : w_i \leq B\}$ and $I^{<B} := \{i \in I : w_i < B\}$ for $B > 0$. Further, we define the restriction of (I, f, w) to a set $T \subseteq I$ as $(I, f, w)|_T := (T, f|_{2^T}, w|_T)$.*

Algorithm 5:

Input: An instance (I, f, w) of SMUC.
Output: Approximately optimal solution.

```
Let N be the improved greedy policy of (I, f, w)
while N_1 does not fit into the knapsack do
    (I, f, w) ← (I, f, w)|_{I^{<w_{N_1}}}
    Let N be the improved greedy policy of (I, f, w)
A ← items packed according to N without discarding
return A
```

Basically, while the first item of the improved greedy policy does not fit into the knapsack, Algorithm 5 discards it and all others of at least the same weight, recomputes

the policy, and repeats discarding again until the first item does fit. Then it packs according to N without discarding.

To prove the approximation guarantee of Algorithm 5, we generalize Theorem 4, by demonstrating that for each instance of SMUC and any knapsack capacity packing without discarding according to the improved greedy policy approximates the optimal solution value by at least ω if the first item of the improved greedy policy fits into the knapsack.

Theorem 5. *Let (I, f, w) be an instance of* SMUC *and B be a knapsack capacity with $B \geq w_{N_1}$, in which N denotes the improved greedy policy of (I, f, w). Then, $g^N(B) \geq \omega f(\mathrm{Opt}^B)$.*

Proof. Let S be the greedy policy of (I, f, w) and let S^B be the greedy policy of $(I, f, w)|_{I^B}$. Let S_l, $l \in \{1, \ldots, n\}$ be the first item that does not fit into the knapsack with capacity B, by packing according to S. To prove the claim, we would ideally want $(S_1, \ldots, S_l) = (S_1^B, \ldots, S_l^B)$, but this equality is not valid in general. Therefore, we introduce an auxiliary knapsack capacity B'. Let N_j, $j \in \{2, \ldots, n\}$ be the first item that does not fit into the knapsack with capacity B, by packing according to N. Choose $\varepsilon > 0$ such that $B' \coloneqq w(\{N_1, \ldots, N_j\}) - \varepsilon \geq B$ and $w_{N_j} \leq B'$. Hence, $w(\{N_1, \ldots, N_{j-1}\}) \leq B'$. Let $S^{B'}$ be the greedy policy of $(I, f, w)|_{I^{B'}}$.

Recall that the improved greedy order N is constructed by starting with the greedy order S and moving the swap item with the highest index in S to the front, if one exists. Hence, it follows directly that $N = S$ if there is no swap item in S, and by $w_{S_i} = w_{N_i} \leq B'$ for all $i \in \{1, \ldots, j\}$, we have $l = j$ and $(S_1, \ldots, S_l) = (S_1^{B'}, \ldots, S_l^{B'})$.

If N_1 is a swap item in S, we distinguish two cases. First, let $N_1 = S_m$ with $m \leq l$. Then we have $\{N_1, \ldots, N_j\} = \{S_1, \ldots, S_l\}$, and since $w(\{N_1, \ldots, N_{j-1}\}) \leq B \leq B'$ and $w(N_j) \leq B'$, it follows directly that $w_{S_i} \leq B'$ for $i \in \{1, \ldots, l\}$ implying $(S_1, \ldots, S_l) = (S_1^{B'}, \ldots, S_l^{B'})$. Now, let $N_1 = S_m$ with $m > l$. Then we have $w_{S_i} \leq w_{S_m} = w_{N_1} \leq B'$ for all $i \in \{1, \ldots, m\}$, since N_1 is a swap item in S. This directly implies $(S_1, \ldots, S_l) = (S_1^{B'}, \ldots, S_l^{B'})$.

Since $(S_1, \ldots, S_l) = (S_1^{B'}, \ldots, S_l^{B'})$, Algorithm 1 applied to (I, f, w) and the knapsack capacity B' either returns the single item S_l or the set $\{S_1, \ldots, S_{l-1}\}$. If $N_1 = S_m$ with $m \leq l - 1$, then S_l is not a swap item in S and Algorithm 1 returns $\{S_1, \ldots, S_{l-1}\}$. Hence,

$$g^N(B) = f(\{S_1, \ldots, S_{l-1}\}) = \Phi(I, f, w, B') \geq \omega f(\mathrm{Opt}^{B'}) \geq \omega f(\mathrm{Opt}^B), \tag{2}$$

where the second last inequality is due to Proposition 1.

