How to Pack Your Items When You Have to Buy Your Knapsack

Antonios Antoniadis[1,⋆], Chien-Chung Huang[2], Sebastian Ott[3], and José Verschae[4]

[1] Computer Science Department, University of Pittsburgh, Pittsburgh, USA
antoniosantoniadis@gmail.com
[2] Chalmers University, Göteborg, Sweden villars@gmail.com
[3] Max-Planck-Institut für Informatik, Saarbrücken, Germany ott@mpi-inf.mpg.de
[4] Departamento de Ingeniería Industrial, Universidad de Chile, Santiago, Chile
jverscha@ing.uchile.cl

Abstract. In this paper we consider a generalization of the classical knapsack problem. While in the standard setting a fixed capacity may not be exceeded by the weight of the chosen items, we replace this hard constraint by a weight-dependent cost function. The objective is to maximize the total profit of the chosen items minus the cost induced by their total weight. We study two natural classes of cost functions, namely convex and concave functions. For the concave case, we show that the problem can be solved in polynomial time; for the convex case we present an FPTAS and a 2-approximation algorithm with the running time of $\mathcal{O}(n \log n)$, where n is the number of items. Before, only a 3-approximation algorithm was known.
We note that our problem with a convex cost function is a special case of maximizing a non-monotone, possibly negative submodular function.

1 Introduction

The knapsack problem is a classical problem of combinatorial optimization [5]. In the standard setting we must choose a subset of items to fit into a given knapsack. Each item comes with a *weight* w_j and a *profit* p_j, and the goal is to maximize the total profit, under the constraint that the total weight of the chosen items should not exceed the knapsack's capacity. A natural generalization of the knapsack problem (and backed up by applications - see discussion below) is to assume that we can "customize" the knapsack in which we pack the selected items. In other words, we can pay more to buy a knapsack of a larger capacity. This motivates our problem.

Consider U to be a set of items j, each with its own positive *weight* w_j and positive *profit* p_j. Let $c(W)$ be a non-decreasing *cost function* $c(W)$, where W is the total weight of the chosen items. The function c is given

⋆ The author was supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).

by an oracle and it is not part of the input. The goal is to maximize the *net value*, which is the total profit of the chosen items minus the cost of their total weight. Precisely, we want to maximize the objective $\pi(A) = \sum_{j \in A} p_j - c(\sum_{j \in A} w_j)$ over all possible subsets $A \subseteq U$ of items.

Clearly, our problem contains the standard knapsack problem as a special case, when the cost function switches from 0 to infinity at the capacity of the knapsack.

If we allow arbitrary non-decreasing functions, the problem is inapproximable within any factor, unless P = NP. This follows by a simple reduction from the 2-partition problem [5]; see Appendix A for details. Therefore, in this work, we consider two natural special cases of cost functions, namely convex and concave functions.

Our Results. Let $n = |U|$ be the number of items. We assume that each call of the cost function takes constant time. Suppose that the cost function is convex. Then our problem is (weakly) NP-hard, since it contains the original knapsack problem as a special case.

- In Section 3 we present an $O(n \log n)$ time 2-approximation algorithm.
- In Section 4, we present an FPTAS (Fully Polynomial Time Approximation Scheme). With any $\epsilon > 0$, our FPTAS returns a solution with net value at least $1 - \epsilon$ fraction of the optimal solution, in $O(n^3/\epsilon^2)$ time. Our FPTAS is only slower than the well-known FPTAS for the original knapsack problem by a factor of $1/\epsilon$.

Suppose that the cost function is convave.

- In Section 5, we present an exact algorithm with running time $O(n \log n)$.

Applications. Applications for our problem naturally arise in many areas, such as cloud computing or connection management in wireless access points [1]. Consider for example a data center which earns a certain amount for every job it processes. When accepting more and more jobs, the workload increases and the data center must run its processors at higher speed to maintain an acceptable quality of service. This results in a higher power consumption, which typically grows convexly with the speed of the processor (cf. the cube-root rule for CMOS based processors [2]). Consequently, the data center wants to select the most profitable set of jobs, taking into account the energy costs, which depend on the total volume of accepted jobs. This directly translates into our generalized knapsack problem with a convex cost function.

The study of concave cost functions is motivated by the *economies of scale* principle [10]. This principle states that the cost per unit decreases if the total number of units increases, because efficiency increases and fixed costs are spread over more units.

Related Work. The standard knapsack problem is well known to be NP-hard, but one can approximate it within any factor greater than 1 in polynomial time [8]. Our generalized problem has been studied before under a convex cost function, by Barman et al. [1]. There the authors consider an online setting, where items arrive over time in random order and the algorithm must make an irrevocable accept/reject decision as soon as a new item arrives. They assume a convex cost function and develop an online algorithm which is constant competitive in expectation. In addition they also study the problem under several feasibility constraints. Even though their focus is on the online version, they also present a result for our offline setting, namely a greedy algorithm which achieves a 3-approximation. To the best of our knowledge, this was the best known approximation ratio for this problem so far.

