Maximizing a Submodular Function with Bounded Curvature Under an Unknown Knapsack Constraint

Max Klimm ✉
Institute for Mathematics, Technische Universität Berlin, Germany

Martin Knaack ✉
Institute for Mathematics, Technische Universität Berlin, Germany

Abstract

This paper studies the problem of maximizing a monotone submodular function under an unknown knapsack constraint. A solution to this problem is a policy that decides which item to pack next based on the past packing history. The robustness factor of a policy is the worst case ratio of the solution obtained by following the policy and an optimal solution that knows the knapsack capacity. We develop an algorithm with a robustness factor that is decreasing in the curvature c of the submodular function. For the extreme cases $c = 0$ corresponding to a modular objective, it matches a previously known and best possible robustness factor of $1/2$. For the other extreme case of $c = 1$ it yields a robustness factor of ≈ 0.35 improving over the best previously known robustness factor of ≈ 0.06.

2012 ACM Subject Classification Mathematics of computing → Submodular optimization and polymatroids; Theory of computation → Packing and covering problems; Theory of computation → Online algorithms

Keywords and phrases submodular function, knapsack, approximation algorithm, robust optimization

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2022.49

Category APPROX

Acknowledgements The authors wish to thank Daniel Schmidt genannt Waldschmidt for fruitful discussion.

1 Introduction

This paper is concerned with the problem

$$\text{maximize} \left\{ f(S) \;\middle|\; S \subseteq N \text{ and } \sum_{i \in S} s(i) \leq C \right\} \tag{1}$$

of maximizing a submodular, monotone, and normalized function $f : 2^N \to \mathbb{R}_{\geq 0}$ under a knapsack constraint, where N is a finite set of items, $s(i) \in \mathbb{R}_{>0}$ is the size of item $i \in N$, and $C \in \mathbb{R}_{>0}$ is a knapsack capacity. This optimization problem is an important abstraction of many problems that appear in various applications, such as facility location (Cornuéjols et al. [4]), sensor placement (Krause and Guestrin [12], Krause et al. [13]), marketing in social networks (Kempe et al. [10]), and maximum entropy sampling (Lee [14]).

For the special case of a cardinality constraint where $s(i) = 1$ for all $i \in N$, a straightforward greedy algorithm by Nemhauser et al. [17] computes a solution with an approximation guarantee of $1 - 1/e$ and this ratio is best possible for any polynomial algorithm unless $\mathsf{P} = \mathsf{NP}$ (Feige [7]). For the case of a general knapsack constraint, combining the greedy algorithm with a partial enumeration of all subsolutions with at most three items yields the same approximation guarantee (Sviridenko [19]).

While these results are tight, the algorithms often perform much better than their theoretical guarantees. In order to explain and quantify this phenomenon, Conforti and Cornuéjols [3] introduce the concept of the *curvature* of a submodular function. Recall that a function $f : 2^N \to \mathbb{R}_{\geq 0}$ is submodular if the marginal increase $f(A \cup \{u\}) - f(A)$ of an element $u \in N \setminus A$ is non-increasing as A increases. The curvature $c \in [0, 1]$ measures how much this marginal increase of an item u varies when varying A and is defined as

$$c = 1 - \min_{j \in N} \frac{f(N) - f(N \setminus \{j\})}{f(\{j\})},$$

where we further used that f is normalized, i.e., $f(\emptyset) = 0$. It is easy to see that $c = 0$ if and only if the function is modular (i.e., linear). The other extreme case $c = 1$ is, e.g., attained when f is the rank function of a matroid. Conforti and Cornuéjols [3] show that the greedy algorithm for the cardinality constraint case has an improved approximation guarantee of $(1 - e^{-c})/c$. A more sophisticated algorithm for the same problem by Sviridenko et al. [20] achieves an even better approximation guarantee of $1 - c/e - \varepsilon$ for any $\varepsilon > 0$.

In all of the results above it is assumed that all data of the problem (1) is given completely. In this paper, we consider a variant of the problem where the set of items N, their sizes $s(i) \in \mathbb{R}_{>0}$, and the function $f : 2^N \to \mathbb{R}_{\geq 0}$ are known, but the knapsack capacity $C \in \mathbb{R}_{>0}$ is unknown. In this context, a solution to the problem is a policy that decides which item to pack next, based on the previous packing history. More formally, a policy is a binary decision tree where nodes correspond to items with the property that no item appears more than once on a path from the root to a leaf. The item at the root of the tree is the item that is attempted to be packed first. If it fits, it is irrevocably included in the solution, the (unknown) capacity is reduced by the size of the item, and the solution proceeds with the left subtree of the decision tree. If the item does not fit, it is discarded, the (unknown) capacity stays the same, and the solution proceeds with the right subtree. This process stops after a leaf is reached. The assumption that the policy can resume packing smaller items after a larger item does not fit is suitable when the knapsack capacity is interpreted as a monetary budget. Generally speaking, such packing policies are desirable when packing problems of this kind have to be solved repeatedly for varying knapsack capacities. For illustration, consider the marketing problem in social networks. By analyzing the social network, a packing policy can be constructed that can then be used in order to run marketing campaigns *for all possible budgets*, without the need to rerun any optimization. In a similar vein, consider the problem of maximum entropy sampling. The Shannon entropy of a set of (dependent) random variables is a submodular function of the (index set) of the variables. Suppose that observing the realization of a random variable comes at a cost (for market research, for evaluating the data, etc.). With our algorithm, one can compute a policy that *for all budgets* allows to retrieve close to optimal information without any need to rerun the optimization for different budgets.

In the examples above, we clearly want to obtain solutions that are good for any possible capacity. We evaluate the quality of a policy in terms of its *robustness factor*. Fix an instance of (1), and a corresponding policy Π. For a capacity $C \in \mathbb{R}_{>0}$, let $\Pi(C)$ be the set of items packed by the policy when the knapsack capacity is C, and let $\textsc{Opt}(C)$ be the items included in an optimal solution for capacity C. The robustness factor is defined as $\alpha := \inf_C f(\Pi(C))/f(\textsc{Opt}(C))$. A policy with robustness factor of $\alpha \in [0, 1]$ is called α-optimal.

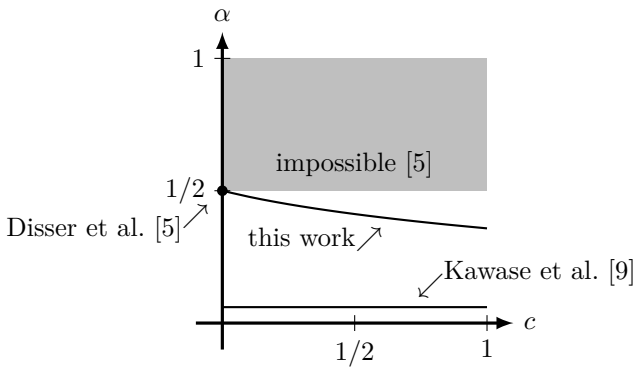

Figure 1 Robustness factors α as a function of the curvature c achieved by this and previous work.

1.1 Our Results and Techniques

For the case that f is modular (corresponding to the case that $c = 0$), Disser et al. [5] show that every instance admits a $1/2$-optimal policy, and that the factor of $1/2$ is best possible. Kawase et al. [9] consider the fully submodular case corresponding to the case $c = 1$. They provide a deterministic policy with robustness factor $2(1 - 1/e)/21 \approx 0.06$ and a randomized policy with robustness factor of $(1 - 1/e)/2 \approx 0.32$.

We provide a deterministic polynomial algorithm that constructs a deterministic policy Π with a robustness factor of

$$\alpha = \frac{1 - x}{2 - (2 - c)x} \tag{2}$$

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

For the most general case of a submodular function with curvature $c = 1$, this yields a robustness factor of ≈ 0.35 which improves over the factor of ≈ 0.06 by Kawase et al.; for smaller values of $c < 1$ the robustness factor increases and retaines the optimal factor of $\alpha = 1/2$ in the modular case when $c = 0$. For an illustration; see Figure 1.