If $N_1 = S_m$ with $m > l - 1$, we have

$$\begin{aligned} g^N(B) &\geq f(S_m) \geq \max\{f(\{S_1, \ldots, S_{l-1}\}), f(S_l)\} \\ &= \Phi(I, f, w, B') \geq \omega f(\mathrm{Opt}^{B'}) \geq \omega f(\mathrm{Opt}^B), \end{aligned} \tag{3}$$

where the second inequality follows, since $m > l - 1$ and S_m is a swap item, and the second last inequality follows from Proposition 1. □

It follows directly by Theorem 5:

Corollary 3. *For every instance of* SMUC *and any knapsack capacity B, we have*

$$f(A) \geq \omega f(\mathrm{Opt}^B),$$

for the set A returned by Algorithm 5.

4.1. Curvature depending performance of Algorithm 5.

We briefly discuss the performance of Algorithm 5 in dependence on the curvature of the submodular function.

Klimm and Knaack (2022) presented for their adaptive algorithm an approximation result depending on the curvature of the objective function of the given instance. Their adaptive algorithm is based on an alternative greedy algorithm, in the following called AGREEDY. Klimm and Knaack (2022) showed that Algorithm 1 is always at least as good as AGREEDY (Klimm and Knaack 2022, Proposition 2), and that AGREEDY approximates the optimal solution value for every instance of SMUC and any knapsack capacity by at least $\frac{1-x}{2-(2-c)x}$, where c is the curvature of the objective function, and x is the unique solution of the equation $1 - e^{-cz} = c\frac{1-z}{2-(2-c)z}$, $z \in [0,1]$ (Klimm and Knaack 2022, Theorem 5 and the subsequent paragraph).

These results directly imply:

Proposition 2. *Let (I, f, w) be an instance of* SMUC*, c be the curvature of f and x be the unique solution of $1 - e^{-cz} = c\frac{1-z}{2-(2-c)z}$, $z \in [0,1]$. Then,*

$$\Phi(I, f, w, B) \geq \frac{1-x}{2-(2-c)x} f(\mathrm{Opt}^B),$$

for any knapsack capacity B.

By Proposition 2, we can replace ω in the Inequalities (2) and (3) by $\frac{1-x}{2-(2-c)x}$, where c is the curvature of the objective function and x is the unique solution of the equation $1 - e^{-cz} = c\frac{1-z}{2-(2-c)z}$, $z \in [0,1]$. Hence, Theorem 4 generalizes to:

Corollary 4. *Let (I, f, w) be an instance of* SMUC*, c be the curvature of f and x be the unique solution of the equation $1 - e^{-cz} = c\frac{1-z}{2-(2-c)z}$, $z \in [0,1]$. Further, let B be a knapsack capacity with $B \geq w_{N_1}$, in which N denotes the improved greedy policy. Then,*

$$g^N(B) \geq \frac{1-x}{2-(2-c)x} f(\mathrm{Opt}^B).$$

It follows directly by Corollary 4:

Corollary 5. *Let (I, f, w) be an instance of* SMUC*, c be the curvature of f, B be an arbitrary knapsack capacity, and A be the set returned by Algorithm 5. Then,*

$$f(A) \geq \frac{1-x}{2-(2-c)x} f(\mathrm{Opt}^B),$$

where x is the unique solution of $1 - e^{-cz} = c\frac{1-z}{2-(2-c)z}$, $z \in [0,1]$.

Thus, the approximation guarantee of Algorithm 5 matches that of the adaptive algorithm in Theorem 9 by Klimm and Knaack (2022).

For completeness, we remark that both adaptive algorithms, Algorithm 5 and the one by Klimm and Knaack (2022), inherit their approximation guarantee depending on the curvature from the underlying Algorithm 1 and AGREEDY, respectively.

5. Non-adaptively maximizing a modular function under an arbitrary unknown knapsack capacity.

In this section, we consider non-adaptive packing with discarding for MMUC with an *arbitrary* unknown knapsack capacity.