We note that in the convex costs setting, our problem amounts to maximizing a non-monotone, possibly negative submodular function. Submodular function maximization in general is a very active area of research that has recently attracted much attention [3, 4, 9], and many other special cases (e.g. several graph cut problems [6, 7]) have been investigated. Not much is known for other problems where the submodular function is non-monotone and can take negative values. Our results can be regarded as a first step in understanding this type of problems.

Challenges and Techniques. A tricky aspect of our problem is that the objective, i.e. the net value, can be very small and even negative. In this situation, the design and analysis of approximation algorithms often becomes more challenging. Although the usual dynamic program by Ibarra and Kim [8] can be easily adapted for our more general case, the *rounding* technique is troublesome. The reason is that our goal is to maximize the difference between the profit and the cost. It could happen that both values are very large and almost coincide in the optimal solution, resulting in a rather small net value. Thus, a relatively small error in the rounding of profit and/or weight would cascade into a large mistake in the final outcome. The same problem prevents us from guessing the weight of the optimal solution, and then applying known algorithms for the classic knapsack problem.

Our 2-approximation algorithm for convex costs does not resort to any rounding or guessing. Informally speaking, we use the optimal solution of a relaxed version of our problem, where the items can be fractionally added into the collection, to upper bound the net value of the original (non-fractional) problem. However, this idea has to be carefully applied since the gap between the cost of a fractional solution and an integral one can be arbitrarily large. We show that this gap is bounded if the contribution (defined appropriately) of a job to the optimal solution is bounded. Turning these ideas into a greedy algorithm we can achieve the claimed result.

As just said, rounding the profits as for the usual knapsack problem might decrease the objective by an arbitrarily large factor. Therefore our FPTAS cannot rely on this technique. To overcome this difficulty, we use two crucial ideas. First, we observe that the convexity of the cost function implies the following fact: if A is an optimal subset of items, and $A' \subseteq A$, we can assume that A' has minimum weight among all subsets of $(U \setminus A) \cup A'$ with net value at least $\pi(A')$. This indicates the use of a specially crafted dynamic programming *tableau* that is indexed by the achievable net value. Secondly, instead of rounding the profits/weights, we merge several entries of the tableau into one. Intuitively, this corresponds to rounding the objective function so that it takes polynomially many different values.

2 Preliminaries and Notations

The input is given by a set U of items, each having a positive weight w_j and positive profit p_j, together with a non-decreasing cost function $c = c(W)$ that denotes the cost of selecting items with total weight W. We assume that c is given by an oracle which can be called in constant time, and that $c(0) = 0$. Our goal is to find a set $A \subseteq U$ of items that maximizes the total net value $\pi(A) = \sum_{j \in A} p_j - c(\sum_{j \in A} w_j)$. W.l.o.g. we assume the weights w_j to be positive integers (otherwise we can multiply them with their least common denominator τ and use the modified cost function $c'(W) = c(W/\tau)$).

Throughout the paper we consider a fixed optimal solution OPT. We slightly abuse notation by letting OPT denote both the set of items and the net value of this solution. Furthermore, let W_{OPT} denote the total weight of all items in OPT. More generally we use W_S to denote the total weight of all items in a given set S.

For any item j, we define the *density* d_j as the profit per unit of weight, i.e. $d_j := p_j/w_j$. Furthermore, it will sometimes be helpful to consider *fractional items*, i.e. we pick only $w < w_j$ units of an item j and assume this fraction has profit $w \cdot d_j$. Finally, we define $\delta_j^w(W) := w \cdot d_j - c(W + w) + c(W)$ for every item j and $w \geq -W$. The intuitive meaning of $\delta_j^w(W)$ is the change in the net value when adding w units of item j to a knapsack with current total weight W. Note that we also allow negative w, which then describes the change in the net value when removing $|w|$ units of item j.

It is easy to see, that the function $\delta_j^w(W)$ satisfies the following property if c is convex.

Proposition 1. *Let c be a convex function. Then $\delta_j^w(W) = w \cdot d_j - c(W + w) + c(W)$ is non-increasing in W, and $\delta_j^{-w}(W) = -w \cdot d_j - c(W - w) + c(W)$ is non-decreasing in W, for any item j and $w \geq 0$.*

3 2-Approximation for Convex Costs

In this section we present a 2-approximation with running time $\mathcal{O}(n \log n)$ for the case of a convex cost function c. The main idea of the algorithm is as follows. We try every item individually and compare their net values to the outcome of a greedy procedure, which adds items in order of non-increasing densities. During the procedure, we add an item j only if $\delta_j^1(W_A + w_j - 1) \geq 0$, i.e. adding the last unit of j does not decrease the net value. Thus, we may discard certain items from the sorted list, but possibly continue to add further items with smaller densities at a later point of time. The details of the algorithm are presented in Figure 1.

```
**Algorithm:**

Initialize a set of candidate solutions $C := \{\emptyset\}$, and an empty solution $A := \emptyset$.
For every item $j \in U$ do: $C := C \cup \{\{j\}\}$.
Sort the items in $U$ by non-increasing densities, and let $i_1, i_2, \ldots i_n$ be the items in this sorted order.
For $k$ from 1 to $n$, do:
    If $\delta^1_{i_k}(W_A + w_{i_k} - 1) \geq 0$, then $A := A \cup \{i_k\}$.
Set $C := C \cup \{A\}$.
Output a set $B \in C$ with maximum net value $\pi(B)$.
```

Fig. 1. A 2-approximation with running time $\mathcal{O}(n \log n)$.

