A Nearly-Linear Time Algorithm for Submodular Maximization with a Knapsack Constraint

Alina Ene
Department of Computer Science, Boston University, MA, USA
aene@bu.edu

Huy L. Nguyen
College of Computer and Information Science, Northeastern University, Boston, MA, USA
hlnguyen@cs.princeton.edu

Abstract

We consider the problem of maximizing a monotone submodular function subject to a knapsack constraint. Our main contribution is an algorithm that achieves a nearly-optimal, $1 - 1/e - \epsilon$ approximation, using $(1/\epsilon)^{O(1/\epsilon^4)} n \log^2 n$ function evaluations and arithmetic operations. Our algorithm is impractical but theoretically interesting, since it overcomes a fundamental running time bottleneck of the multilinear extension relaxation framework. This is the main approach for obtaining nearly-optimal approximation guarantees for important classes of constraints but it leads to $\Omega(n^2)$ running times, since evaluating the multilinear extension is expensive. Our algorithm maintains a fractional solution with only a constant number of entries that are strictly fractional, which allows us to overcome this obstacle.

2012 ACM Subject Classification Theory of computation → Submodular optimization and polymatroids

Keywords and phrases submodular maximization, knapsack constraint, fast algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.53

Category Track A: Algorithms, Complexity and Games

Related Version https://arxiv.org/abs/1709.09767

Funding *Alina Ene*: Partially supported by NSF CAREER grant CCF-1750333 and NSF grant CCF-1718342.
Huy L. Nguyen: Partially supported by NSF CAREER grant CCF-1750716.

Acknowledgements This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing.

1 Introduction

A set function $f : 2^V \to \mathbb{R}$ is *submodular* if for every $A, B \subseteq V$, we have $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. Submodular functions naturally arise in a variety of contexts, both in theory and practice. Submodular functions capture many well-studied combinatorial functions including cut functions of graphs and digraphs, weighted coverage functions, as well as continuous functions including the Shannon entropy and log-determinants. Submodular functions are used in a wide range of application domains from machine learning to economics. In machine learning, it is used for document summarization [9], sensor placement [7], exemplar clustering [3], potential functions for image segmentation [4], etc. In an economics context, it can be used to model market expansion [2], influence in social networks [5], etc. The core mathematical problem underpinning many of these applications is the meta problem of maximizing a submodular objective function subject to some constraints.

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 53; pp. 53:1–53:12


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A common approach to submodular maximization is a two-step framework based on the multilinear extension F of f, a continuous function that extends f to the domain $[0,1]^V$. The program first (1) maximizes $F(x)$ subject to a continuous relaxation of the constraint and then (2) rounds the solution x to an integral vector satisfying the constraint. This paradigm has been very successful and it has led to the current best approximation algorithms for a wide variety of constraints including cardinality constraints, knapsack constraints, matroid constraints, etc. One downside with this approach is that in general, evaluating the multilinear extension is expensive and it is usually approximately evaluated. To achieve the desirable approximation guarantees, the evaluation error needs to be very small and in a lot of cases, the error needs to be $O(n^{-1})$ times the function value. Thus, even an efficient algorithm with $O(n)$ queries to the multilinear extension would require $\Omega(n^2)$ running time.

In this work, we develop a new algorithm that achieves $1 - 1/e - \epsilon$ approximation for maximizing a monotone submodular function subject to a knapsack constraint. The basic approach is still based on the multilinear extension but the algorithm ensures that the number of fractional coordinates is constant, which allows evaluating the multilinear extension exactly in constant number of queries to the original function. This approach allows us to bypass the obstructions discussed above and get nearly linear running time.

▶ **Theorem 1.** *There is an algorithm for maximizing a monotone submodular function subject to a knapsack constraint that achieves a $1 - 1/e - \epsilon$ approximation using $(1/\epsilon)^{O(1/\epsilon^4)} n \log n$ function evaluations and $(1/\epsilon)^{O(1/\epsilon^4)} n \log^2 n$ arithmetic operations.*

For simplicity, when stating running times, we assume that each call to the value oracle of f takes constant time, since for the algorithms discussed the number of evaluations dominates the running time up to logarithmic factors. Previously, Wolsey [11] gives an algorithm with a $1 - 1/e^\beta \approx 0.35$ approximation, where β is the unique root of the equation $e^x = 2 - x$. Building on the work of Khuller *et al.* for the maximum k-coverage problem [6], Sviridenko [10] gives an algorithm with a $1 - 1/e$ approximation that runs in $O(n^5)$ time. Badanidiyuru and Vondrak [1] give an algorithm with a $1 - 1/e - \epsilon$ approximation running in $n^2(\log n/\epsilon)^{O(1/\epsilon^8)}$ time. Our work builds on [1] and we discuss the relationship between the two algorithms in more detail in Section 1.1.

Kulik *et al.* [8] obtain a $1 - 1/e - \epsilon$ approximation for d knapsack constraints in time $\Omega(n^{d/\epsilon^4})$ that comes from enumerating over d/ϵ^4 items. The techniques in this paper could likely be extended to obtain an algorithm for the continuous problem of maximizing the multilinear extension subject to d knapsack constraints, with a running time that is exponential in d and nearly-linear in n. We leave it as an open problem whether the rounding can also be extended to multiple knapsack constraints.