A central technique for solving submodular maximization problems with a known or unknown knapsack capacity are greedy algorithms, and this work is no exception. Disser et al. [5] compare the solution obtained by their policy with a greedy algorithm called MGreedy that either takes the greedy sequence or the first item that does not fit the knapsack anymore. As discussed by Kawase et al. [9] this approach seems difficult to apply to submodular functions because the greedy sequence is different for different sizes of the knapsack due to the substitute effects among the items for the objective. They instead single out valuable items that provide a significant ratio of the optimum solution. This approach, however, comes at the expense of a much lower robustness factor.

We circumvent this issue by analyzing a different kind of greedy algorithm that we call AGreedy and that seems to be more compatible with robust policies. We first analyze this algorithm for the case of a known knapsack capacity in Section 3 and show that it provides an approximation guarantee as in (2). As a byproduct of our analysis, we further obtain that the MGreedy algorithm also has the approximation guarantee as in (2). This generalizes a result of Wolsey [22] who analyzed this algorithm only for the general submodular case where $c = 1$. In Section 4, we then devise a robust policy that achieves a robustness ratio that is at least as good as the approximation guarantee of AGreedy.

1.2 Further Related Work

The problem of maximizing a submodular function under different constraints has a long history in the optimization literature. Nemhauser et al. [18] consider the problem of maximizing a monotonic submodular function under a cardinality constraint and show that the greedy algorithm that iteratively adds an item that maximizes the increase of the objective function achieves an approximation guarantee of $(1 - 1/e) \approx 0.63$. Nemhauser and Wolsey [17] prove that this ratio is best possible for algorithms that have only access to f via a value oracle that can only be queried a polynomial number of times. For the special case that f is given explicitly and corresponds to a maximum coverage function, there is no better approximation possible in polynomial time, unless $\mathsf{P} = \mathsf{NP}$, as shown by Feige [7]. Wolsey [22] considers the more general problem of maximizing a submodular function under a knapsack constraint and achieves an approximation guarantee of $1 - e^{-x} \approx 0.35$ where x is the unique root of the equation $e^x = 2 - x$. Sviridenko [19] shows that a combination of the greedy algorithm with a partial enumeration scheme achieves an approximation guarantee of $1 - 1/e$. Another way to generalize the cardinality constrained case is to allow for arbitrary matroid constraints. For this case, the greedy algorithm yields an approximation guarantee of $1/2$, as shown by Fisher et al. [8]. Calinescu et al. [2] achieve a $1 - 1/e$ approximation by solving a fractional relaxation of the problem and combining it with a suitable rounding technique.

Conforti and Cornuéjols [3] introduce the curvature c as a measure for the non-linearity of a (submodular) function and show that the greedy algorithm has an approximation guarantee of $(1 - e^{-c})/c$ for the case of a cardinality constraint and $1/(c + 1)$ for the case of a matroid constraint. Vondrák [21] shows that the continuous greedy algorithm yields an approximation guarantee of $(1 - e^{-c})/c$ for the case of a matroid constraint, and proves that no better approximation is possible in the value oracle model with a polynomial number of queries. Sviridenko et al. [20] give an algorithm with approximation guarantee of $1 - c/e - \mathcal{O}(\varepsilon)$ for the problem with a matroid constraint. Yoshida [23] obtains the same approximation guarantee for the problem under a knapsack constraint. The algorithm relies on a continuous version of the greedy algorithm which seems to be incompatible with an unknown knapsack constraint since many items will be fractional during the course of the algorithm for smaller knapsack constraints. Also the distinction between small and large items which is elementary in the algorithm cannot be employed when the capacity is not known.

Packing problems with an unknown knapsack are studied by Megow and Mestre [15]. They consider the modular case and assume that the policy stops when an item does not fit the knapsack. In this setting, no constant robustness factor is achievable on all instances and Megow and Mestre provide a polynomial time approximation scheme (PTAS) for the computation of an optimal policy. Navarra and Pinotti [16] show how to construct a policy with robustness factor $1/2$ for instances that have the property that every item fits into the empty knapsack. Disser et al. [6] consider the optimization of a fractionally subadditive objective with the additional property that every singleton set has a value of 1, and give a policy with robustness factor of ≈ 0.30. For the case of an unknown cardinality constraint, there is no difference between policies that continue or stop packing after an item does not fit. Bernstein et al. [1] introduce a property on the objective function that they term accountability and that is more general than submodularity. They show that the optimal robustness factor for maximization of an accountable objective under an unknown cardinality constraint is between $1/(1 + \phi) \approx 0.38$ where ϕ is the golden ratio and 0.46.

2 Preliminaries

2.1 Submodular Functions

Let N be a finite set. A function $f : 2^N \to \mathbb{R}_{\geq 0}$ is called *monotone* if $f(A) \leq f(B)$ for every $A, B \in 2^N$ with $A \subseteq B$, is called *normalized* if $f(\emptyset) = 0$, and is called *submodular* if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \in 2^N$. For our purposes, it is without loss of generality to assume that $f(\{j\}) > 0$ for all $j \in N$ since an element j with $f(\{j\}) = f(\emptyset)$, by submodularity, has no influence on the value of f and, thus, can be removed from (1). It is well-known that a function f is submodular if and only if one of the following statements is satisfied:

$$f(A \cup \{u\}) - f(A) \geq f(B \cup \{u\}) - f(B) \quad \text{for all } A \subseteq B \subseteq N,\ u \in N \setminus B,$$
$$f(A \cup \{u_1\}) + f(A \cup \{u_2\}) \geq f(A \cup \{u_1, u_2\}) + f(A) \quad \text{for all } A \subset N,\ u_1, u_2 \in N \setminus A,\ u_1 \neq u_2.$$

If is straightforward to verify that a submodular and monotone function satisfies the following inequality, see, e.g., Nemhauser et al. [18] for a reference

$$f(B) \leq f(A) + \sum_{u \in B \setminus A} f(A \cup \{u\}) - f(A) \quad \text{for all } A \subseteq B \subseteq N. \tag{3}$$

2.2 Curvature

The curvature of a normalized, monotone and submodular function $f : 2^N \to \mathbb{R}_{\geq 0}$ is defined as

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

The following lemma summarizes a couple of inequalities that are valid for submodular functions with a given curvature that are easy to show yet useful for the remainder of the paper. For a proof, see Appendix A.

▶ **Lemma 1.** *For a normalized, monotonic, and submodular function $f : 2^N \to \mathbb{R}_{\geq 0}$ with curvature $c \in [0, 1]$, the following inequalities are satisfied:*

(i) $(1 - c)f(\{j\}) \leq f(A \cup \{j\}) - f(A)$ *for all $A \subset N$ and all $j \in N \setminus A$;*

(ii) $f(A \cup B) \geq f(A) + (1 - c) \sum_{i \in B} f(\{i\})$ *for all $A, B \subset N$ with $A \cap B = \emptyset$.*

2.3 Submodular Maximization under a Knapsack Constraint

An instance of the submodular maximization problem under a known knapsack constraint is given by a set of n items $N = \{i_1, i_2, \ldots, i_n\}$ where each item $i \in N$ has a size $s(i) \in \mathbb{R}_{>0}$. We further have given a monotone, normalized and submodular function $f : 2^N \to \mathbb{R}_{\geq 0}$ that assigns a value $f(A)$ to every subset $A \subseteq N$ of items, and a capacity $C \in \mathbb{R}_{>0}$. For a subset $A \subseteq N$, we write $s(A) := \sum_{i \in A} s(i)$. A solution to the problem is a set of items $A \subseteq N$. A solution A is called *feasible* if $s(A) \leq C$, and called *optimal* if $f(A) \geq f(B)$ for every feasible solution B.

An instance of the submodular maximization problem under an unknown knapsack constraint is as above except that we do not know the capacity $C \in \mathbb{R}_{>0}$, i.e., we are again given a set of items N, their sizes $s(i)$, $i \in N$ and the submodular function f. A solution to this problem is a policy Π that governs the order in which items are added to the solution.