It is easy to see that the running time of the proposed algorithm is dominated by the sorting of the items, which can be done in $\mathcal{O}(n \log n)$ time. We now turn to proving the claimed approximation ratio of 2.

Theorem 1. *The set B returned by the algorithm in Figure 1 satisfies $\pi(B) \geq 1/2 \cdot \mathrm{OPT}$.*

The high-level idea of the proof of Theorem 1 is as follows. First, we divide the items into two categories: *heavy* and *light* (see Definition 1). In Lemma 1 we show that each heavy item in the optimal solution, by itself, is a 2-approximation. On the other hand, if there is no heavy item in OPT, in Lemma 3 we show that the greedy procedure results in a 2-approximation. Then Theorem 1 follows easily from the two lemmas.

Definition 1. *An item $j \in U$ is called* heavy *if one of the following is true:*

- $w_j > W_{\mathrm{OPT}}$
- $w_j \leq W_{\mathrm{OPT}}$ *and* $\delta_j^{-w_j}(W_{\mathrm{OPT}}) < -1/2 \cdot \mathrm{OPT}$ *(removing j from a knapsack with total weight W_{OPT} decreases the net value by more than $1/2 \cdot \mathrm{OPT}$)*

Items which are not heavy are called light. *We denote the set of heavy items by $H := \{j | j \text{ is heavy}\}$.*

We remark that this definition is only used in the analysis of our algorithm and never in the algorithm itself.

Lemma 1. *Assume* OPT *contains some heavy item j. Then $\pi(\{j\}) > 1/2 \cdot \mathrm{OPT}$.*

Proof. First of all, observe that any heavy item in OPT satisfies the second condition of Definition 1, because its weight cannot be more than the total weight of OPT. Thus we have $\delta_j^{-w_j}(W_{\mathrm{OPT}}) < -1/2 \cdot \mathrm{OPT}$. Furthermore it holds that $\delta_j^{-w_j}(w_j) \leq \delta_j^{-w_j}(W_{\mathrm{OPT}})$ by Proposition 1. Therefore $\delta_j^{-w_j}(w_j) < -1/2 \cdot \mathrm{OPT}$, and since $\pi(\{j\}) + \delta_j^{-w_j}(w_j) = 0$, we get $\pi(\{j\}) > 1/2 \cdot \mathrm{OPT}$. □

We now consider the case that OPT does not contain any heavy item. Our first step is to define a family of fractional solutions, which will serve as an upper bound on OPT. To this end, consider the items in $U \setminus D$, where D is a given set of items that were discarded by the algorithm in the second for loop, and sort them by non-increasing densities (breaking ties in the same way as the algorithm does). We regard these items as a continuous "stream", i.e. a sequence of unit-size item fractions.

Definition 2. *We denote by $F_D(W)$ the (fractional) solution obtained by picking the first W units from the described stream.*

Note that $F_D(W)$ contains only complete (non-fractional) items, plus at most one fractional item. The next lemma summarizes a number of useful properties satisfied by $F_D(W)$.

Lemma 2. *The following three statements are true for any set of items D and integer weights W and W':*

(1) $\pi\big(F_D(W-1)\big) \le \pi\big(F_D(W)\big) \;\Rightarrow\; \pi\big(F_D(W')\big) \le \pi\big(F_D(W)\big) \;\forall\, W' \le W$
(2) $\pi\big(F_D(W-1)\big) > \pi\big(F_D(W)\big) \;\Rightarrow\; \pi\big(F_D(W')\big) < \pi\big(F_D(W)\big) \;\forall\, W' > W$
(3) $\mathrm{OPT} \cap D = \emptyset \;\Rightarrow\; \pi\big(F_D(W_{\mathrm{OPT}})\big) \ge \mathrm{OPT}$

Proof. Observe that $\pi\big(F_D(W)\big) - \pi\big(F_D(W-1)\big) = d - c(W) + c(W-1)$, where d is the density of the W^{th} unit in the stream. Since the items in the stream are ordered by non-increasing densities, and c is convex, it follows that $\pi\big(F_D(W)\big) - \pi\big(F_D(W-1)\big)$ is non-increasing in W. This immediately proves statements (1) and (2).

For statement (3), note that $F_D(W)$ maximizes the net value among all (possibly fractional) solutions with total weight W, that do not contain any item from D. This is clearly true, because the cost is the same for all those solutions ($= c(W)$), and $F_D(W)$ maximizes the profit by choosing the units with the highest densities. Statement (3) follows. □

Using these properties, we can now prove the following lemma, which is the main technical contribution of this section.