Remark on the algorithm of [1]. We note that there are some technical issues in the algorithm proposed in [1] for a knapsack constraint. The main issue, which was pointed out by Yoshida [12], arises in the partitioning of the items into large and small items: an item e is small if it has value $f(\{e\}) \leq \epsilon^6 f(\mathrm{OPT})$ and cost $c_e \leq \epsilon^4$, and it is large otherwise. The algorithm enumerates the marginal values of the large items and thus the set of large items was intended to be of size $\mathrm{poly}(1/\epsilon)$. But this may not be true in general, as there could be many items in OPT with singleton value greater than $\epsilon^6 f(\mathrm{OPT})$. On the other hand, the assumption that the small items have small singleton values is crucial to ensuring that the algorithm obtains a good value from the small items. Another issue arises in the rounding algorithm. The fractional solution is rounded using a rounding algorithm for a partition matroid that treats the parts independently. But in this setting an item participates in several parts and we need to ensure that it is not selected more than once.

1.1 Our techniques

As in the classical knapsack problem with a linear objective, the algorithms achieving optimal approximation are based on enumeration techniques. One such approach is to enumerate the most valuable items in OPT (in the submodular problem, we can determine which items of OPT are valuable based on the Greedy ordering of OPT, see (1)) and greedily pack the remaining items based on the marginal gain to cost density. This approach leads to the optimal $1 - 1/e$ approximation provided that we enumerate 3 items [10]. The running time of the resulting algorithm is $O(n^5)$ and it can be improved to $O(n^4 \log(n/\epsilon)/\epsilon)$ time at a loss of ϵ in the approximation.

A different approach, inspired by the algorithms for the classical knapsack problem that use dynamic programming over the (appropriately discretized) profits of the items, is to enumerate over the marginal gains of the valuable items of OPT. Unlike the classical setting with linear profits, it is considerably more challenging to leverage such an approach in the submodular setting. Badanidiyuru and Vondrak [1] propose a new approach based on this enumeration technique and continuous density Greedy with a running time of $n^2 \left(\frac{\log n}{\epsilon}\right)^{O\left(\frac{1}{\epsilon^8}\right)}$, which overcame the $\Omega(n^4)$ running time barrier for the approaches that are based on enumerating items.

In this work, we build on the approach introduced by [1] and we obtain a faster running time of $\left(\frac{1}{\epsilon}\right)^{O\left(\frac{1}{\epsilon^4}\right)} n \log^2 n$. Our algorithm is impractical due to the high dependency on ϵ, but it is theoretically interesting. Obtaining near-optimal approximations in nearly-linear time for submodular maximization has been out of reach for all but a cardinality constraint.

Obtaining a fast running time poses several conceptual and technical challenges, and we highlight some of them here. Let us denote the valuable items of OPT as OPT_1, and let $\mathrm{OPT}_2 = \mathrm{OPT} \setminus \mathrm{OPT}_1$. For our algorithm, the set OPT_1 has $\mathrm{poly}(1/\epsilon)$ items and we can handle them by enumerating over their marginal gains, appropriately discretized. Similarly to [1], we use the guessed marginal gains to pack items that are competitive with OPT_1: for each guessed marginal gain, we find the cheapest item whose marginal gain is at least the guessed value, and we add ϵ of the item to the fractional solution. The continuous approach is necessary for ensuring that we obtain a good approximation, but it is already introducing the following conceptual and technical difficulties:

- *We do not know how much budget is available for the remaining items.* Since we packed the items fractionally, we will need to perform the rounding to find out which of the items will be in the final solution and their total budget. But we cannot do the rounding before packing the remaining items. Additionally, we cannot afford to guess the budget of OPT_1, even approximately.
- *In the continuous setting, evaluating the multilinear extension takes $\Omega(n^2)$ time in general.*
- *We will need to ensure that we can round the resulting fractional solution.*

A key idea in our algorithm, and an important departure from the approach of [1], is to *integrally* pack the remaining items using density Greedy with lazy evaluations to obtain a nearly-linear running time. The resulting fractional solution has only a constant number of entries that are strictly fractional, and we show that this is beneficial both in terms of running time and rounding: we can evaluate the multilinear extension in constant time and we can exploit the special structure of the solution to round. However, the first difficulty mentioned above remains a significant conceptual barrier for realizing this plan: if we cannot get a handle on how much budget to allocate to density Greedy, we will not be able to round the solution without violating the budget or losing value. Our solution here is based on the following insights.

Algorithm 1 KNAPSACK(f, ϵ).

$t \leftarrow 1/\epsilon^3$
$r \leftarrow 1/\epsilon$
$M \leftarrow \Theta(f(\mathrm{OPT}))$
$S_{\mathrm{best}} \leftarrow \emptyset$
Try all possible sequences:
  $\{v_{p,i}\}$: $p \in \{1, 2, \ldots, 1/\epsilon\}$, $i \in \{1, 2, \ldots, t\}$, $v_{p,i} \in \{0, \epsilon M/t, 2\epsilon M/t, \ldots, 1\}$
  $\{W_p\}$: $p \in \{1, 2, \ldots, 1/\epsilon\}$, $W_p \in \{0, \epsilon M, 2\epsilon M, \ldots, M\}$
  $\{w_{p,i}\}$: $p \in \{1, 2, \ldots, 1/\epsilon\}$, $i \in \{1, 2, \ldots, r+1\}$, $w_{p,i} \in \{0, \epsilon^2 W_p/r, 2\epsilon^2 W_p/r, \ldots, W_p\}$
for every choice $\{v_{p,i}\}$, $\{W_p\}$, $\{w_{p,i}\}$ **do**
  $x \leftarrow \mathrm{KNAPSACKGUESS}(f, \epsilon, \{v_{p,i}\}, \{W_p\}, \{w_{p,i}\})$
  $S \leftarrow \mathrm{ROUND}(x)$
  if $f(S) > f(S_{\mathrm{best}})$ **then**
    $S_{\mathrm{best}} \leftarrow S$
  end if
end for
Return S_{best}