Definition 12. *Let (I, f, w) be an instance of* SMUC *and N be a universal policy of I. The* ***exhaustively packed set*** *$E(N, B)$ packed with discarding according to the policy 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 exhaustively packed set $E(N, B)$ and is denoted by*

$$h^N \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0},\ h^N(B) \coloneqq f(E(N, B)).$$

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

We present an algorithm that computes, for every instance of MMUC, a universal policy better than 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 2. *(Korte and Vygen 2012, Proposition 17.6.) Let (I, f, w) be an instance of* MMUC*, then it holds, for any $B \geq 0$, that $\Phi(I, f, w, B) \geq \frac{1}{2} f(\mathrm{Opt}^B)$.*

It follows by Lemma 2 that packing with discarding according to a universal policy better than or equal to Algorithm 1 approximates an optimal solution value by at least $\frac{1}{2}$, for any knapsack capacity. Thus, in this setting, we match the result of Disser et al. (2017). However, we will present an algorithm that is more intuitive than theirs and allows for an uncomplicated and straightforward proof of correctness.

The following example illustrates that the improved greedy policy and the universal policy returned by Algorithm 3 are not suitable for non-adaptive packing with discarding for MMUC with an arbitrary knapsack capacity, as they can perform arbitrarily poorly.

Example 4. *Let $I = \{a, b, c\}$ with weights $w_a = \frac{M}{2}$, $w_b = 1$ and $w_c = M$, where $M > 1$. Further, let $f\colon 2^I \to \mathbb{R}_{\geq 0}$ be a modular function defined by $f(\emptyset) = 0$, $f(a) = \frac{M}{2}$, $f(b) = 2$, and $f(c) = M+1$. The greedy policy of I is $S = (b, c, a)$. Since $f(c) = M+1 > 2 = f(b)$, item c is a swap item and since $f(a) = \frac{M}{2} < M + 3 = f(c, b)$, item a is not a swap item, and therefore Algorithm 3 returns the universal policy $N = (c, b, a)$, which also corresponds to the improved greedy policy.*

Let $B \coloneqq \frac{M}{2}$. Then we have

$$\lim_{M\to\infty} \frac{h^N(B)}{f(\mathrm{Opt}^B)} = \lim_{M\to\infty} \frac{f(b)}{f(a)} = \lim_{M\to\infty} \frac{2}{\frac{M}{2}} = 0.$$

To obtain a universal policy better than or equal to Algorithm 1, we first generalize the notion of swap items.

Definition 13. *Let (I, f, w) be an instance of* MMUC *and $N = (N_1, \ldots, N_n)$ be an arbitrary policy of I. For $2 \leq j \leq n$ we call N_j a* ***generalized swap item*** *in N, if there exists $k \in \{1, \ldots, j-1\}$ with $f(N_j) > f(\{N_k, \ldots, N_{j-1}\})$.*

Notice that whenever N_j is a generalized swap item, simply $k = j - 1$ would suffice. But to make it useful, we will choose the smallest possible k next. Now, we modify Algorithm 3 by starting with the greedy policy S and moving any generalized swap item N_j to the smallest position $k < j$, so that $\{N_k, \ldots, N_{j-1}\}$ is the largest set directly in front of N_j with a function value smaller than the function value of N_j.

Algorithm 6:

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

Determine the greedy policy S
$N^{(1)} \leftarrow S$
for $j = 2, \ldots, n$ **do**
  if $N_j^{(j-1)}$ is a generalized swap item in $N^{(j-1)}$ **then**
    $k \leftarrow \min\{l \in \{1, \ldots, j-1\} \colon f(\{N_l^{(j-1)}, \ldots, N_{j-1}^{(j-1)}\}) < f(N_j^{(j-1)})\}$
    $N^{(j)} = (N_1^{(j-1)}, \ldots, N_{k-1}^{(j-1)}, N_j^{(j-1)}, N_k^{(j-1)}, \ldots, N_{j-1}^{(j-1)}, N_{j+1}^{(j-1)}, \ldots, N_n^{(j-1)})$
return $N = N^{(n)}$

We demonstrate that for any instance of MMUC, Algorithm 6 computes a universal policy better than or equal to Algorithm 1.