Lemma 3. *If $\mathrm{OPT} \cap H = \emptyset$, then the candidate A constructed by the greedy procedure achieves $\pi(A) \ge 1/2 \cdot \mathrm{OPT}$.*

Proof. The algorithm starts the construction of A with an empty set $A_0 = \emptyset$, and then iterates through the sorted list of items to augment A_0. During the iteration, some items are added to A_0, while others are discarded. We restrict our attention to the moment t, when it happens for the first time that a light item z gets discarded. If this never happens, t is just the time when the iteration is finished and all items have been considered. Let D_t be the set of items discarded up to this point (excluding z), and let A_t be the solution constructed until now. Then $D_t \subseteq H$, while $\mathrm{OPT} \cap H = \emptyset$. Thus $D_t \cap \mathrm{OPT} = \emptyset$, and hence

$$\pi\big(F_{D_t}(W_{\mathrm{OPT}})\big) \ge \mathrm{OPT} \qquad (1)$$

by Lemma 2 (3). In the remaining part of the proof, our goal is to show the following inequality:

$$\pi\big(F_{D_t}(W_{A_t})\big) \geq \pi\big(F_{D_t}(W_{\mathrm{OPT}})\big) - 1/2 \cdot \mathrm{OPT}. \tag{2}$$

Together with (1) this proves the lemma for the following reason. First of all, observe that A_t is exactly the same set of items as $F_{D_t}(W_{A_t})$, and therefore (1) and (2) imply $\pi(A_t) \geq 1/2 \cdot \mathrm{OPT}$. As the iteration proceeds, the algorithm may add further jobs to A_t, but for each such job j we know that $\delta_j^1(W_{cur}+w_j-1) \geq 0$, where W_{cur} is the total weight of A before j is added. Therefore adding item j changes the net value by

$$\delta_j^{w_j}(W_{cur}) = \sum_{r=0}^{w_j-1} \delta_j^1(W_{cur}+r) \geq w_j \cdot \delta_j^1(W_{cur}+w_j-1) \geq 0,$$

where the second inequality holds by Proposition 1. Thus every further job which is added to A_t does not decrease the net value, and hence the final candidate A satisfies $\pi(A) \geq \pi(A_t) \geq 1/2 \cdot \mathrm{OPT}$.

For the proof of (2), we distinguish two cases. Let us first assume that $W_{\mathrm{OPT}} \leq W_{A_t}$. Clearly (2) is satisfied when $W_{\mathrm{OPT}} = W_{A_t}$, so we can assume $W_{\mathrm{OPT}} < W_{A_t}$. Then at least one item has been added to A_0 during the iteration up to time t. Let k be the last item that was added on the way from A_0 to A_t. Since the algorithm decided to add k, we have $0 \leq \delta_k^1(W_{A_t}-1)$. Now observe that $\pi\big(F_{D_t}(W_{A_t})\big) - \pi\big(F_{D_t}(W_{A_t}-1)\big) = d_k - c(W_{A_t}) + c(W_{A_t}-1) = \delta_k^1(W_{A_t}-1) \geq 0$. Hence we can apply Lemma 2 (1) to obtain $\pi\big(F_{D_t}(W_{A_t})\big) \geq \pi\big(F_{D_t}(W_{\mathrm{OPT}})\big)$.

We are left with the case that $W_{\mathrm{OPT}} > W_{A_t}$. In this case there must indeed be a light item z which gets discarded at time t, because if the iteration was finished at t and all light items were accepted, we would have $\mathrm{OPT} \subseteq A_t$, implying $W_{\mathrm{OPT}} \leq W_{A_t}$, a contradiction.

The reason for the rejection of z is that $0 > \delta_z^1(W_{A_t}+w_z-1)$, where the latter is equal to $\pi\big(F_{D_t}(W_{A_t}+w_z)\big) - \pi\big(F_{D_t}(W_{A_t}+w_z-1)\big)$. We claim that this implies $W_{\mathrm{OPT}} \leq W_{A_t}+w_z$. To see this, suppose for the sake of contradiction that $W_{\mathrm{OPT}} > W_{A_t}+w_z$. Then Lemma 2 (2) yields $\pi\big(F_{D_t}(W_{\mathrm{OPT}})\big) < \pi\big(F_{D_t}(W_{A_t}+w_z)\big)$, and together with (1) we obtain $\pi\big(F_{D_t}(W_{A_t}+w_z)\big) > \mathrm{OPT}$. However, $F_{D_t}(W_{A_t}+w_z)$ is a feasible (non-fractional) solution, which contradicts the optimality of OPT. Therefore we have

$$W_{A_t} < W_{\mathrm{OPT}} \leq W_{A_t} + w_z. \tag{3}$$

We continue by comparing the solutions $F_{D_t}(W_{A_t})$ and $F_{D_t}(W_{\mathrm{OPT}})$. Using (3), we see that both solutions differ only by some non-zero fraction of z, namely by $\Delta w := W_{\mathrm{OPT}} - W_{A_t}$ units of z. In terms of the net value, we have

$$\pi\big(F_{D_t}(W_{A_t})\big) = \pi\big(F_{D_t}(W_{\mathrm{OPT}})\big) + \delta_z^{-\Delta w}(W_{\mathrm{OPT}}).$$