First, note that we may assume that every item in OPT_2 has a cost that is small relative to the total budget of OPT_2: there can only be a small number of heavy items and each of them has small marginal gain on top of OPT_1, and thus we can discard them without losing too much in the approximation. Moreover, if there are no heavy items at all, we can show that density Greedy will not exceed the budget. Thus, if we knew the budget of OPT_2, we could remove all of the heavy items and run density Greedy on the remaining items.

Unfortunately, we cannot guess the budget of OPT_2 since there are too many possible choices. Instead, note that, since the cost of an item is its marginal value divided by its density, a heavy item has large value or small density. If it has small density then intuitively Greedy will not pick it. The problematic items are the ones that have large marginal values, as density Greedy may pick them and they may be too heavy. Unfortunately, we cannot filter out all the items with large marginal value, since those items may include items in OPT_2 (note that even though every item in OPT_2 has small marginal value on top of OPT_1, it can have large marginal value on top of our current fractional solution that does not necessarily contain OPT_1). Now the key observation is that the number of such items is small, and we can handle them with additional guessing.

The final step of the algorithm is to round the fractional solution to a feasible integral solution. Here we take advantage of the fact that the only entries that are strictly fractional were introduced in the OPT_1 stages of the algorithm. The fractional items can be mapped to the items in OPT_1 in such a way that every item in OPT_1 is assigned a fractional mass of at most 1 coming from items with smaller or equal cost. Thus, for each item in OPT_1, we want to select one of the items fractionally assigned to it. This is reminiscent of a partition matroid and thus a natural approach is to use a matroid rounding algorithm such as pipage rounding or swap rounding. However, an item may be fractionally assigned to more than one item in OPT_1, and we need to ensure that the rounding does not select the same item for different items in OPT_1. We show that we can do so using a careful application of swap rounding.

Algorithm 2 KnapsackGuess$(f, \epsilon, \{v_{p,i}\}, \{W_p\}, \{w_{p,i}\})$.

$t \leftarrow 1/\epsilon^3$
$r \leftarrow 1/\epsilon$
$x_0 \leftarrow 0$
for $p = 1, 2, \ldots, 1/\epsilon$ **do**
  $y^{(p,0)} \leftarrow x_{p-1}$
  $A_p \leftarrow \emptyset$
  for $i = 1, 2, \ldots, t$ **do**
    $a_{p,i} \leftarrow$ element with minimum size c_e in $\{e \notin A_p \colon F(y^{(p,i-1)} \vee \mathbf{1}_e) - F(y^{(p,i-1)}) \geq v_{p,i}\}$
    $y^{(p,i)} \leftarrow y^{(p,i-1)} + \epsilon \mathbf{1}_{a_{p,i}}$
    $A_p \leftarrow A_p \cup \{a_{p,i}\}$
  end for
  if $W_p = 0$ **then**
    Continue to the next phase $p + 1$
  end if
  $z^{(p,0)} \leftarrow y^{(p,t)}$
  $B_p \leftarrow \emptyset$
  Let r_p be the smallest $i \in \{0, 1, \ldots, r\}$ such that $w_{p,i+1} \leq \epsilon(1-\epsilon)W_p/r$. If no such i exists, let $r_p = r$. ⟨⟨ r_p `is the number of large value elements in` OPT_2 ⟩⟩
  for $i = 1, 2, \ldots, r_p$ **do**
    $b_{p,i} \leftarrow$ element with minimum size c_e in $\{e \colon F(z^{(p,i-1)} \vee \mathbf{1}_e) - F(z^{(p,i-1)}) \geq w_{p,i}\}$
    $z^{(p,i)} \leftarrow z^{(p,i-1)} \vee \mathbf{1}_{b_{p,i}}$
    $B_p \leftarrow B_p \cup \{b_{p,i}\}$
    if $F(z^{(p,i)}) - F(z^{(p,0)}) \geq \epsilon(1 - 12\epsilon)W_p$ **then**
      Set $x_p \leftarrow z^{(p,i)}$ and continue to phase $p + 1$
    end if
  end for
  if $F(z^{(p,r_p)}) - F(z^{(p,0)}) < \epsilon(1 - 12\epsilon)W_p$ **then**
    $V' \leftarrow V \setminus \{e \colon F(z^{(p,r_p)} \vee \mathbf{1}_e) - F(z^{(p,r_p)}) \geq \epsilon W_p/r\}$
    $C_p \leftarrow$ DensityGreedy$(f, z^{(p,r_p)}, \epsilon(1 - 12\epsilon)W_p - F(z^{(p,r_p)}) + F(z^{(p,0)}), V')$
    $x_p \leftarrow z^{(p,r_p)} \vee \mathbf{1}_{C_p}$
  end if
end for
Return $x_{1/\epsilon}$