(a) Modified Greedy Algorithm MGreedy.

```
1: G_0 ← ∅; j ← 1
2: U ← {i ∈ N | s(i) ≤ C}
3: while U ≠ ∅ do
4:     i_j ← arg max_{i∈U} { (f(G_{j−1}∪{i}) − f(G_{j−1})) / s(i) }
5:     if s(G_{j−1} ∪ {i_j}) ≤ C then
6:         G_j ← G_{j−1} ∪ {i_j}
7:         U ← U \ {i_j}
8:         j ← j + 1
9:     else
10:        break
11: k ← j − 1
12: if U = ∅ then
13:     return G_k
14: else
15:     if f(G_k) ≥ f({i_{k+1}}) then
16:         return G_k
17:     else
18:         return {i_{k+1}}
```

(b) Alternative Greedy Algorithm AGreedy.

```
1: G_0 ← ∅; j ← 1
2: U ← {i ∈ N | s(i) ≤ C}
3: while U ≠ ∅ do
4:     i_j ← arg max_{i∈U} { (f(G_{j−1}∪{i}) − f(G_{j−1})) / s(i) }
5:     if s(G_{j−1} ∪ {i_j}) ≤ C then
6:         G_j ← G_{j−1} ∪ {i_j}
7:         U ← U \ {i_j}
8:         j ← j + 1
9:     else
10:        break
11: k ← j − 1
12: if U = ∅ then
13:     return G_k
14: else
15:     if f(G_k) ≥ f(G_k∪{i_{k+1}})−f(G_k) then
16:         return G_k
17:     else
18:         return {i_{k+1}}
```

Figure 2 Greedy algorithms for maximizing a submodular function f over a knapsack constraint.

3 Submodular Knapsack Problem with Known Capacity

In this section, we analyze the approximation guarantee for two natural greedy algorithms that, for the sake, of a better distinction, we call *modified greedy algorithm* (MGreedy) and *alternative greedy algorithm* (AGreedy). The modified greedy algorithm was proposed and analyzed by Wolsey [22] where he shows that it has an approximation ratio of $1 - e^{-x} \approx 0.35$ where x is the unique root of the equation $e^x = 2 - x$. To the best of our knowledge, there is no better analysis of this algorithm for submodular functions with bounded curvature. The alternative greedy algorithm is a slight variation of this algorithm that we need in order to derive policies for the optimization problem with unknown knapsack constraints in Section 4.

Both algorithms first discard all items i that do not fit into an empty knapsack, i.e., $s(i) > C$. Then, the algorithms start in iteration 0 with an empty solution $G_0 = \emptyset$. In every iteration $j = 1, 2, \ldots$, both algorithms choose an item

$$i_j \in \arg\max \left\{ \frac{f(G_{j-1} \cup \{i\}) - f(G_{j-1})}{s(i)} \;\middle|\; i \in N \setminus G_{j-1} \right\}$$

that is not yet contained in the solution G_{j-1} and maximizes the ratio of the increment of the objective function and the size of the item. If item i_j still fits the knapsack, i.e., $s(G_{j-1} \cup \{i_j\}) \leq C$, then the item is added to the solution. Otherwise, the algorithm stops. Let k be the last index such that item i_k still fits into the knapsack.

Then, algorithm MGreedy either returns the better of the solutions G_k and $\{i_{k+1}\}$, i.e., it either returns the maximum prefix of the greedy sequence that still fits into the knapsack, or the first item that did not fit into the knapsack anymore. The alternative greedy also either returns G_k or $\{i_{k+1}\}$ but the rule when to return one of the solutions slightly differs. The item $\{i_{k+1}\}$ is only returned if the increment $f(G_k \cup \{i_{k+1}\}) - f(G_k)\}$ of adding it to G_k is larger than $f(G_k)$. In all other cases, G_k is returned.

Since MGreedy always returns the better of the two solutions G_k and $\{i_{k+1}\}$ while AGreedy may also return G_k even though $f(G_k) < f(\{i_{k+1}\})$, it is clear that the solution returned by MGreedy is always at least as good as the one returned by AGreedy. Despite this fact, we are still interested in analyzing AGreedy for two reasons. First, it turns out that AGreedy is better suited in order to design robust packing policies for the problem with an unknown knapsack capacity. Second, it will turn out, that in the worst case, the approximation guarantees that we obtain for MGreedy and AGreedy are actually the same.

In the following, we fix an instance of the submodular maximization problem under a knapsack constraint with known capacity. Furthermore, we assume that the items are ordered $N = \{i_1, i_2, \dots, i_n\}$ in the order as they would be considered by the greedy algorithms. We also set $s_j = s(i_j)$ for all $j \in \{1, \dots, n\}$. We further let k be the maximal prefix of this ordering that still fits into the knapsack, i.e., $k = \max\{j \in \{1, \dots, n\} \mid \sum_{i=1}^{j} s(i) \leq C\}$. Note that we order the items in this order beyond the $(k+1)$-st item when the algorithms stop. We further set $G_j = \bigcup_{l=1,\dots,j}\{i_l\}$ and $\delta_j = f(G_j) - f(G_{j-1})$ for all $j \in \{1, \dots, n\}$. We let MG denote the set of items returned by MGreedy, and let AG denote the set of items returned by AGreedy. Let Opt denote the set of items in an optimal solution. Additionally, let χ_j be the indicator for $i_j \in$ Opt, i.e., $\chi_j = 1$ if $i_j \in$ Opt, and $\chi_j = 0$, otherwise. We further set $S_j = s(\text{Opt} \cap G_j)$.

Summarizing the above discussion, the following result is immediate.

▶ **Proposition 2.** *For every instance, $f(MG) \geq f(AG)$.*

We first provide a lemma that bounds the increase of the value of the greedy solution from below in two different ways. The proofs use standard submodularity arguments as well as the property of the greedy sequence; for the proof see Appendix B.

▶ **Lemma 3.** *For all $j \in \{1, \dots, k+1\}$, we have*

(i) $\delta_j \geq \frac{cs_j}{C}\Big(f(\text{Opt}) - \sum_{m=1}^{j-1}\delta_m\Big) + \frac{(1-c)s_j}{C - S_{j-1}}\Big(f(\text{Opt}) - \sum_{m=1}^{j-1}\chi_m\delta_m\Big)$,

(ii) $\delta_j \geq \frac{s_j}{C - (1-c)s(G_{j-1})}\Big(f(\text{Opt}) - \sum_{m=1}^{j-1}\delta_m\Big)$.

The following theorem bounds the value of every prefix of the greedy sequence $f(G_j)$ in terms of $f(\text{Opt})$. For the proof, we use inductive arguments together with Lemma 3; see Appendix C.

▶ **Theorem 4.** *For all $j \in \{1, \dots, k+1\}$ we have $f(G_j) \geq \frac{1}{c}\Big(1 - \exp\big(-c\frac{s(G_j)}{C}\big)\Big) f(\text{Opt})$.*

For $j = k+1$ we can derive from Theorem 4 that

$$f(\text{AG}) \geq \frac{1}{2} f(G_{k+1}) \geq \frac{1}{2c}\Big(1 - e^{-c}\Big) f(\text{Opt}),$$

but we can improve the robustness factor with the ideas Wolsey [22] uses for MGreedy in the general submodular case. Specifically, we obtain the following approximation guarantee. The main ideas of the proof are similar to that in Wolsey [22]; see Appendix D for the proof.

▶ **Theorem 5.** *Let $c \in (0, 1]$. For AGreedy we have*

$$f(AG) \geq \frac{1-x}{2-(2-c)x} f(\text{Opt}),$$

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

The result of Theorem 5 coincides with the known robustness factors of MGreedy for the cases $c = 0$ and $c = 1$. For the limit $c \to 0$ we have

$$\lim_{c \to 0} \frac{1}{c}\left(1 - e^{-cz}\right) = z,$$

and the equation simplifies to $z = 1/2$ which is the known robustness factor for the modular case; see, e.g., the textbook by Korte and Vygen [11]. For the other extreme case of $c = 1$ the equation simplifies to $1 - e^{-z} = \frac{1-z}{2-z}$, which was shown by Wolsey [22].

4 Submodular Knapsack Problem with Unknown Capacity

In this section we introduce an algorithm that generates a policy which is always at least as good as AGreedy even though it does not know the capacity of the knapsack. For that purpose we introduce indispensable items in the first part of this section. They are defined similar to swap items defined by Disser et al. [5], which they used to achieve their 1/2-optimal policy for the linear case.

As discussed by Kawase et al. [9], one major challenge when going from the case of a linear objective function to a submodular objective function is that the greedy order of items depends on the capacity of the knapsack. When an item is packed into the knapsack then other items that have a large overlap in terms of the objective with the packed item decrease in density. On the other hand, for another capacity where the first item is not packed since it does not fit they remain attractive. This issue makes it difficult to compare the outcome of a packing policy that does not know the capacity with the outcome of the MGreedy algorithm as it was done in Disser et al. [5].