So all we need to show for completing the proof of (2), is that

$$\delta_z^{-\Delta w}(W_{\mathrm{OPT}}) \geq -1/2 \cdot \mathrm{OPT}.$$

As z is a light item, we already know that $\delta_z^{-w_z}(W_{\mathrm{OPT}}) \geq -1/2 \cdot \mathrm{OPT}$. Now consider the function $f(x) := x \cdot d_z - c(W_{\mathrm{OPT}} + x) + c(W_{\mathrm{OPT}})$ and observe that f is concave. Clearly, $f(0) = 0$ and $f(-w_z) = \delta_z^{-w_z}(W_{\mathrm{OPT}}) \geq -1/2 \cdot \mathrm{OPT}$. By the concavity of f we can conclude that $f(x) \geq -1/2 \cdot \mathrm{OPT}$ for all $-w_z \leq x \leq 0$. Therefore $-1/2 \cdot \mathrm{OPT} \leq f(-\Delta w) = \delta_z^{-\Delta w}(W_{\mathrm{OPT}})$. This completes the proof of (2), and hence the proof of the entire lemma. □

Now Theorem 1 follows from Lemmas 1 and 3.

4 FPTAS for Convex Costs

In this section we describe an FPTAS for our problem if the cost function is convex. The main idea of the FPTAS is to build up a tableau of polynomial size using dynamic programming. Although obtaining a dynamic program of pseudo-polynomial size is a relatively straightforward task, designing one that can be effectively rounded is significantly more involved. To this end, we exploit the convexity of the objective function to obtain a tableau that is indexed by the net value achievable by sub-solutions. We will show that we can reduce the size of this tableau to make it polynomial. In this process we again exploit convexity. For the sake of conciseness we describe directly the tableau with polynomial size.

Before defining the dynamic program, we use the 2-approximation algorithm from the previous section to obtain an upper bound UB on OPT, s.t. $\mathrm{OPT} \leq \mathrm{UB} \leq 2\mathrm{OPT}$. Further, for all $k \in \{0, \dots, n\}$ and $N \in [0, \mathrm{UB}]$, we define $T^*(k, N)$ to be the minimum weight achievable by a subset of items $A \subseteq \{1, \dots, k\}$ whose net value is at least N, i.e.,

$$T^*(k, N) := \min\{W_A \,|\, \pi(A) \geq N, A \subseteq \{1, \dots, k\}\}.$$

If no subset $A \subseteq \{1, \dots, k\}$ with $\pi(A) \geq N$ exists, we let $T^*(k, N) := \infty$.

To elaborate the details of the FPTAS, let us fix an error bound $\epsilon > 0$, and let $\ell_{max} := \lceil (n+1)/\epsilon \rceil$ and $C := \mathrm{UB}/\ell_{max}$. Using dynamic programming, we build up a table $T(k, l)$, where $k \in \{0, \dots, n\}$ and $\ell \in \{0, \dots, \ell_{max}\}$. Intuitively, $T(k, \ell)$ compacts all values of $T^*(k, N)$ for $N \in [\ell C, (\ell+1)C)$ into one entry. We now describe the dynamic program. Initially, we set $T(0, 0) := 0$ and $T(0, \ell) := \infty$ for all $\ell \neq 0$. The remaining entries are computed by the recursive formula

$$\begin{aligned} T(k, \ell) := \min\bigl\{&T(k-1, \ell), \\ &\min_{\ell' \geq 0}\{T(k-1, \ell') + w_k \,|\, \ell' C + \delta_k^{w_k}\bigl(T(k-1, \ell')\bigr) \geq \ell C\}\bigr\}. \end{aligned} \tag{4}$$

Intuitively, the first term in the recursive formula corresponds to the case in which item k does not belong to the set A in $\arg\min\{W_A : \pi(A) \geq \ell C, A \subseteq \{1, \dots, k\}\}$. The second term is when k belongs to this set A. In this case we need to check for the best choice among all $T(k-1, \ell')$ for $0 \leq \ell'$.

In the following lemma we show that unless $T(k, \ell) = \infty$, there always exists a set $A \subseteq \{1, \dots, k\}$ with weight $T(k, \ell)$ and net value at least ℓC.

Lemma 4. *If $T(k,\ell) < \infty$, then there exists $A \subseteq \{1,\ldots,k\}$ such that $W_A = T(k,\ell)$ and $\pi(A) \geq \ell C$.*

Proof. We show this by induction on k. The base case for $k = 0$ holds trivially. If $T(k,\ell) = T(k-1,\ell)$ then the claim is trivially true. Otherwise, let ℓ' be such that $T(k,\ell) = T(k-1,\ell') + w_k$ and $\ell' C + \delta_k^{w_k}\big(T(k-1,\ell')\big) \geq \ell C$. By induction hypothesis, we have that there exists $A' \subseteq \{1,\ldots,k-1\}$ with $W_{A'} = T(k-1,\ell')$ and $\pi(A') \geq \ell' C$. Then taking $A := A' \cup \{k\}$ yields a set satisfying $W_A = T(k,\ell') + w_k = T(k,\ell)$ and $\pi(A) = \pi(A') + \delta_k^{w_k}(W_{A'}) \geq \ell' C + \delta_k^{w_k}\big(T(k-1,\ell')\big) \geq \ell C$. □

The next lemma shows that the table T provides, in an appropriate fashion, a good approximation for T^*. In the following we denote by $x_+ := \max\{x, 0\}$ for all $x \in \mathbb{R}$.