2 The algorithm

We consider the problem of maximizing a monotone submodular function subject to a single knapsack constraint. Each element $e \in V$ has a cost $c_e \in \mathbb{R}_+$, and the goal is to find a set $\mathrm{OPT} \in \mathrm{argmax}\{f(S) \colon \sum_{e \in S} c_e \leq 1\}$. We assume that the knapsack capacity is 1, which we may assume without loss of generality by scaling the cost of each element by the knapsack capacity. We let $F : [0,1]^V \to \mathbb{R}_+$ denote the multilinear extension f. For every $x \in [0,1]^V$, we have

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{e \in S} x_e \prod_{e \notin S} (1 - x_e) = \mathbb{E}[f(R(x))],$$

where $R(x)$ is a random set that includes each element $e \in V$ independently with probability x_e. For two vectors x and y, we let $x \vee y$ denote the vector such that $(x \vee y)_i = \max\{x_i, y_i\}$ for all $i \in V$.

We fix an optimal solution to the problem that we denote by OPT. We assume that the algorithm knows a constant approximation of $f(\mathrm{OPT})$; such an approximation can be obtained in nearly linear time by taking the best of the following two solutions: the solution obtained by running Density Greedy (implemented using lazy evaluations, similarly to Algorithm 3) and the solution consisting of the best single element. Let $f(\mathrm{OPT}) \geq M \geq (1-\epsilon)f(\mathrm{OPT})$ denote the algorithm's guess for the optimal value. There are $O(1/\epsilon)$ choices for M given the constant approximation of $f(\mathrm{OPT})$.

We order OPT as $o_1, o_2, \ldots, o_{|\mathrm{OPT}|}$, where

$$o_i = \mathrm{argmax}_{o \in \mathrm{OPT}}(f(\{o_1, \ldots, o_{i-1}\} \cup \{o\}) - f(\{o_1, \ldots, o_{i-1}\})) \tag{1}$$

Let $t = O(1/\epsilon^3)$, $\mathrm{OPT}_1 = \{o_1, o_2, \ldots, o_t\}$, and $\mathrm{OPT}_2 = \mathrm{OPT} \setminus \mathrm{OPT}_1$.

We emphasize that we use the above ordering of OPT and the partition of OPT into OPT_1 and OPT_2 only for the analysis and to motivate the choices of the algorithm. In particular, the algorithm does not know this ordering or partition.

It is useful to filter out from OPT_2 the items that have large cost, more precisely, cost greater than $\epsilon^2(1 - c(\mathrm{OPT}_1))$. Since every element $o \in \mathrm{OPT}_2$ satisfies $f(\mathrm{OPT}_1 \cup \{o\}) - f(\mathrm{OPT}_1) \leq \epsilon^3 f(\mathrm{OPT}_1)$ and there are at most $1/\epsilon^2$ such elements, this will lead to only an $\epsilon f(\mathrm{OPT})$ loss. For ease of notation, we use OPT_2 to denote the set without these elements, i.e., we assume that $c_o \leq \epsilon^2(1 - c(\mathrm{OPT}_1))$ for every $o \in \mathrm{OPT}_2$.

Algorithm 1 gives a precise description of the algorithm. The algorithm guesses a sequence of values as follows.

Guessed values. Throughout the paper, we assume for simplicity that $1/\epsilon$ is an integer. Recall that $t = 1/\epsilon^3$. Let $r = 1/\epsilon$ (r is an upper bound on the number of items of OPT_2 that have large marginal value in each phase).

- A sequence $\{v_{1,1}, v_{1,2}, \ldots, v_{1/\epsilon,t}\}$ where $v_{p,i} \in \{0, \epsilon M/t, 2\epsilon M/t, \ldots, M\}$ is an integer multiple of $\epsilon M/t$, for all integers p and i such that $1 \leq p \leq 1/\epsilon$ and $1 \leq i \leq t$. The value $v_{p,i}$ is an approximate guess for the marginal value of $o_i \in \mathrm{OPT}_1$ during phase p. There are $t/\epsilon + 1 = 1/\epsilon^4 + 1$ choices for each $v_{p,i}$ and thus there are $(1/\epsilon^4 + 1)^{1/\epsilon^4} = (1/\epsilon)^{O(1/\epsilon^4)}$ possible sequences.
- A sequence $\{W_1, W_2, \ldots, W_{1/\epsilon}\}$ where $W_p \in \{0, \epsilon M, 2\epsilon M, \ldots, M\}$ is an integer multiple of ϵM, for all integers p such that $1 \leq p \leq 1/\epsilon$. The value W_p is an approximate guess for the total marginal value of OPT_2 in phase p. There are $1/\epsilon + 1$ choices for each W_p and thus there are $(1/\epsilon + 1)^{1/\epsilon}$ possible sequences.
- A sequence $\{w_{1,1}, w_{1,2}, \ldots, w_{1/\epsilon,1/\epsilon+1}\}$ where $w_{p,i} \in \{0, \epsilon^2 W_p/r, 2\epsilon^2 W_p/r, \ldots, W_p\}$ is an integer multiple of $\epsilon^2 W_p/r$, for all integers p and i such that $1 \leq p, i \leq 1/\epsilon$ (the value W_p is the same as in the sequence above). The values $w_{p,i}$, where $i \in \{1, 2, \ldots, 1/\epsilon\}$, are approximate guesses for the marginal values of the items in OPT_2 with large marginal value in phase p. There are $r/\epsilon^2 + 1 = 1/\epsilon^3 + 1$ choices for each $w_{p,i}$ and thus there are $(1/\epsilon)^{O(1/\epsilon^2)}$ possible sequences.