Kawase et al. [9] overcome this issue by introducing the concept of the *single-valuable items* i with the property $f(\{i\}) \geq 2f(\textsc{Opt}(s(i)/2))$, i.e., Kawase et al. do not compare items with the greedy solution at all and instead compare the value of an item directly with Opt. In their policy, the most valuable single-valuable item that fits in the knapsack is inserted first. Afterwards, they try to insert the rest of the items in their greedy order. This deterministic policy achieves a robustness factor of $2(1 - 1/e)/21 \approx 0.06$.

We use another idea to overcome the capacity-dependency of the greedy order. We define the concept of an *indispensable item*. These are items that the alternative greedy algorithm AGreedy returns instead of the greedy solution. It turns out that the performance of this algorithm can be also obtained by a policy that does not know the capacity.

4.1 Indispensable Items

▶ **Definition 6.** *An item $i \in N$ is called* indispensable *if there exists a capacity $C \in \mathbb{R}_{>0}$ for which AGreedy returns $\{i\} = \{i_{k+1}\}$ instead of the greedy solution G_k.*

We say that an item $i \in N$ is indispensable for capacity $C \in \mathbb{R}_{\geq s(i)}$ if AGreedy returns $\{i\} = \{i_{k+1}\}$ instead of the greedy solution G_k for capacity C.

For ease of exposition, we assume in the following that there are no ties when an algorithm compares items by value, differences in value, or density. In practice this could be achieved by small perturbations of the values, or by using a lexicographic order that breaks ties in a systematic way. However, to avoid heavy notation, we assume that ties do not exist.

For a capacity $C \in \mathbb{R}_{>0}$, let $G_C = (i_1, i_2, \dots i_{n_C})$ be the greedy order of all items in $N_C = \{i \in N \mid s(i) \leq C\}$ with $n_C = |N_C|$. The following lemma contains important properties of indispensable items that are key to our definition of robust policies.

▶ **Lemma 7.** *Let item $i \in N$ be an indispensable item for capacity $C \in \mathbb{R}_{\geq s(i)}$ and let $G_C = (i_1, i_2, \dots i_{n_C})$ be the greedy order for the given capacity with $i = i_{k+1}$ for $k \in \{0, \dots, n_C - 1\}$. Then the following properties hold:*

(i) *$k \geq 1$ and $s_{k+1} > \sum_{j=1}^{k} s_j$.*

(ii) *i is and only is an indispensable item for capacities in the interval $[C_1, C_2[$, with*

$$C_1 = s_{k+1},$$
$$C_2 = \min\Big\{ s(\{i_1, \dots, i_{k+1}\}), \min\big\{\tilde{C} > C \mid \text{the first } k+1 \text{ items in } G_{\tilde{C}} \text{ are not the first } k+1 \text{ items of the greedy-order } G_{C_1}\big\}\Big\}.$$

(iii) *Let $\tilde{C}$ be the smallest capacity larger than s_{k+1}, such that the first $k+1$ items in $G_{\tilde{C}}$ are not the first $k+1$ items in $G_{s_{k+1}}$. Then the first item in $G_{\tilde{C}}$ that is larger than s_{k+1} is either the first item in $G_{\tilde{C}}$ or an indispensable item for capacity $\tilde{C}$.*

Proof. In the following we denote the indispensable item i by i_{k+1}.

We start to show (i). Obviously, i_1 cannot be an indispensable item for capacity C by definition. Therefore, $1 \leq k \leq n_C - 1$. Since i_{k+1} is an indispensable item, we have for all $j \in \{1, 2, \dots, k\}$ that

$$f(G_k) < f(G_k \cup \{i_{k+1}\}) - f(G_k) \leq f(G_j \cup \{i_{k+1}\}) - f(G_j). \tag{4}$$

Additionally, we know for every greedy order that for all $j \in \{1, 2, \dots, k\}$ it holds that

$$\frac{f(G_{j-1} \cup \{i_j\}) - f(G_{j-1})}{s_j} \geq \frac{f(G_{j-1} \cup \{i_{k+1}\}) - f(G_{j-1})}{s_{k+1}}. \tag{5}$$

Furthermore, we have

$$f(G_k) = \sum_{j=1}^{k} f(G_{j-1} \cup \{i_j\}) - f(G_{j-1}) \geq \sum_{j=1}^{k} \frac{s_j}{s_{k+1}} (f(G_{j-1} \cup \{i_{k+1}\}) - f(G_{j-1}))$$

where the first sum is telescopic and then we used inequality (5). We obtain

$$s_{k+1} \geq \sum_{j=1}^{k} s_j \frac{f(G_{j-1} \cup \{i_{k+1}\}) - f(G_{j-1})}{f(G_k)} > \sum_{j=1}^{k} s_j$$

by inequality (4).

We next show (ii). Let C be a capacity for which i_{k+1} is an indispensable item. Obviously, $C \geq C_1 = s_{k+1}$. By (i) we know that the size of every item $i_j, j \in \{1, \dots, k\}$ is smaller than the size of i_{k+1}. Therefore, we have that the first k items of the greedy order G_C are identical to the first k items of the greedy order $G_{\hat{C}}$ for all capacities $\hat{C} \in [C_1, C]$ and thus, i_{k+1} is an indispensable item for all those capacities.

For all capacities $\hat{C} > C$ we have that i_{k+1} is an indispensable item until there is a capacity for which the first $k+1$ items of the greedy order change or i_{k+1} does no longer exceed the capacity, which is the case for $s(i_1, \dots, i_{k+1})$, if the greedy order does not change. Therefore, i_{k+1} is an indispensable item for all capacities $\hat{C} \in [C, C_2[$.

Assume that i_{k+1} is an indispensable item for a capacity higher or equal to C_2 and therefore, it is the first item in the greedy order that exceeds the capacity. Then we have that the first $k+1$ items of the greedy order have to have changed in comparison to G_C. But then, there is an item in front of i_{k+1} in the greedy order, which is larger than i_{k+1}. That contradicts property (i) for indispensable items.

Algorithm 1 Determine indispensable items.

```
1: procedure isIndispensable(N, i*)
2:     G_0 ← ∅, j ← 1
3:     U ← {i ∈ N | s(i) ≤ s(i*)}
4:     while s(G_{j-1}) ≤ s(i*) do
5:         i_j ← arg max_{i∈U} { (f(G_{j-1} ∪ {i}) − f(G_{j-1})) / s(i) }
6:         if i_j = i* then
7:             if j ≥ 2 and f(G_{j-1} ∪ {i_j}) − f(G_{j-1}) > f(G_{j-1}) then
8:                 return (True, G_{j-1})
9:             else
10:                return (False, ∅)
11:        else
12:            G_j ← G_{j-1} ∪ {i_j}
13:            U ← U \ {i_j}
14:            j ← j + 1
15:    return (False, ∅)
```

Finally, we show (iii). Let $G_{\tilde{C}}$ be the greedy order for capacity $\tilde{C}$ and let $\tilde{G}_j$ denote the first j items in $G_{\tilde{C}}$. Further let $\tilde{\imath}_{\tilde{k}+1}$ be the first item in $G_{\tilde{C}}$ that has a larger size than i_{k+1}. It holds $\tilde{k} \leq k$, $G_j = \tilde{G}_j$ for all $j \in \{1, \dots, \tilde{k}\}$ and $s(\tilde{\imath}_{\tilde{k}+1}) = \tilde{C}$. We proceed to prove that $\tilde{\imath}_{\tilde{k}+1}$ is an indispensable item for $\tilde{C}$, if $\tilde{k} \geq 1$. Since $s(\tilde{G}_{\tilde{k}}) = s(G_{\tilde{k}}) < s_{k+1} < s(\tilde{\imath}_{\tilde{k}+1})$ we have that $\tilde{\imath}_{\tilde{k}}$ is the first item in the greedy order $G_{\tilde{C}}$ that exceeds the capacity. Additionally, we have

$$\begin{aligned} f(\tilde{G}_{\tilde{k}} \cup \{\tilde{\imath}_{\tilde{k}+1}\}) - f(\tilde{G}_{\tilde{k}}) &> f(\tilde{G}_{\tilde{k}} \cup \{i_{k+1}\}) - f(\tilde{G}_{\tilde{k}}) \\ &\geq f(G_k \cup \{i_{k+1}\}) - f(G_k) > f(G_k) \geq f(\tilde{G}_{\tilde{k}}). \end{aligned}$$

The first inequality holds, since $\tilde{\imath}_{\tilde{k}+1}$ is in front of i_{k+1} in the greedy order $G_{\tilde{C}}$ and $s(\tilde{\imath}_{\tilde{k}+1}) > s(i_{k+1})$. Second and last inequality follow from submodularity and the fact that $\tilde{G}_{\tilde{k}} = G_{\tilde{k}} \subseteq G_k$. The third inequality holds, because i_{k+1} is an indispensable item. ◀

With Lemma 7 (ii) we can determine if an item i is an indispensable item by checking if it is indispensable for the capacity $s(i)$. Such a procedure is given in Algorithm 1. The algorithm builds the greedy order $G_{s(i)}$ until the first item exceeds the capacity or item i is chosen. Only if i is the item that exceeds the capacity and fulfills the condition $f(G_{j-1} \cup \{i\}) - f(G_{j-1}) > f(G_{j-1})$, the algorithm returns `True` together with the greedy items in front of i, which will be helpful later on.