Lemma 5. *For all $k \in \{0,\ldots,n\}$ and $N \in [0, \mathrm{UB}]$ it holds that*

$$T\left(k, (\ell_N - k)_+\right) \leq T^*(k,N), \text{ where } \ell_N = \left\lfloor \frac{N}{C} \right\rfloor.$$

Before showing this lemma we show why it implies an FPTAS for our problem.

Theorem 2. *If the function c is convex, our problem admits a $(1-\epsilon)$-approximation algorithm with running time $\mathcal{O}(n^3/\epsilon^2)$ for any $\epsilon > 0$.*

Proof. Using the proposed dynamic program, we fill the table T and search for the largest ℓ such that $T(n,\ell) < \infty$. For proving correctness, take $N := \mathrm{OPT}$ and $k := n$ in our previous lemma. It follows that $T\left(n, (\ell_{\mathrm{OPT}} - n)_+\right) \leq T^*(n, \mathrm{OPT}) < \infty$. Then, by Lemma 4, there exists a set of items A with

$$\pi(A) = (\ell_{\mathrm{OPT}} - n)_+ C \geq (\left\lfloor \frac{\mathrm{OPT}}{C} \right\rfloor - n)C \geq (\frac{OPT}{C} - n - 1)C = \mathrm{OPT} - C(n+1)$$
$$\geq \mathrm{OPT} - \epsilon \mathrm{UB} \geq \mathrm{OPT} - 2\epsilon \mathrm{OPT}.$$

This set A can be easily computed by back-tracking. Thus, redefining ϵ as $\epsilon/2$ yields a $(1-\epsilon)$-approximation. The table has $\mathcal{O}(n^2/\epsilon)$ many entries, and each entry takes $\mathcal{O}(n/\epsilon)$ time to compute. This implies the claimed running time. □

It remains to show Lemma 5. In order to do that we need the following technical property.

Proposition 2. *For all k and $\ell_1 \leq \ell_2$ it holds that $T(k,\ell_1) \leq T(k,\ell_2)$.*

Proof. We argue by induction on k. If $k = 0$ then the proposition holds immediately. By induction hypothesis, we assume that the proposition holds for $k-1$ and all $\ell_1 \leq \ell_2$. Let us fix a value for ℓ_1 and ℓ_2 with $\ell_1 \leq \ell_2$. By (4) we have that either $T(k,\ell_2) = T(k-1,\ell_2)$, or $T(k,\ell_2) = T(k-1,\ell') + w_k$ for some $\ell' \geq 0$ such that $\ell' C + \delta_k^{w_k}(T(k-1,\ell')) \geq \ell_2 C$. If the first case holds, then (4) implies that $T(k, \ell_1) \leq T(k-1, \ell_1)$, and the induction hypothesis yields that $T(k-1, \ell_1) \leq T(k-1, \ell_2) = T(k, \ell_2)$. We conclude that $T(k, \ell_1) \leq T(k, \ell_2)$ in this case.

For the second case, where $T(k, \ell_2) = T(k-1, \ell') + w_k$, it is enough to notice that $T(k, \ell_1) \leq T(k-1, \ell') + w_k = T(k, \ell_2)$. Indeed, this follows from (4), by noting that

$$\ell' C + \delta_k^{w_k}(T(k-1, \ell')) \geq \ell_2 C \geq \ell_1 C.$$

□

Proof (Lemma 5). We show the lemma by induction on k. For $k = 0$, the claim follows directly from the initial conditions for T. We now assume that the claim is true for $k-1$ and all $N \in [0, \mathrm{UB}]$. For the induction step, let us fix a value N. If $T^*(k, N) = \infty$, the claim is obviously true. Otherwise there exists a set of items $A \subseteq \{1, \ldots, k\}$, such that $\pi(A) \geq N$ and $W_A = T^*(k, N)$. We consider two cases.
Case 1. $k \notin A$.

In this case we have that $T^*(k-1, N) = T^*(k, N)$. Therefore it holds that

$$\begin{aligned} T\left(k, (\ell_N - k)_+\right) &\leq T\left(k-1, (\ell_N - k)_+\right) \leq T\left(k-1, (\ell_N - k + 1)_+\right) \\ &\leq T^*(k-1, N) = T^*(k, N). \end{aligned}$$

Here, the first inequality follows from (4), the second by the previous proposition, and the third one by the induction hypothesis.
Case 2. $k \in A$.

For this case we consider the value $N' := \pi(A \setminus \{k\})$, and we use the convexity of the cost function to show the following property.
Claim. It holds that $T^*(k-1, N') = W_{A \setminus \{k\}}$.