The algorithm enumerates all possible such sequences. For each choice, the algorithm works as follows. Let $\{v_{p,i}\}$, $\{W_p\}$, and $\{w_{p,i}\}$ denote the current sequences. The algorithm performs $1/\epsilon$ phases. Each phase is comprised of three stages, executed in sequence in this order: an OPT_1 stage, a stage for the large value items in OPT_2, and a Density Greedy stage. We describe each of these stages in turn.

The $\mathbf{OPT_1}$ stage of phase p. This stage uses the values $\{v_{p,i} : 1 \le i \le t\}$ as follows. We perform t iterations. In each iteration i, we consider the items not selected in previous iterations that have marginal value at least $v_{p,i}$ on top of the current solution, i.e., $F(x \vee \mathbf{1}_e) - F(x) \ge v_{p,i}$. Among these items, we select the item with minimum cost and increase its fractional value by ϵ. Together, the t iterations select t different items and increase their fractional value by ϵ.

The stage of phase p for the large value items in $\mathbf{OPT_2}$. This stage uses the value W_p and the values $\{w_{p,i} : 1 \le i \le 1/\epsilon\}$ as follows. We perform at most r iterations. In each iteration i, we find the minimum cost element that has marginal value at least $w_{p,i}$ on top of the current solution, and we integrally select this item. (Note that this is similar to the OPT_1 stage, except that we select items integrally.) At the end of the stage, if the items selected in this phase have total marginal gain at least $\epsilon(1-12\epsilon)W_p$, then we end phase p and proceed to the next phase. Otherwise, the algorithm proceeds to the Density Greedy stage.

The Density Greedy stage of phase p. If the previous stage did not reach a total marginal gain of at least $\epsilon(1-12\epsilon)W_p$, we run the discrete Density Greedy algorithm until we reach a gain of $\epsilon(1-12\epsilon)W_p$. Before running Density Greedy, we remove from consideration all elements whose marginal value is at least $\epsilon W_p/r$. In every step, the Density Greedy algorithm fully selects the item with largest density, i.e., ratio of marginal value to cost.

In order to achieve nearly linear time, we implement the Density Greedy algorithm using approximate lazy evaluations as shown in Algorithm 3. We maintain the items in a priority queue sorted by density. We initialize the marginal values and the densities with respect to the initial solution. In each iteration of the algorithm, we find an item whose density with respect to the current solution is within a factor of $(1-\epsilon)$ of the maximum density as follows. We remove the item at the top of the queue. The marginal value of the item may be stale, so we evaluate its marginal gain with respect to the current solution. If the new marginal gain is within a factor of $(1-\epsilon)$ of the old marginal gain, it follows from submodularity that the density of the item is within a factor of $(1-\epsilon)$ of the maximum density, and we select the item. If the marginal gain has changed by a factor larger than $(1-\epsilon)$, we update the density and reinsert the item in the queue. We also keep track of how many times each item's density has been updated and, if an item has been updated more than $2\ln(n/\epsilon)/\epsilon$ times, we discard the item since it can no longer contribute a significant value to the solution.

Rounding the fractional solution. After $1/\epsilon$ phases, we obtain a fractional solution with $O(1/\epsilon^4)$ fractional entries. We round the resulting fractional solution to an integral solution using swap rounding, as shown in Algorithm 4.

The following theorem states our main result for the fractional solution. We will use the second guarantee to obtain a fast rounding algorithm (see Section 3). We defer the proof of the theorem to the full version of the paper.

▶ **Theorem 2.** *There are choices for the guessed values $\{v_{p,i}\}$, $\{W_p\}$, and $\{w_{p,i}\}$ for which Algorithm 2 returns a fractional solution x with the following properties:*

(1) $F(x) \ge \left(1 - \frac{1}{e} - O(\epsilon)\right) f(\mathrm{OPT})$;

(2) *Let E be the set of all items $e \in V$ such that $0 < x_e < 1$. There exists a mapping $\sigma : E \times \{1, 2, \ldots, 1/\epsilon\} \to \mathrm{OPT}_1$ with the following properties:*

- **(a)** *For every element $e \in E$ and every phase $p \in \{1, 2, \ldots, 1/\epsilon\}$ such that $e \in A_p$, $\sigma(e,p)$ is defined and $c(e) \le c(\sigma(e,p))$.*
- **(b)** *For every element $o \in \mathrm{OPT}_1$, there are at most $1/\epsilon$ pairs (e,p) such that $\sigma(e,p) = o$.*

Algorithm 3 LazyDensityGreedy(f, x, W, V').