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

To prove Theorem 6, we need a simple observation regarding the restriction of universal policies.

Definition 14. *Let I be a set of items, N be 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 $|T| = m \leq n$, 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 $1 \leq i \leq m$.*

Notice that any generalized swap item never crosses a heavier item, and we trivially have that the greedy policy I restricted to items lighter than any knapsack capacity B equals the greedy policy regarding the instance that consists of all items lighter than B. Combining these two observations, we obtain the following.

Lemma 3. *Let (I, f, w) be an instance of* MMUC*, and N be the universal policy returned by Algorithm 6 applied to the instance (I, f, w). Let $B \geq 0$ be a knapsack capacity, and N^B be the universal policy returned by Algorithm 6 applied to the instance $(I, f, w)|_{I^B}$, where I^B is defined as in Definition 11. Then, $N|_{I^B} = N^B$.*

Now, we prove Theorem 6.

Proof of Theorem 6. We distinguish two cases.

If $B \geq \max_{i \in I} w_i$: Let S be the greedy policy of (I, f, w). If Algorithm 1 outputs the single item S_j with $j > 1$, then S_j is a swap item, thus a generalized swap item. Hence, in the j-th iteration of the for loop in Algorithm 6, S_j is moved to the front of N, and any item ordered to the front of N in a later iteration must be a swap item with a greater objective value than S_j. Therefore, it follows directly that $h^N(B) \geq f(N_1) \geq f(S_j) = \Phi(I, f, w, B)$.

Now, assume that Algorithm 1 outputs $R = \{S_1, \dots, S_{j-1}\}$ with $2 \leq j \leq n$.

Define $q \colon \{S_2, \dots, S_n\} \to 2^I, S_i \mapsto \{x \in \{S_1, \dots, S_{i-1}\} \colon S_i \text{ is ordered somewhere before } x \text{ in } N^{(i)}\}$. Thus, for any S_i, $2 \leq i \leq n$, the set $q(S_i)$ denotes the items which S_i crosses in the $i-1$-th iteration of the for loop in Algorithm 6.

We have $f(i) \geq f(q(i))$ for every $i \in \{S_2, \dots, S_n\}$, by the notion of generalized swap items. This holds even in the case that i is not a generalized swap item, as this implies $q(i) = \emptyset$.

Notice that the relative order of any pair of items S_h, S_i with $1 \leq h < i \leq n$ remains the same in all policies $N^{(k)}$ with $i \leq k \leq n$. Therefore, for any $1 \leq h < n$ with $S_h \in R \setminus E(N, B)$, there has to exist $i > h$ with $S_i \in E(N, B) \setminus R$ and $S_i = N_k$ and $S_h = N_l$ with $k < l$ (otherwise S_h could be packed by packing according to N). This implies $S_h \in q(S_i)$.

It is straightforward to see that the claim follows now immediately by the modularity of f: For any h with $S_h \in R \setminus E(N, B)$ define $l(S_h)$ as an arbitrary $S_i \in E(N, B) \setminus R$ with $i > h$, $S_i = N_k$ and $S_h = N_l$ with $k < l$. Then, we have

$$\begin{aligned} f(R \setminus E(N, B)) &= \sum_{x \in R \setminus E(N,B)} f(x) \leq \sum_{y \in \{l(x) | x \in R \setminus E(N,B)\}} f(y) \leq \sum_{y \in E(N,B) \setminus R} f(y) \\ &= f(E(N, B) \setminus R), \end{aligned}$$

thus $\sum_{y \in R} f(y) \leq \sum_{y \in E(N,B)} f(y)$, and therefore $\Phi(I, f, w, B) = \sum_{y \in R} f(y) \leq \sum_{y \in E(N,B)} f(y) = h^N(B)$.