4.2 Robust Policy

The general idea of the adaptive policy is to choose a reasonable start item based on Lemma 7 (iii). We build a list of those items and then start to build our solution with the biggest item of the list that fits in the knapsack. Note that AGreedy always returns the greedy solution for all capacities smaller than the size of the smallest indispensable item, thus we start the list with the smallest indispensable item and only add larger items to the list. By Lemma 7 (iii) we have for an indispensable item i_{k+1}, that it is part of the output of AGreedy until there is a capacity for which there is a larger indispensable item or there is a new first item in the greedy order for this capacity. Therefore, we add exactly those items to the list.

Algorithm 2 List of start items.

```
1:  procedure StartItemList(N)
2:      i_1, ..., i_|N| ← < items sorted non-decreasingly by size >
3:      F ← [], j ← 1
4:      while j ≤ |N| do
5:          U ← {i ∈ N | s(i) ≤ s(i_j)}
6:          i ← arg max { f({i})/s(i) | i ∈ U }
7:          (isIndis, G) ← isIndispensable(N, i)
8:          if isIndis = True then
9:              F ← [i] + F
10:         else if i = i_1 and F ≠ [] then
11:             F ← [i] + F
12:         j ← j + 1
13:     return F
```

We create a list of start items as follows. It starts by ordering the items non-decreasingly by size. For every item i it is checked if the item is indispensable or if it is the first item in the greedy order for capacity $s(i)$. Indispensable items are always added to the list and items, which are first in the greedy order, are only added if there is already an item in the list. Thus, the smallest indispensable item is the first item added to the list. Note that it is not possible that two items of the same size are added to the list. Since the algorithm always adds an item to the start of the list, the size of items in the list is strictly decreasing. For a formal description; see Algorithm 2.

Finally, Algorithm 3 generates an adaptive policy for the submodular knapsack problem with unknown capacity. The algorithm consists of three main steps.

Step 1. If possible, insert the largest item from the list of starting items F that fits in the knapsack. Let this item be i_{k+1}.

Step 2. If i_{k+1} is an indispensable item, try to insert the first k items of the greedy order $G_{s(i_{k+1})}$.

Step 3. Try to pack all other items in their greedy order.

If AGreedy returns an indispensable item, Step 1 guarantees that this indispensable item is also in the solution returned by Algorithm 3. For the case, when an indispensable item i_{k+1} is added to the solution in Step 1, but the capacity is higher or equal to $s(\{i_1, \dots, i_{k+1}\})$, such that i_{k+1} is not an indispensable for this capacity, we want to add the items $\{i_1, \dots, i_k\}$, contained in G_k to our solution (Step 2). In Step 3 we complete the solution. We try to add items to the knapsack in their greedy order. We will show in Theorem 8 that this policy generated by Algorithm 3 is always as good as AGreedy.

▶ **Theorem 8.** *For a capacity $C \in \mathbb{R}_{>0}$ let $\Pi(C)$ be the policy generated from Algorithm 3 and let $AG(C)$ be the output of AGreedy. Then, we have $f(\Pi(C)) \geq f(AG(C))$ for every capacity $C \in \mathbb{R}_{>0}$.*

Proof. Let F be the list of starting items created by Algorithm 2. If there are no indispensable items in N, then F and G_k are empty in Algorithm 3 and the algorithm tries in Step 3 to insert all items in their greedy order. Furthermore, AGreedy always returns the greedy solution in this case. Assume that Algorithm 3 tries to pack an item in the knapsack that is not part of the greedy solution returned by AGreedy, before the algorithm has packed all items of the greedy solution. Since this item was not considered by AGreedy, it has to be larger than the knapsack capacity and thus, cannot be inserted by Algorithm 3. Therefore, we obtain $f(\Pi(C)) \geq f(\mathrm{AG}(C))$ for all capacities $C \in \mathbb{R}_{>0}$ in this case.

Algorithm 3 Robust policy.

$S \leftarrow \emptyset, G \leftarrow \emptyset$
$F \leftarrow$ StartItemList(N)
$j \leftarrow 1$
while $S = \emptyset$ **and** $j \leq \text{len}(F)$ **do**
    $i_{k+1} \leftarrow F[j]$
    if i_{k+1} fits in the knapsack **then**
        $(\sim, G_k) \leftarrow$ isIndispensable(N, i_{k+1})
        $S = \{i_{k+1}\}$
    else
        $N \leftarrow \{i \in N \mid s(i) < s(i_{k+1})\}$
    $j \leftarrow j + 1$
for $i \in G_k$ **do**
    if $S \cup \{i\}$ fits in the knapsack **then**
        $S \leftarrow S \cup \{i\}$
    $N \leftarrow N \setminus \{i\}$
while $N \neq \emptyset$ **do**
    $i_{\max} \leftarrow \arg\max\{\frac{f(S\cup\{i\})-f(S)}{s(i)} \mid i \in N\}$
    if $S \cup \{i_{\max}\}$ fits into the knapsack **then**
        $S \leftarrow S \cup \{i_{\max}\}$
        $N \leftarrow N \setminus \{i_{\max}\}$
    else
        $N \leftarrow \{i \in N \mid s(i) < s(i_{\max})\}$
return S

Now we assume there is at least one indispensable item, implying that F is not empty. Let $f_1, \ldots, f_{|F|}$ be the items in F sorted increasingly by size. Consider the partition of all capacities in the intervals $[I_j, I_{j+1})$ for $j \in \{0, \ldots, |F|\}$, with $I_0 = \min\{s(i) \mid i \in N\}$, $I_{|F|+1} = s(N)$ and $I_j = s(f_j)$ for $j \in \{1 \ldots, |F|\}$.

First, we consider all capacities $C \in [I_0, I_1)$. Since, the capacities are smaller than any item in F, Algorithm 3 inserts no item from F and G_k remains empty. Therefore, Algorithm 3 again tries to insert all items by their greedy order and AGreedy returns the greedy solution for all those capacities, because all indispensable items have a larger size. We have $f(\Pi(C)) \geq f(\text{AG}(C))$ by the same argumentation as in the case where F is empty.

Second, we consider all capacities $C \in [I_j, I_{j+1})$ for an arbitrary $j \in \{1 \ldots, |F|\}$. Note that Algorithm 3 inserts the item f_j from F for all those capacities as the first item. We distinguish the cases where f_j is an indispensable item and where f_j is not an indispensable item. In the following we denote item f_j as i_{k+1}.

Case 1: i_{k+1} is an indispensable item. By Lemma 7 (ii) we have that i_{k+1} is packed by AGreedy instead of the greedy solution G_k for all capacities $C \in [C_1, C_2)$. For all capacities $C \in [C_1, C_2) \cap [I_j, I_{j+1})$ we have $f(\Pi(C)) \geq f(\text{AG}(C))$, since Algorithm 3 inserts the same indispensable item in step 1. Since $C_1 = I_j$, there are only two more cases to distinguish: $C \in [C_2, I_{j+1})$ if $C_2 < I_{j+1}$ and $C \in [I_{j+1}, C_2)$ if $C_2 > I_{j+1}$.