$S_0 \leftarrow \emptyset$
$D \leftarrow \emptyset$
$u(e) \leftarrow 0$ for all $e \in V'$
$v(e) \leftarrow F(x \vee \mathbf{1}_e) - F(x)$ for all $e \in V'$
Maintain the elements in a priority queue sorted in decreasing order by key, where the key of each element e is initialized to its density $\frac{v(e)}{c(e)}$
for $i = 1, 2, \ldots$ **do**
  while true **do**
    if queue is empty **then**
      return S_{i-1}
    end if
    Remove the element e from the priority queue with maximum key
    $v'(e) \leftarrow F(x \vee \mathbf{1}_{S_{i-1} \cup \{e\}}) - F(x \vee \mathbf{1}_{S_{i-1}})$
    $u(e) \leftarrow u(e) + 1$
    if $v(e) \geq (1 - \epsilon)v'(e)$ **then**
      $e_i \leftarrow e$
      $v(e) \leftarrow v'(e)$
      $S_i \leftarrow S_{i-1} \cup \{e_i\}$
      if $F(x \vee 1_{S_i}) - F(x) \geq W$ **then**
        return S_i
      end if
      Exit the while loop and continue to iteration $i + 1$
    else
      if $u(e) \leq \frac{2\ln(n/\epsilon)}{\epsilon}$ **then**
        $v(e) \leftarrow v'(e)$
        Reinsert e into the queue with key $\frac{v'(e)}{c(e)}$
      else
        $D \leftarrow D \cup \{e\}$
      end if
    end if
  end while
end for

3 Rounding algorithm and analysis of the final solution

In this section, we analyze the rounding algorithm (Algorithm 4) that rounds the fractional solution x guaranteed by Theorem 2. We round the fractional entries of x as follows. We initialize $\hat{x} = x$. For analysis purposes, we initialize $O = \mathrm{OPT}_1$. We sort the fractional elements in non-increasing order according to their cost. While there are fractional elements, we repeatedly move fractional mass between the two elements with highest cost as follows. Let e_1 and e_2 be the fractional elements with the highest and second-highest cost, respectively. We consider two cases:

Case 1: $\hat{x}_{e_1} + \hat{x}_{e_2} \leq 1$. With probability $\hat{x}_{e_1}/(\hat{x}_{e_1} + \hat{x}_{e_2})$, we update $\hat{x}_{e_1} \leftarrow \hat{x}_{e_1} + \hat{x}_{e_2}$ and $\hat{x}_{e_2} \leftarrow 0$; with the remaining probability, we update $\hat{x}_{e_2} \leftarrow \hat{x}_{e_1} + \hat{x}_{e_2}$ and $\hat{x}_{e_1} \leftarrow 0$. If an element becomes integral, we remove it from the list. For analysis purposes, if an element is rounded up to 1, we pair it up with the element $o_1 \in O$ with highest cost, and we update $O \leftarrow O \setminus \{o_1\}$.

Algorithm 4 ROUND(x).

Let $\sigma_1, \ldots, \sigma_k$ be the fractional coordinates of x.
Sort $\sigma_1, \ldots, \sigma_k$ so that $c_{\sigma_1} \leq c_{\sigma_2} \leq \cdots \leq c_{\sigma_k}$.
while $k > 0$ **do**
  if $k = 1$ **then**
    $x_{\sigma_1} \leftarrow 1$
    return x
  end if
  if $x_{\sigma_k} + x_{\sigma_{k-1}} > 1$ **then**
    Pick $u \in \{0, 1\}$ randomly such that $\Pr[u = 1] = \frac{1 - x_{\sigma_{k-1}}}{2 - x_{\sigma_k} - x_{\sigma_{k-1}}}$
    if $u = 1$ **then**
      $x_{\sigma_k} \leftarrow 1$
      $x_{\sigma_{k-1}} \leftarrow x_{\sigma_{k-1}} + x_{\sigma_k} - 1$
      $k \leftarrow k - 1$
    else
      $x_{\sigma_{k-1}} \leftarrow 1$
      $x_{\sigma_k} \leftarrow x_{\sigma_{k-1}} + x_{\sigma_k} - 1$
      $\sigma_{k-1} \leftarrow \sigma_k$
      $k \leftarrow k - 1$
    end if
  else
    Pick $u \in \{0, 1\}$ randomly such that $\Pr[u = 1] = \frac{x_{\sigma_k}}{x_{\sigma_k} + x_{\sigma_{k-1}}}$
    if $u = 1$ **then**
      $x_{\sigma_k} \leftarrow x_{\sigma_{k-1}} + x_{\sigma_k}$
      $x_{\sigma_{k-1}} \leftarrow 0$
      $\sigma_{k-1} \leftarrow \sigma_k$
      $k \leftarrow k - 1$
    else
      $x_{\sigma_{k-1}} \leftarrow x_{\sigma_{k-1}} + x_{\sigma_k}$
      $x_{\sigma_k} \leftarrow 0$
      $k \leftarrow k - 1$
    end if
    if $x_{\sigma_k} = 1$ **then**
      $k \leftarrow k - 1$
    end if
  end if
end while

Case 2: $\hat{x}_{e_1} + \hat{x}_{e_2} > 1$. With probability $(1 - \hat{x}_{e_2})/(2 - \hat{x}_{e_1} - \hat{x}_{e_2})$, we update $\hat{x}_{e_1} \leftarrow 1$ and $\hat{x}_{e_2} \leftarrow \hat{x}_{e_1} + \hat{x}_{e_2} - 1$; with the remaining probability, we update $\hat{x}_{e_2} \leftarrow 1$ and $\hat{x}_{e_1} \leftarrow \hat{x}_{e_1} + \hat{x}_{e_2} - 1$. If an element becomes integral, we remove it from the list. For analysis purposes, if an element is rounded up to 1, we pair it up with an element in O as follows. If the element e_1 with the highest cost is rounded up to 1, we pair up e with the element $o_1 \in O$ with highest cost, and we update $O \leftarrow O \setminus \{o_1\}$. If the element e_2 with the second-highest cost is rounded up to 1, we pair up e_2 with the element $o_2 \in O$ with the second-highest cost, and we update $O \leftarrow O \setminus \{o_2\}$.