If $B < \max_{i \in I} w_i$: Let N^B be the universal policy returned by Algorithm 6 for the instance $(I, f, w)|_{I^B}$. By Lemma 3, we have $N|_{I^B} = N^B$. Therefore, $h^{N|_{I^B}}(B) = h^{N^B}(B) \geq \Phi(I, f, w, B)$, in which the last inequality follows from the previous case. □

6. Conclusion.

In this paper, we considered the problem of maximizing a monotone-increasing submodular function under an unknown knapsack capacity non-adaptively and adaptively. We presented the first algorithm that returns for any instance of SMUC a universal policy better than or equal to Algorithm 1 (due to Wolsey (1982)) for any reasonable knapsack capacity. Thus, we demonstrated that the optimal solution value of maximizing a monotone-increasing submodular function under an unknown *reasonable* knapsack capacity can be approximated non-adaptively and without discarding items by at least 0.357.

As a byproduct, we derived a simple adaptive algorithm that, for any instance and any knapsack capacity, returns a set whose value approximates the optimal solution value by at least 0.357, providing a simpler and more intuitive proof of this approximation factor than those previously known in the literature.

For the special case that the submodular function is modular, we presented an algorithm that generates a universal policy for every instance of MMUC better than or equal to Algorithm 1. Thus, we provided a simpler proof of the result by Disser et al. (2017), that the optimal solution value for maximizing a modular function under an unknown arbitrary knapsack capacity can be approximated non-adaptively with discarding by a factor of at least 0.5.

It remains an interesting open question whether non-adaptive packing with discarding can achieve 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.

Acknowledgements.

The authors thank the DFG for their support within RTG 2126 "Algorithmic Optimization". Martin Knaack announced to us a proof for Theorem 1 at OR2023 that we have not seen yet, hence we provide our own.

References

Ageev, A.A. and M.I. Sviridenko (1999). "An 0.828-approximation algorithm for the uncapacitated facility location problem." In: *Discrete Applied Mathematics* 93.2-3, pp. 149–156. DOI: 10.1016/S0166-218X(99)00103-1.

Bernstein, Aaron, Yann Disser, Martin Groß, and Sandra Himburg (2022). "General bounds for incremental maximization." In: *Mathematical Programming* 191, pp. 953–979. DOI: 10.1007/s10107-020-01576-0.

Cornuejols, Gerard, Marshall L. Fisher, and George L. Nemhauser (1977). "Exceptional Paper - Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms." In: *Management science* 23.8, pp. 789–810. DOI: 10.1287/mnsc.23.8.789.

Disser, Yann, Max Klimm, Annette Lutz, and David Weckbecker (2024). "Fractionally Subadditive Maximization under an Incremental Knapsack Constraint with Applications to Incremental Flows." In: *SIAM Journal on Discrete Mathematics* 38.1, pp. 764–789. DOI: 10.1137/23M1569265.

Disser, Yann, Max Klimm, Nicole Megow, and Sebastian Stiller (2017). "Packing a knapsack of unknown capacity." In: *SIAM Journal on Discrete Mathematics* 31.3, pp. 1477–1497. DOI: 10.1137/16M1070049.

Feige, Uriel (1998). "A threshold of ln n for approximating set cover." In: *Journal of the ACM (JACM)* 45.4, pp. 634–652. DOI: 10.1145/285055.285059.

Goemans, Michel X. and David P. Williamson (1995). "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming." In: *Journal of the ACM (JACM)* 42.6, pp. 1115–1145. DOI: 10.1145/227683.227684.

Kawase, Yasushi, Hanna Sumita, and Takuro Fukunaga (2019). "Submodular Maximization with Uncertain Knapsack Capacity." In: *SIAM Journal on Discrete Mathematics* 33.3, pp. 1121–1145. DOI: 10.1137/18M1174428.

Klimm, Max and Martin Knaack (2022). "Maximizing a Submodular Function with Bounded Curvature Under an Unknown Knapsack Constraint." In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*. Vol. 245. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 49:1–49:19. DOI: 10.4230/LIPIcs.APPROX/RANDOM.2022.49.

Korte, Bernhard and Jens Vygen (2012). *Combinatorial Optimization: Theory and Algorithms*. 5th ed. Springer.

Navarra, Alfredo and Cristina M. Pinotti (2017). "Online knapsack of unknown capacity: How to optimize energy consumption in smartphones." In: *Theoretical Computer Science* 697, pp. 98–109. DOI: 10.1016/j.tcs.2017.07.029.