Subcase 1.1: $C \in [C_2, I_{j+1})$ if $C_2 < I_{j+1}$ Note that we have $C_2 = s(\{i_1, \dots, i_{k+1}\})$, because $C_2 < s(i_1, \dots, i_{k+1})$ implies that an item with size C_2 changed the greedy order of the first $k+1$ items. Such an item would have been added to F, since it is an indispensable item or the first item in the greedy order by Lemma 7 (iii), but then $C_2 = I_{j+1}$.

For all capacities $C \in [C_2, I_{j+1})$ we know by Lemma 7 (iii) that the first $k+1$ items in the greedy order G_C are still identical to the first $k+1$ items of $G_{s_{k+1}}$. We also know that there is no indispensable item for any capacity $C \in [C_2, I_{j+1})$. Therefore, AGreedy returns the greedy solution. After Algorithm 3 packed the indispensable item i_{k+1}, the algorithm continues in step 2 to insert the first k greedy items of the greedy order $G_{s(i_{k+1})}$. We already discussed that all these items fit in the knapsack, since $C \geq C_2 = s(\{i_1, \dots, i_{k+1}\})$ and that they are also the first $k+1$ items packed by AGreedy.

After the first $k+1$ items, AGreedy continues to pack more items greedily until the first item exceeds the capacity. Algorithm 3 also continues in step 3 to pack more items greedily and thus we have $f(\Pi(C)) \geq f(\mathrm{AG}(C))$ for all capacities $C \in [C_2, I_{j+1})$.

Subcase 1.2: $C \in [I_{j+1}, C_2)$ if $C_2 > I_{j+1}$ We will show that this case cannot occur. Consider item $f_{j+1} \in F$ with $s(f_{j+1}) = I_{j+1}$. It is either an indispensable item for capacity I_{j+1} or the first item in the greedy order $G_{I_{j+1}}$.

By Lemma 7 (ii) we know, that f_{j+1} is not allowed to change the first $k+1$ items in the greedy order $G_{s_{k+1}}$, since then we would have $C_2 = I_{j+1}$. Thus, f_{j+1} cannot be the first item in the greedy order $G_{I_{j+1}}$ and as an indispensable item for capacity I_{j+1} it has to appear behind the first k+1 items of $G_{s_{k+1}}$. But this also gives a contradiction, since

$$s(f_{j+1}) > \sum_{j=1}^{k+1} s_j \geq C_2 > I_{j+1} = s(f_{j+1}). \tag{6}$$

The first inequality follows from Lemma 7 (i) and the second inequality from Lemma 7 (ii).

Case 2: i_{k+1} is not an indispensable item Then, i_{k+1} is the first item in the greedy order $G_{s(i_{k+1})}$ and it is the first item in the greedy order G_C for every capacity $C \in [I_j, I_{j+1})$ too, since a new first item would have been added to F. Additionally, we know that there is no indispensable item for all capacities $C \in [I_j, I_{j+1})$, since it also would have been added to F. Therefore, AGreedy returns the greedy solution for all capacities $C \in [I_j, I_{j+1})$ and the first item packed by Algorithm 3 is always the first item of the greedy solution returned by AGreedy. In step 2 of Algorithm 3 no items are added to the knapsack, because G_k is empty, if i_{k+1} is not an indispensable item. Then, Algorithm 3 tries to insert items exactly by their greedy order in step 3 and this yields $f(\Pi(C)) \geq f(\mathrm{AG}(C))$ for all capacities $C \in [I_j, I_{j+1})$.

Thus, the statement holds for all capacities in the intervals $[I_j, I_{j+1})$, $j \in \{0, \dots, |F|\}$. For capacities smaller than I_0 no item can be packed and for capacities greater than $I_{|F|+1}$ every item can be packed in the knapsack. This completes the proof. ◀

From Theorem 5 and Theorem 8 we obtain the main result of this paper.

▶ **Theorem 9.** *The policy Π generated by Algorithm 3 has a robustness factor of $\alpha = \frac{1-x}{2-(2-c)x}$ where x is the unique root in $[0,1]$ of the equation $\frac{1}{c}\left(1 - e^{-cz}\right) = \frac{1-z}{2-(2-c)z}$.*

References

1 Aaron Bernstein, Yann Disser, Martin Groß, and Sandra Himburg. General bounds for incremental maximization. *Math. Program.*, 2020. doi:10.1007/s10107-020-01576-0.

2 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

3 Michele Conforti and Gérard Cornuéjols. Submodular set functions, matroids and the greedy algorithm: Tight worst-case bounds and some generalizations of the Rado-Edmonds theorem. *Discret. Appl. Math.*, 7(3):251–274, 1984. doi:10.1016/0166-218X(84)90003-9.

4 Gerard Cornuéjols, Marshall L. Fisher, and George L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Sci.*, 23:789–810, 1977. doi:10.1287/mnsc.23.8.789.

5 Yann Disser, Max Klimm, Nicole Megow, and Sebastian Stiller. Packing a knapsack of unknown capacity. *SIAM J. Discret. Math.*, 31(3):1477–1497, 2017. doi:10.1137/16M1070049.

6 Yann Disser, Max Klimm, and David Weckbecker. Fractionally subadditive maximization under an incremental knapsack constraint. In Jochen Könemann and Britta Peis, editors, *Approximation and Online Algorithms - 19th International Workshop (WAOA)*, volume 12982 of *Lecture Notes in Computer Science*, pages 206–223. Springer, 2021. doi:10.1007/978-3-030-92702-8_13.

7 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

8 Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. *An analysis of approximations for maximizing submodular set functions - II*, volume 8 of *Mathematical Programming Studies*, page 73–87. Springer, Berlin, Heidelberg, 1978. doi:10.1007/BFb0121195.

9 Yasushi Kawase, Hanna Sumita, and Takuro Fukunaga. Submodular maximization with uncertain knapsack capacity. *SIAM J. Discret. Math.*, 33(3):1121–1145, 2019. doi:10.1137/18M1174428.

10 David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory Comput.*, 11:105–147, 2015. doi:10.4086/toc.2015.v011a004.

11 Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer, Heidelberg, Berlin, 2018.

12 Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM Trans. Intell. Syst. Technol.*, 2(4):32:1–32:20, 2011. doi:10.1145/1989734.1989736.

13 Andreas Krause, Ajit Paul Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *J. Mach. Learn. Res.*, 9:235–284, 2008. URL: https://dl.acm.org/citation.cfm?id=1390689.

14 Jon Lee. Constrained maximum-entropy sampling. *Oper. Res.*, 46(5):655–664, 1998. doi:10.1287/opre.46.5.655.

15 Nicole Megow and Julián Mestre. Instance-sensitive robustness guarantees for sequencing with unknown packing and covering constraints. In Robert D. Kleinberg, editor, *Innovations in Theoretical Computer Science (ITCS)*, pages 495–504. ACM, 2013. doi:10.1145/2422436.2422490.

16 Alfredo Navarra and Cristina M. Pinotti. Online knapsack of unknown capacity: How to optimize energy consumption in smartphones. *Theor. Comput. Sci.*, 697:98–109, 2017. doi:10.1016/j.tcs.2017.07.029.

17 George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978. doi:10.1287/moor.3.3.177.

18 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978. doi:10.1007/BF01588971.

19 Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004. doi:10.1016/S0167-6377(03)00062-2.

20 Maxim Sviridenko, Jan Vondrák, and Justin Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. *Math. Oper. Res.*, 42(4):1197–1218, 2017. doi:10.1287/moor.2016.0842.

21 Jan Vondrák. Submodularity and curvature: The optimal algorithm. *RIMS Kôkyûroku Bessatsu*, B23:253–266, 2010.

22 Laurence A. Wolsey. Maximising real-valued submodular functions: Primal and dual heuristics for location problems. *Math. Oper. Res.*, 7(3):410–425, 1982. doi:10.1287/moor.7.3.410.

23 Yuichi Yoshida. Maximizing a monotone submodular function with a bounded curvature under a knapsack constraint. *SIAM J. Discret. Math.*, 33(3):1452–1471, 2019. doi:10.1137/16M1107644.

A Proof of Lemma 1

Proof. We first show (i). Let $A \subset N$ and $j \in N \setminus A$ be arbitrary. We calculate

$$1 - c = \min_{i \in N} \frac{f(N) - f(N \setminus \{i\})}{f(\{i\})} \leq \frac{f(N) - f(N \setminus \{j\})}{f(\{j\})} \leq \frac{f(A \cup \{j\}) - f(A)}{f(\{j\})}$$

where for the equation, we used the definition of curvature, and for the last equation, we used submodularity.