If there is only one fractional entry then we can round this entry up to 1 and pair up this element with the element $o_1 \in O$ with highest cost.

We now turn to the analysis of the rounding. We first show that the expected value of the rounded solution is at least $F(x)$. We then show that the cost of the fractional elements that were rounded up to 1 is at most $c(\mathrm{OPT}_1)$, thus ensuring that the final rounded solution is feasible.

▶ **Lemma 3.** $\mathbb{E}[F(\hat{x})] \geq F(x)$.

Proof. Note that each iteration updates the solution as follows: $\hat{x}' = \hat{x} + \delta(\mathbf{1}_{e_1} - \mathbf{1}_{e_2})$, where δ is a random value satisfying $\mathbb{E}_\delta[\hat{x}'] = \hat{x}$. The multilinear extension is convex along the direction $\mathbf{1}_e - \mathbf{1}_{e'}$ for every pair of elements e and e'. Therefore $\mathbb{E}_\delta[F(\hat{x}')] \geq F(\mathbb{E}_\delta[\hat{x}']) = F(\hat{x})$, and the claim follows by induction. ◀

▶ **Lemma 4.** *Let $\hat{E}$ be the set of elements corresponding to the fractional entries that were rounded to* 1. *We have $c(\hat{E}) \leq c(\mathrm{OPT}_1)$.*

Proof. The lemma follows from the following invariant maintained by the algorithm for the partially rounded solution $\hat{x}$ and the set $O \subseteq \mathrm{OPT}_1$:

Invariant: Let $o_1, o_2, \ldots, o_p$ be the elements of O, labeled such that $c_{o_1} \geq c_{o_2} \geq \cdots \geq c_{o_p}$. Let $e_1, e_2, \ldots, e_\ell$ be the elements corresponding to the fractional entries of $\hat{x}$, labeled such that $c_{e_1} \geq c_{e_2} \geq \cdots \geq c_{e_\ell}$. We define the following grouping of the elements $e_1, e_2, \ldots, e_\ell$ where each group contributes a fractional mass of 1 and each element belongs to at most two groups. Consider the interval $[0, \sum_{i=1}^{\ell} x_{e_i}]$ that is divided among the elements as follows: $[0, x_{e_1})$ corresponds to e_1 and, for all $2 \leq i \leq \ell$, $[\sum_{j=1}^{i-1} x_{e_j}, \sum_{j=1}^{i} x_{e_j})$ corresponds to e_i. The elements that overlap with the interval $[i-1, i)$ define the i-th group. The invariant is that $\hat{x}$ and O satisfy the following properties:

(1) $\sum_{i=1}^{\ell} \hat{x}_{e_i} \leq |O|$, and

(2) for every $i \geq 1$ and each element e in the i-th group, we have $c_e \leq c_{o_i}$.

We will show the invariant using induction on the number of iterations. We start by showing the invariant at the beginning of the rounding algorithm. We can show the invariant for x and OPT_1 using Theorem 2.

▷ Claim 5. The invariant holds for x and OPT_1.

Proof. Recall that each phase p of the KnapsackGuess algorithm selects a set A_p of elements and it increases the values of each of these elements by ϵ. Thus the fractional value x_{e_i} of each element $e_i \in E$ is equal to ϵ times the number of phases p such that $e_i \in A_p$. Moreover, by Theorem 2, there is a mapping $\sigma : \{e_1, \ldots, e_\ell\} \times \{1, 2, \ldots, 1/\epsilon\} \to \mathrm{OPT}_1$ such that, for each phase p such that $e_i \in A_p$, $\sigma(e_i, p)$ exists and $c(e_i) \leq c(\sigma(e_i, p))$.

We can think of each element e_i having x_{e_i}/ϵ copies and each element $o \in \mathrm{OPT}_1$ having $|\sigma^{-1}(o)| \leq 1/\epsilon$ copies. By letting $\tilde{E}$ and $\tilde{O}$ be the copies of the elements in E and OPT_1 (respectively), we can equivalently view σ as a bijection between $\tilde{E}$ and $\tilde{O}$ with the property that, if $\sigma((e, i)) = (o, j)$ then $c(e) \leq c(o)$. We may also assume that the elements of O with the highest costs have $1/\epsilon$ copies, i.e., there exists an index p' such that $o_1, \ldots, o_{p'}$ have $1/\epsilon$ copies and $o_{p'+1}, \ldots, o_p$ have zero copies; we can ensure this property by reassigning pairs in $\tilde{E}$ to elements of O with higher cost. Thus, if we sort $\tilde{E}$ and $\tilde{O}$ in non-increasing order according to costs, σ maps the first $1/\epsilon$ elements of $\tilde{E}$ to o_1, the next $1/\epsilon$ elements to o_2, etc. Since the i-th consecutive block of $1/\epsilon$ elements of $\tilde{E}$ represents the fractional mass of the i-th group of elements, the second property of the invariant follows. The first property of the invariant follows from the fact that $\frac{\|x\|_1}{\epsilon} = |\tilde{E}| = |\tilde{O}| \leq \frac{|\mathrm{OPT}_1|}{\epsilon}$. ◁