We show the claim by contradiction. Let us assume that there exists a set $A^* \subseteq \{1, \ldots, k-1\}$ with $\pi(A^*) \geq N'$ and $W_{A^*} < W_{A \setminus \{k\}}$. Then $W_{A^* \cup \{k\}} < W_A$, and

$$\begin{aligned} \pi(A^* \cup \{k\}) &= \pi(A^*) + \delta_k^{w_k}(W_{A^*}) \\ &\geq N' + \delta_k^{w_k}(W_{A^*}) \\ &\geq N' + \delta_k^{w_k}(W_{A \setminus \{k\}}) = \pi(A), \end{aligned}$$

where the last inequality follows from Proposition 1 and $W_{A^*} < W_{A \setminus \{k\}}$. This implies that $T^*(k, N) < W_A$, which contradicts the definition of A. The claim follows.

Using this claim we can conclude

$$N' + \delta_k^{w_k}\left(T^*(k-1, N')\right) \geq N. \tag{5}$$

To simplify notation, let us call $r := (\ell_{N'} - k + 1)_+$. Our goal is to prove

$$T(k, (\ell_N - k)_+) \leq T(k-1, r) + w_k. \tag{6}$$

Then

$$T(k-1,r)+w_k \le T^*(k-1,N')+w_k = W_{A\setminus\{k\}}+w_k = T^*(k,N),$$

where the first inequality follows from the induction hypothesis, and the subsequent equality holds by the above claim, completes the induction step. To see that (6) holds, note that

$$\begin{aligned} r\cdot C+\delta_k^{w_k}\big(T(k-1,r)\big) &\ge N'-kC+\delta_k^{w_k}\big(T(k-1,r)\big)\\ &\ge N'-kC+\delta_k^{w_k}\big(T^*(k-1,N')\big)\\ &\ge N-kC \ge (\ell_N-k)C. \end{aligned}$$

In this computation, the first inequality holds by the definition of $\ell_{N'}$. The second one follows by Proposition 1, and since the induction hypothesis implies that $T(k-1,r)\le T^*(k-1,N')$. The third inequality is implied by (5).

This chain of inequalities implies (6), as long as $\ell_N-k=(\ell_N-k)_+$, since $\ell'=r$ is one possible choice in (4). On the other hand, if $(\ell_N-k)_+=0$, then (6) also holds. Indeed, it is enough to notice that $T(0,0)=0$, and that applying (4) iteratively implies that $T(k,0)=0$. This completes the proof. $\square$

5 Concave Costs

In this section we consider the problem under a concave cost function c.

Lemma 6. *Let* OPT *be an optimal solution of maximal weight. Then* OPT *is comprised of exactly the items that have density at least* $c(W_{\mathrm{OPT}}+1)-c(W_{\mathrm{OPT}})$.

Proof. Let P be the total profit and C the total cost of OPT. We prove the lemma by contradiction.

First, assume that there exists an item $i\in\mathrm{OPT}$ with $d_i<c(W_{\mathrm{OPT}}+1)-c(W_{\mathrm{OPT}})$. It follows that $p_i<w_i(c(W_{\mathrm{OPT}}+1)-c(W_{\mathrm{OPT}}))$. The solution $\mathrm{OPT}\setminus\{i\}$ has cost

$$\begin{aligned} C' &= C-(c(W_{\mathrm{OPT}})-c(W_{\mathrm{OPT}}-w_i))\\ &= C-\sum_{k=1}^{w_i}(c(W_{\mathrm{OPT}}-k+1)-c(W_{\mathrm{OPT}}-k))\\ &\le C-w_i(c(W_{\mathrm{OPT}}+1)-c(W_{\mathrm{OPT}})), \end{aligned}$$

where the last inequality holds because every addend in the sum is at least $c(W_{\mathrm{OPT}}+1)-c(W_{\mathrm{OPT}})$ by concavity of c. The profit of $\mathrm{OPT}\setminus\{i\}$ is given by $P'=P-p_i$. The total net value of $\mathrm{OPT}\setminus\{i\}$ is therefore

$$\pi(\mathrm{OPT}\setminus\{i\})=P'-C'\ge P-p_i-C+w_i(c(W_{\mathrm{OPT}}+1)-c(W_{\mathrm{OPT}}))>\mathrm{OPT}.$$

This contradicts the optimality of OPT and shows therefore that no item of density less than $c(W_{\mathrm{OPT}}+1)-c(W_{\mathrm{OPT}})$ can be included in an optimal solution of total weight W_{OPT}.