To show (ii), we successively apply (i) on the elements in B. ◀

B Proof of Lemma 3

Proof. Using Lemma 1 (ii) with $A = \textsc{Opt}$ and $B = G_{j-1} \setminus \textsc{Opt}$, we obtain

$$f(\textsc{Opt}) + (1 - c) \sum_{i \in G_{j-1} \setminus \textsc{Opt}} f(\{i\}) - f(G_{j-1}) \leq f(\textsc{Opt} \cup G_{j-1}) - f(G_{j-1}). \tag{7}$$

We proceed to bound the term $f(\textsc{Opt} \cup G_{j-1}) - f(G_{j-1})$ that appears on the right hand side of (7) via

$$\begin{aligned} f(\textsc{Opt} \cup G_{j-1}) - f(G_{j-1}) &\leq \sum_{i \in \textsc{Opt} \setminus G_{j-1}} f(G_{j-1} \cup \{i\}) - f(G_{j-1}) \\ &= \sum_{i \in \textsc{Opt} \setminus G_{j-1}} s(i) \frac{f(G_{j-1} \cup \{i\}) - f(G_{j-1})}{s(i)} \\ &\leq \left(\sum_{i \in \textsc{Opt} \setminus G_{j-1}} s(i) \right) \frac{f(G_{j-1} \cup \{i_j\}) - f(G_{j-1})}{s_j} \\ &= s(\textsc{Opt} \setminus G_{j-1}) \frac{f(G_j) - f(G_{j-1})}{s_j} = s(\textsc{Opt} \setminus G_{j-1}) \frac{\delta_j}{s_j}, \end{aligned}$$

where we used (3) and the definition of the greedy sequence. To show (i), we bound the term $(1-c)\sum_{i\in G_{j-1}\setminus \textsc{Opt}} f(\{i\}) - f(G_{j-1})$ on the left hand side of (7) by

$$\begin{aligned}(1-c)\sum_{i\in G_{j-1}\setminus \textsc{Opt}} f(\{i\}) - f(G_{j-1}) &\geq (1-c)\sum_{m=1}^{j-1}(1-\chi_m)\delta_m - f(G_{j-1})\\ &= (1-c)\sum_{m=1}^{j-1}(1-\chi_m)\delta_m - \sum_{m=1}^{j-1}\delta_m\\ &= -(1-c)\sum_{m=1}^{j-1}\chi_m\delta_m - c\sum_{m=1}^{j-1}\delta_m,\end{aligned}$$

where we used submodularity for the inequality. Both bounds applied to inequality (7) yield

$$\begin{aligned}s(\textsc{Opt}\setminus G_{j-1})\frac{\delta_j}{s_j} &\geq f(\textsc{Opt}) - (1-c)\sum_{m=1}^{j-1}\chi_m\delta_m - c\sum_{m=1}^{j-1}\delta_m\\ &= c\Big(f(\textsc{Opt}) - \sum_{m=1}^{j-1}\delta_m\Big) + (1-c)\Big(f(\textsc{Opt}) - \sum_{m=1}^{j-1}\chi_m\delta_m\Big).\end{aligned}$$

Multiplication with $\frac{s_j}{s(\textsc{Opt}\setminus G_{j-1})}$ and applying $s(\textsc{Opt}\setminus G_{j-1}) \leq C - S_{j-1} \leq C$ completes the proof of (i).

To prove (ii), we bound $\sum_{i\in G_{j-1}\setminus \textsc{Opt}} f(\{i\})$ in a different way by

$$\begin{aligned}\sum_{i\in G_{j-1}\setminus \textsc{Opt}} f(\{i\}) &\geq \sum_{m=1}^{j-1}(1-\chi_m)\delta_m\\ &= \sum_{m=1}^{j-1}(1-\chi_m)s_m\frac{\delta_m}{s_m}\\ &\geq \Big(\sum_{m=1}^{j-1}(1-\chi_m)s_m\Big)\frac{\delta_j}{s_j} = s(G_{j-1}\setminus \textsc{Opt})\frac{\delta_j}{s_j},\end{aligned}$$

where for the first inequality we used submodularity and for the second inequality we used a property of the greedy sequence. Applied to inequality (7) together with the first bound yields

$$\begin{aligned}f(\textsc{Opt}) - f(G_{j-1}) &\leq s(\textsc{Opt}\setminus G_{j-1})\frac{\delta_j}{s_j} - (1-c)s(G_{j-1}\setminus \textsc{Opt})\frac{\delta_j}{s_j}\\ &= \Big(s(\textsc{Opt}\setminus G_{j-1}) - (1-c)s(G_{j-1}\setminus \textsc{Opt})\Big)\frac{\delta_j}{s_j}\\ &\leq \Big(s(\textsc{Opt}) - (1-c)s(G_{j-1})\Big)\frac{\delta_j}{s_j}\\ &\leq \Big(C - (1-c)s(G_{j-1})\Big)\frac{\delta_j}{s_j},\end{aligned}$$

as claimed. ◀

C Proof of Theorem 4

Proof. Applying the bound of Lemma 3 (i) for all j, we obtain the following lower bound on G_j.

▷ Claim 10. For all $j \in \{1,\dots,k+1\}$ and for all $l \in \{0,\dots,j\}$ we have

$$f(G_j) \geq f(\textsc{Opt}) - \left(\prod_{m=l+1}^{j}\left(1-\frac{cs_m}{C}\right)\right)\left(f(\textsc{Opt}) - \sum_{m=1}^{l}\delta_m\right) + \frac{1-c}{c}\frac{C}{C-S_l}\left(1-\prod_{m=l+1}^{j}\left(1-\frac{cs_m}{C}\right)\right)\left(f(\textsc{Opt}) - \sum_{m=1}^{l}\chi_m\delta_m\right). \tag{8}$$

For a fixed $j \in \{1,\dots,k+1\}$ we proof the claim by induction over l starting with $l = j$ and going down by one in the induction step.

For $l = j$ both products in (8) are equal to 1 and the right side simplifies to $\sum_{m=1}^{j}\delta_m = f(G_j)$. For the induction step assume that the statement of the claim holds for $l \in \{1,\dots,j\}$ and consider the statement for $l-1$. To shorten the notation we set

$$P_t := \prod_{m=t}^{j}\left(1-\frac{cs_m}{C}\right),$$

for $t = 1,\dots,j+1$. The induction hypothesis is

$$f(G_j) \geq f(\textsc{Opt}) - P_{l+1}\left(f(\textsc{Opt}) - \sum_{m=1}^{l}\delta_m\right) + \frac{1-c}{c}\frac{C}{C-S_l}\left(1-P_{l+1}\right)\left(f(\textsc{Opt}) - \sum_{m=1}^{l}\chi_m\delta_m\right).$$

To apply Lemma 3 (i) for $j = l$, we rearrange δ_l terms. Those terms read

$$\left(P_{l+1} - \chi_l\frac{1-c}{c}\frac{C}{C-S_l}\left(1-P_{l+1}\right)\right)\delta_l.$$

We can transform the factor of δ_l to see that it is greater or equal to zero. Note that the factor $1/c$ vanishes in $1-P_{l+1}$.

$$\begin{aligned} &P_{l+1} - \chi_l\frac{1-c}{c}\frac{C}{C-S_l}\left(1-P_{l+1}\right)\\ &= (1-c) + c\left(1-\frac{1}{c}\left(1-P_{l+1}\right)\right) - \chi_l(1-c)\frac{C}{C-S_l}\frac{1}{c}\left(1-P_{l+1}\right)\\ &= c\left(1-\frac{1}{c}\left(1-P_{l+1}\right)\right) + (1-c)\left(1-\chi_l\frac{C}{C-S_l}\frac{1}{c}\left(1-P_{l+1}\right)\right) \geq 0. \end{aligned}$$