Now consider some iteration of the rounding algorithm, and suppose that the invariant holds at the beginning of the iteration. The invariant guarantees that the total fractional mass $\|\hat{x}\|_1$ is at most $|O|$ and, if we sort the fractional elements in non-increasing order according to the cost, the first unit of fractional mass can be assigned to the element o_1 with highest cost in O, the next unit of fractional mass can be assigned to the element o_2 with second-highest cost in O, etc. We will use such an assignment to argue that the invariant is preserved.

Suppose we are in Case 1, i.e., $\hat{x}_{e_1} + \hat{x}_{e_2} \leq 1$, where e_1 and e_2 are the fractional elements with the highest and second-highest cost. Let o_1 be the element of O with the highest cost. Since $\hat{x}_{e_1} + \hat{x}_{e_2} \leq 1$, it follows from the invariant that the entire fractional mass of $\hat{x}_{e_1} + \hat{x}_{e_2}$ is assigned to o_1. Since the rounding step moves fractional mass between e_1 and e_2, this property will continue to hold after the rounding step. If neither e_1 nor e_2 is rounded to 1, the updated fractional solution clearly satisfies the invariant. Therefore we may assume that one of e_1, e_2 is rounded to 1, and thus we must have had $\hat{x}_{e_1} + \hat{x}_{e_2} = 1$ before the rounding. Since o_1 is assigned a fractional mass of 1 in total, e_1 and e_2 are the only elements assigned to o_1. Therefore, after removing o_1, e_1, and e_2, the remaining fractional entries and the set $O \setminus \{o_1\}$ satisfy the invariant.

Suppose we are in Case 2, i.e., $1 < \hat{x}_{e_1} + \hat{x}_{e_2} \leq 2$, where e_1 and e_2 are the fractional elements with the highest and second-highest cost, respectively. Let o_1 and o_2 be the elements of O with the highest and second-highest cost, respectively. It follows from the invariant that the fractional mass $\hat{x}_{e_1} + \hat{x}_{e_2}$ is assigned to o_1 and o_2 as follows: the 1 unit of fractional mass assigned to o_1 is comprised of $\hat{x}_{e_1}$ from e_1 and $1 - \hat{x}_{e_2}$ from e_2, and o_2 is assigned the remaining $\hat{x}_{e_1} + \hat{x}_{e_2} - 1$ fractional mass of e_2. The rounding step either rounds e_1 to 1 by moving $1 - \hat{x}_{e_1}$ mass from e_2 to e_1 or it rounds e_2 to 1 by moving $1 - \hat{x}_{e_2}$ mass from e_1 to e_2. In the former case, after removing e_1 and o_1, the remaining fractional entries and the set $O \setminus \{o_1\}$ satisfy the invariant. Therefore we may assume that it is the latter, i.e., we round e_2 to 1 and we remove e_2 and o_2. In this case, the fractional values on the elements $e_3, e_4, \ldots$ move forward by $1 - \hat{x}_{e_2}$ to fill in the space vacated by e_2. We can also move forward their assignment to $O \setminus \{o_2\}$: e_1 remains entirely assigned to o_1 as before, and the assignment of each of the elements $e_3, e_4, \ldots$ is shifted forward. Since we remove one unit from both the total fractional mass and O, every remaining element becomes assigned to an element of $O \setminus \{o_2\}$ whose cost is at least as much as the element of O that it was previously assigned. Therefore the invariant is preserved. ◀

References

1. Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014.
2. Shaddin Dughmi, Tim Roughgarden, and Mukund Sundararajan. Revenue Submodularity. *Theory of Computing*, 8(1):95–119, 2012.
3. Ryan Gomes and Andreas Krause. Budgeted Nonparametric Learning from Data Streams. In *International Conference on Machine Learning (ICML)*, pages 391–398, 2010.
4. Stefanie Jegelka and Jeff A. Bilmes. Submodularity beyond submodular energies: Coupling edges in graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.
5. David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 137–146, 2003.
6. Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information processing letters*, 70(1):39–45, 1999.

7 Andreas Krause, Ajit Paul Singh, and Carlos Guestrin. Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. *Journal of Machine Learning Research*, 9:235–284, 2008.

8 Ariel Kulik, Hadas Shachnai, and Tami Tamir. Approximations for monotone and nonmonotone submodular maximization with knapsack constraints. *Mathematics of Operations Research*, 38(4):729–739, 2013.

9 Hui Lin and Jeff A. Bilmes. Multi-document Summarization via Budgeted Maximization of Submodular Functions. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics*, pages 912–920, 2010.

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

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

12 Yuichi Yoshida. Maximizing a Monotone Submodular Function with a Bounded Curvature under a Knapsack Constraint. *CoRR*, abs/1607.04527, 2016. `arXiv:1607.04527`.