Nemhauser, G. L., L. A. Wolsey, and M. L. Fisher (1978). "An analysis of approximations for maximizing submodular set functions-I." In: *Mathematical Programming* 14, pp. 265–294. DOI: 10.1007/BF01588971.

Sviridenko, Maxim (2004). "A note on maximizing a submodular set function subject to a knapsack constraint." In: *Operations Research Letters* 32.1, pp. 41–43. DOI: 10.1016/S0167-6377(03)00062-2.

Wolsey, Laurence A. (1982). "Maximising Real-Valued Submodular Functions: Primal and Dual Heuristics for Location Problems." In: *Mathematics of Operations Research* 7.3, pp. 410–425. DOI: 10.1287/moor.7.3.410.

A. Appendix.

A.1. Proof of Theorem 2.

To prove Theorem 2, we need the following observation due to Wolsey (1982).

Proposition 3. *(Wolsey 1982, Theorem 2) Let (I, f, w) be an instance of* SMUC *and let D be a knapsack capacity. Further, let R^D with $w(R^D) = D$ be the set constructed in the while loop of Algorithm 1 applied to the instance (I, f, w) and the knapsack capacity D. Then, for any knapsack capacity $B \geq D$, we have $\frac{f(R^D)}{f(\mathrm{Opt}^B)} \geq 1 - e^{-\frac{D}{B}}$.*

Now, we prove Theorem 2.

Proof. Let N be the universal policy returned by the *modified* Algorithm 2, S be the greedy policy in the *modified* Algorithm 2 and $B \geq \max_{i \in I} w(i)$. Let $R^B = \{S_1, \ldots, S_{k-1}\}$ with $2 \leq k \leq n$ be the set constructed in the while loop 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 of Algorithm 1 ends, we have $\sum_{j=1}^{k} w(S_j) > B \geq \sum_{j=1}^{k-1} w(S_j) = w(R^B)$. 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 $2w(R^B) > w(R^B) + w(S_k) > B$.

Applying Algorithm 1 to the instance (I, f, w) and the capacity $D = w(R^B)$, Algorithm 1 constructs the set R^B and it holds $w(R^B) = D$. It follows by Proposition 3 that

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

□

A.2. Proof of Theorem 4.

Proof. Let S be the greedy policy and $B \geq \max_{i \in I} w_i$. To prove the claim, we compare $g^N(B)$ with $\Phi(I, f, w, B)$. If Algorithm 1 outputs in Line 7 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 $R = \{S_1, \ldots, S_{k-1}\}$ with $2 \leq k \leq n$. Clearly, S_k is not a swap item, since otherwise Algorithm 1 would have returned S_k, and therefore we have $N_1 \neq S_k$. 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 $N_1 = S_j$, $j > k$. Then, S_j is a swap item, and by the notion of swap items $f(S_j) > f(\{S_1, \ldots, S_{j-1}\}) \geq f(R)$. It follows immediately that $g^N(B) \geq f(N_1) \geq f(R) = \Phi(I, f, w, B)$.

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

A.3. Proof of Lemma 3.

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

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 $k \geq l$.

Let $Z^l_t = S_l$ with $t \in \{1, \ldots, l\}$. Since $f(S_l) \geq f(S_{j+1})$ for any $j \geq l$, by the modularity of f and $w_{S_j} \leq \max_{i \in I} w_i$, the inequality $f(S_{j+1}) > f(\{Z^j_i \ldots, Z^j_j\})$ is not true for any $i \leq t$. Moreover, for any $j \geq l$ the inequality $f(S_{j+1}) > f(\{Z^j_i, \ldots, Z^j_j\})$ for $t < i \leq j$ is true if and only if $f(\bar{S}_j) > f(\{\bar{Z}^{j-1}_{i-1}, \ldots, \bar{Z}^{j-1}_{j-1}\})$.

Therefore, for any $j \geq l$ we have $Z^j_{k+1} = \bar{Z}^j_k$ for $k \geq t$ and $Z^j_k = \bar{Z}^j_k$ for $k < t$. This directly implies that the policy Z^n returned by Algorithm 6 applied to (I, f, w) with S_l removed equals the policy $\bar{Z}^n$ returned by Algorithm 6 applied to $(I, f, w)|_{I \setminus \{S_l\}}$. Now, the claim follows by induction. □