Applying

$$\delta_l \geq \frac{cs_l}{C}\left(f(\textsc{Opt}) - \sum_{m=1}^{l-1}\delta_m\right) + \frac{(1-c)s_l}{C-S_{l-1}}\left(f(\textsc{Opt}) - \sum_{m=1}^{l-1}\chi_m\delta_m\right),$$

yields

$$f(G_j) \geq f(\textsc{Opt}) - \alpha\left(f(\textsc{Opt}) - \sum_{m=1}^{l-1}\delta_m\right) + \beta\left(f(\textsc{Opt}) - \sum_{m=1}^{l-1}\chi_m\delta_m\right),$$

with

$$\begin{aligned}\alpha &= P_{l+1} - \frac{cs_l}{C}\Big(P_{l+1} - \chi_l \frac{1-c}{c}\frac{C}{C-S_l}\Big(1-P_{l+1}\Big)\Big)\\ &= \Big(1-\frac{cs_l}{C}\Big)P_{l+1} + \chi_l(1-c)\frac{s_l}{C-S_l}\Big(1-P_{l+1}\Big)\Big) = P_l + \chi_l(1-c)\frac{s_l}{C-S_l}\Big(1-P_{l+1}\Big)\Big),\end{aligned}$$

and

$$\begin{aligned}\beta &= \frac{1-c}{c}\frac{C}{C-S_l}\Big(1-P_{l+1}\Big) + \frac{(1-c)s_l}{C-S_{l-1}}\Big(P_{l+1} - \chi_l\frac{1-c}{c}\frac{C}{C-S_l}\Big(1-P_{l+1}\Big)\Big)\\ &= \frac{1-c}{c}\frac{C}{C-S_l}\Big(1-\chi_l\frac{(1-c)s_l}{C-S_{l-1}}\Big)\Big(1-P_{l+1}\Big) + \frac{(1-c)s_l}{C-S_{l-1}}P_{l+1}\\ &= \frac{1-c}{c}\frac{C}{C-S_l}\Big(\frac{C-S_{l-1}-\chi_l s_l}{C-S_{l-1}} + \chi_l\frac{cs_l}{C-S_{l-1}}\Big)\Big(1-P_{l+1}\Big) + \frac{(1-c)s_l}{C-S_{l-1}}P_{l+1}\\ &= \frac{1-c}{c}\frac{C}{C-S_{l-1}}\Big(1-P_{l+1}\Big) + \chi_l\frac{1-c}{c}\frac{C}{C-S_l}\frac{cs_l}{C-S_{l-1}}\Big(1-P_{l+1}\Big) + \frac{(1-c)s_l}{C-S_{l-1}}P_{l+1}\\ &= \frac{1-c}{c}\frac{C}{C-S_{l-1}}\Big(1-(1-\frac{cs_l}{C})P_{l+1}\Big) + \chi_l(1-c)\frac{C}{C-S_l}\frac{s_l}{C-S_{l-1}}\Big(1-P_{l+1}\Big)\\ &= \frac{1-c}{c}\frac{C}{C-S_{l-1}}\Big(1-P_l\Big) + \chi_l(1-c)\frac{C}{C-S_l}\frac{s_l}{C-S_{l-1}}\Big(1-P_{l+1}\Big)\\ &\geq \frac{1-c}{c}\frac{C}{C-S_{l-1}}\Big(1-P_l\Big) + \chi_l(1-c)\frac{s_l}{C-S_l}\Big(1-P_{l+1}\Big).\end{aligned}$$

Thus, we have

$$\begin{aligned}f(G_j) &\geq f(\textsc{Opt}) - P_l\Big(f(\textsc{Opt}) - \sum_{m=1}^{l-1}\delta_m\Big)\\ &\quad + \frac{1-c}{c}\frac{C}{C-S_{l-1}}\Big(1-P_l\Big)\Big(f(\textsc{Opt}) - \sum_{m=1}^{l-1}\chi_m\delta_m\Big)\\ &\quad + \chi_l(1-c)\frac{s_l}{C-S_l}\Big(1-P_{l+1}\Big)\Big(\sum_{m=1}^{l-1}\delta_m - \sum_{m=1}^{l-1}\chi_m\delta_m\Big)\\ &\geq f(\textsc{Opt}) - P_l\Big(f(\textsc{Opt}) - \sum_{m=1}^{l-1}\delta_m\Big)\\ &\quad + \frac{1-c}{c}\frac{C}{C-S_{l-1}}\Big(1-P_l\Big)\Big(f(\textsc{Opt}) - \sum_{m=1}^{l-1}\chi_m\delta_m\Big),\end{aligned}$$

as claimed.

For $j \in \{1,\dots,k+1\}$ and $l = 0$ the statement of the claim simplifies to

$$\begin{aligned}f(G_j) &\geq f(\textsc{Opt}) - \left(\prod_{m=1}^{j}\Big(1-\frac{cs_m}{C}\Big)\right)f(\textsc{Opt}) + \frac{1-c}{c}\left(1-\prod_{m=1}^{j}\Big(1-\frac{cs_m}{C}\Big)\right)f(\textsc{Opt})\\ &= \Big(1+\frac{1-c}{c}\Big)\left(1-\prod_{m=1}^{j}\Big(1-\frac{cs_m}{C}\Big)\right)f(\textsc{Opt})\\ &= \frac{1}{c}\left(1-\prod_{m=1}^{j}\Big(1-\frac{cs_m}{C}\Big)\right)f(\textsc{Opt}).\end{aligned}$$

We derive the final statement of the theorem

$$\begin{aligned}
f(G_j) &\geq \frac{1}{c}\left(1 - \prod_{m=1}^{j}\left(1 - \frac{cs_m}{C}\right)\right) f(\textsc{Opt}) \\
&\geq \frac{1}{c}\left(1 - \left(\frac{1}{j}\sum_{m=1}^{j}\left(1 - \frac{cs_m}{C}\right)\right)^j\right) f(\textsc{Opt}) \\
&= \frac{1}{c}\left(1 - \left(1 - \frac{c}{jC}\sum_{m=1}^{j} s_m\right)^j\right) f(\textsc{Opt}) \\
&= \frac{1}{c}\left(1 - \left(1 - \frac{cs(G_j)}{jC}\right)^j\right) f(\textsc{Opt}) \\
&\geq \frac{1}{c}\left(1 - \exp\left(-c\frac{s(G_j)}{C}\right)\right) f(\textsc{Opt}),
\end{aligned}$$

where we used that the geometric mean is always smaller or equal to the arithmetic mean for non-negative values and that $(1 + \frac{x}{j})^j \leq e^x$ for all $j \geq 1$ and $x \in \mathbb{R}$. ◀

D Proof of Theorem 5

Proof. We define $z := \frac{s(G_k)}{C}$. By Theorem 4 we have for $j = k$ that

$$f(G_k) \geq \frac{1}{c}\left(1 - e^{-cz}\right) f(\textsc{Opt}). \tag{9}$$

Additionally, we have by Lemma 3 (ii) for $j = k + 1$ that

$$\delta_{k+1} \geq \frac{s_{k+1}}{C - (1-c)s(G_k)}(f(\textsc{Opt}) - f(G_k)).$$

Solving for $f(\textsc{Opt})$ and replacing $s(G_k)$ by zC gives

$$f(\textsc{Opt}) \leq f(G_k) + \frac{C(1 - (1-c)z)}{s_{k+1}}\delta_{k+1}.$$

With $C < s(G_k) + s_{k+1} = zC + s_{k+1}$ we have $C < \frac{s_{k+1}}{1-z}$ and this yields

$$\begin{aligned}
f(\textsc{Opt}) &\leq f(G_k) + \frac{1 - (1-c)z}{1-z}\delta_{k+1} \\
&\leq f(\text{AG}) + \frac{1 - (1-c)z}{1-z} f(\text{AG}) = \frac{2 - (2-c)z}{1-z} f(\text{AG}),
\end{aligned}$$

where the last inequality holds in both cases of AGreedy. Together with (9) we get

$$f(\text{AG}) \geq \min_{z \in [0,1]} \max\left\{\frac{1}{c}\left(1 - e^{-cz}\right), \frac{1-z}{2-(2-c)z}\right\} f(\textsc{Opt}).$$

Note that $\frac{1}{c}\left(1 - e^{-cz}\right)$ is monotonically increasing and $\frac{1-z}{2-(2-c)z}$ is monotonically decreasing for $z \in [0,1]$ and a fixed $c \in (0,1)$ and that they have a unique intersection for $z \in [0,1]$. Therefore, the minimum is attained at this intersection. ◀