Assume now that there exists an item $i \notin \mathrm{OPT}$, with $d_i \geq c(W_{\mathrm{OPT}} + 1) - c(W_{\mathrm{OPT}})$. It follows that $p_i \geq w_i(c(W_{\mathrm{OPT}} + 1) - c(W_{\mathrm{OPT}}))$. We show that $\pi(\mathrm{OPT} \cup \{i\}) \geq \mathrm{OPT}$, which again contradicts our assumptions for OPT. The solution $\mathrm{OPT} \cup \{i\}$ has profit $P' = P + p_i$, and total cost

$$\begin{aligned} C' &= C + (c(W_{\mathrm{OPT}} + w_i) - c(W_{\mathrm{OPT}})) \\ &= C + \sum_{k=1}^{w_i} (c(W_{\mathrm{OPT}} + k) - c(W_{\mathrm{OPT}} + k - 1)) \\ &\leq C + w_i(c(W_{\mathrm{OPT}} + 1) - c(W_{\mathrm{OPT}})), \end{aligned}$$

where the last inequality holds because every addend in the sum is at most $c(W_{\mathrm{OPT}} + 1) - c(W_{\mathrm{OPT}})$ by concavity of c. Therefore

$$\begin{aligned} \pi(\mathrm{OPT} \cup \{i\}) &\geq P + p_i - C - w_i(c(W_{\mathrm{OPT}} + 1) - c(W_{\mathrm{OPT}})) \\ &= \pi(\mathrm{OPT}) + p_i - w_i(c(W_{\mathrm{OPT}} + 1) - c(W_{\mathrm{OPT}})) \\ &\geq \mathrm{OPT}. \end{aligned}$$

□

Lemma 6 directly suggests the algorithm in Figure 2.

```
Algorithm:

Initialize a solution $A := \emptyset$, and a candidate solution $D := \emptyset$.
Sort the items in $U$ by non-increasing densities, and let $i_1, i_2, \dots i_n$ be the
items in this sorted order.
For $j$ from 1 to $n$, do:
    $D := D \cup \{i_j\}$.
    If $\pi(D) \geq \pi(A)$, then
        $A := D$.
Output set $A$.
```

Fig. 2. An optimal algorithm for the concave costs setting.

It is easy to see that this algorithm returns an optimal solution, since it selects the best prefix of the sorted items, and Lemma 6 implies that OPT is such a prefix. The running time of the algorithm is dominated by the sorting of the items. Note that by storing some additional information, $\pi(D)$ can be computed in constant time in each iteration, because $\pi(D) = \pi(D') + \delta_{i_j}^{w_{i_j}}(W')$, where D' corresponds to the candidate solution of the previous iteration, and W' to its weight. Therefore we obtain the following theorem.

Theorem 3. *For concave cost functions, an optimal solution can be computed in time $\mathcal{O}(n \log n)$.*

6 Conclusion

In this paper, we study a generalization of the classical knapsack problem, where the "hard" constraint on the capacity is replaced by a "soft" weight-dependent cost function. An interesting direction for further study is to impose certain restrictions on the set of chosen items. In Barman et al. [1], they show that if a matroid constraint is imposed on the items, i.e. the chosen items must be an independent set in the matroid, they can achieve a 4-approximation for the case of a convex cost function. An obvious question is whether this can be further improved.

References

1. Barman, S., Umboh, S., Chawla, S., Malec, D.L.: Secretary problems with convex costs. In: Czumaj, A., Mehlhorn, K., Pitts, A.M., Wattenhofer, R. (eds.) ICALP (1). Lecture Notes in Computer Science, vol. 7391, pp. 75–87. Springer (2012)
2. Brooks, D., Bose, P., Schuster, S., Jacobson, H.M., Kudva, P., Buyuktosunoglu, A., Wellman, J.D., Zyuban, V.V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. IEEE Micro 20(6), 26–44 (2000)
3. Buchbinder, N., Feldman, M., Naor, J., Schwartz, R.: A tight linear time (1/2)-approximation for unconstrained submodular maximization. In: FOCS. pp. 649–658. IEEE Computer Society (2012)
4. Feige, U., Mirrokni, V.S., Vondrák, J.: Maximizing non-monotone submodular functions. In: FOCS. pp. 461–471. IEEE Computer Society (2007)
5. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
6. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM 42(6), 1115–1145 (1995)
7. Halperin, E., Zwick, U.: Combinatorial approximation algorithms for the maximum directed cut problem. In: Kosaraju, S.R. (ed.) SODA. pp. 1–7. ACM/SIAM (2001)
8. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. J. ACM 22(4), 463–468 (1975)
9. Lee, J., Mirrokni, V.S., Nagarajan, V., Sviridenko, M.: Non-monotone submodular maximization under matroid and knapsack constraints. In: Mitzenmacher, M. (ed.) STOC. pp. 323–332. ACM (2009)
10. O'Sullivan, A., Sheffrin, S.: Economics: Principles in Action. Pearson Prentice Hall (2003)

A Inapproximability of Our Problem with Arbitrary Cost Functions

Lemma 7. *If the cost function c is arbitrary, it is NP-hard to decide if* $\mathrm{OPT} = 0$ *or* $\mathrm{OPT} = 1$.

Proof. Consider an instance of the 2-Partition problem defined by $a_1, \ldots, a_n \in \mathbb{Z}_{>0}$ and $A = \sum_{i=1}^{n} a_i$. It is known [5] that it is NP-hard to decide whether there exists a subset $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = A/2$. We define an instance of our problem where each item $i \in \{1, \ldots, n\}$ has profit and weight equal to a_i. The cost function is defined by $c(W) = W$ if $W \neq A/2$, and $c(A/2) = A/2 - 1$. Thus, the net value of a subset S of items equals 1 if $W_S = A/2$ and 0 otherwise. □