Fast algorithms for maximizing submodular functions

Ashwinkumar Badanidiyuru * Jan Vondrák †

October 9, 2013

Abstract

There has been much progress recently on improved approximations for problems involving submodular objective functions, and many interesting techniques have been developed. However, the resulting algorithms are often slow and impractical. In this paper we develop algorithms that match the best known approximation guarantees, but with significantly improved running times, for maximizing a monotone submodular function $f : 2^{[n]} \to \mathbb{R}_+$ subject to various constraints. As in previous work, we measure the number of oracle calls to the objective function which is the dominating term in the running time.

Our first result is a simple algorithm that gives a $(1 - 1/e - \epsilon)$-approximation for a cardinality constraint using $O(\frac{n}{\epsilon} \log \frac{n}{\epsilon})$ queries, and a $1/(p + 2\ell + 1 + \epsilon)$-approximation for the intersection of a p-system and ℓ knapsack (linear) constraints using $O(\frac{n}{\epsilon^2} \log^2 \frac{n}{\epsilon})$ queries. This is the first approximation for a p-system combined with linear constraints. (We also show that the factor of p cannot be improved for maximizing over a p-system.) The main idea behind these algorithms serves as a building block in our more sophisticated algorithms.

Our main result is a new variant of the *continuous greedy algorithm*, which interpolates between the classical greedy algorithm and a truly continuous algorithm. We show how this algorithm can be implemented for matroid and knapsack constraints using $\tilde{O}(n^2)$ oracle calls to the objective function. (Previous variants and alternative techniques were known to use at least $\tilde{O}(n^4)$ oracle calls.) This leads to an $O(\frac{n^2}{\epsilon^4} \log^2 \frac{n}{\epsilon})$-time $(1 - 1/e - \epsilon)$-approximation for a matroid constraint. For a knapsack constraint, we develop a more involved $(1 - 1/e - \epsilon)$-approximation algorithm that runs in time $O(n^2(\frac{1}{\epsilon} \log n)^{poly(1/\epsilon)})$.

*Department of Computer Science, Cornell University. Email: ashwinkumarbv@gmail.com. Parts of this research have been done while A. Badanidiyuru was a research intern at IBM Almaden Research Center. A. Badanidiyuru was partially supported by NSF grant AF-0910940 and NSF grant IIS-0905467.

†IBM Almaden Research Center, San Jose, CA. E-mail: jvondrak@us.ibm.com

1 Introduction

Optimization problems involving maximization of a submodular objective function have attracted a lot of attention recently. This interest has been driven by several kinds of applications in which submodular objective functions arise naturally. Let us mention welfare maximization in combinatorial auctions, where the objective function captures the notion of utility function with diminishing returns [27, 4, 6, 8, 33]. A second application arises in the area of social networks, where the functions characterizing the influence of a subset of nodes turn out to be submodular [17, 18]. Another one is the problem of intelligent gathering of information, where sensors should be placed in order to maximize the information that can be collected from them [14, 21, 28, 23, 20, 22]. In this case, submodularity arises due to the fact that the entropy of a collection of random variables is a submodular function. Submodular functions also provide a unifying framework which captures optimization problems that have been studied earlier, such that Min Cut, Max Cut, Multiway Cut, Max k-Cover, the Generalized Assignment Problem, the Separable Assignment Problem, etc.

In these settings, the goal is to optimize a submodular function subject to certain constraints. In fact, in many cases the constraints at hand are quite simple, such as cardinality ($\max\{f(S) : |S| \leq k\}$), or a partition matroid ($\max\{f(S) : |S \cap P_i| \leq 1 \forall i\}$, where P_i are disjoint sets). In some applications, combinations of several of such constraints can arise, such as a combination of several matroid constraints and knapsack constraints ($\sum_{j \in S} c_j \leq 1$). For a long time, the only work in this area had been a sequence of papers by Fisher, Nemhauser and Wolsey [30, 12, 29]. The main result of this work is that the *greedy algorithm* provides relatively good approximation for these problems: in particular, a $(1 - 1/e)$-approximation for maximizing a monotone submodular functions subject to a cardinality constraint, and a $(k + 1)$-approximation for maximization of a monotone submodular function subject to k matroid constraints. An additional virtue of the greedy algorithm is that it is very simple and fast (quadratic running time in the straightforward implementation).

Recently, there has been a wave of effort to improve this result and extend it to more general scenarios. A $(1-1/e)$-approximation was given for the case of 1 knapsack constraint [32]. For the case of 1 matroid constraint, which is relevant particularly in the setting of combinatorial auctions, an optimal $(1-1/e)$-approximation was found in [33, 2]. For any constant number of knapsack constraints, a $(1-1/e-\epsilon)$-approximation was given in [24]. For k matroid constraints, the $(k+1)$-approximation has been improved to $k+\epsilon$ for any fixed $\epsilon > 0$ [26]. Extensions and generalizations to nonmonotone submodular functions have been developed in [7, 25, 34, 26, 16, 13, 9, 10, 3, 1].

A common trait of these algorithms is that they provide good approximations, often close to optimal, but their running time is typically far from practical. (With two exceptions: [16] gives a fast algorithm for maximizing a submodular function subject to a p-system, generalizing the monotone case from [12]; [1] gives an optimal 1/2-approximation for maximizing a submodular function without any constraints, and runs in linear time.) Let us mention some other recent advances and their limitations in terms of running time:

- The notion of *multilinear relaxation* has been used to achieve an optimal $(1-1/e)$-approximation for submodular maximization subject to a matroid constraint [2], as well as improved approximations for more general variants [24, 34, 9, 13, 3]. Nevertheless, the *continuous greedy algorithm*, the primary algorithmic tool here, has been quoted to run in time $\Theta(n^8)$ [11].
- Recently, an alternative way to derive a $(1-1/e)$-approximation for 1 matroid constraint, which does not require the multilinear relaxation, was given in [11]. It gives a $(1-1/e-\epsilon)$-approximation in time $O(nr^4\epsilon^{-3})$, where r is the rank of the matroid. The authors show that the continuous greedy algorithm would have a similar performance if adapted to give a $(1-1/e-\epsilon)$-approximation.
- For combined constraints such as multiple knapsacks, known algorithms rely on enumeration of $poly(1/\epsilon)$ items [24, 3] or a reduction to partition matroids [15], both of which lead to $\Omega(n^{poly(1/\epsilon)})$ running times. The dependence in the exponent is bad enough that these algorithms do not seem practical even for $\epsilon = \frac{1}{2}$.

Due to these issues, it appears that few of these new algorithms are fast enough to be applied in practice. In fact, it turns out that even the original greedy algorithm, which takes $O(rn)$ running time, is too slow for many applications, and researchers have sought ways to speed up the greedy algorithm as well [28, 23]. Our goal in this paper is to remedy this situation and develop algorithms that have both theoretical approximation guarantees, and a more practical running time.

Our results. In the following, n denotes the total number of elements, and r denotes the maximum cardinality of a feasible solution. We measure the complexity of an algorithm in terms of the number of oracle calls. We note that the total running time of our algorithms is not worse than $O(n)\times$ the number of oracle calls, and typically this would be the actual running time, since an oracle call involves processing a set of up to n elements. Since the earlier algorithms have been typically analyzed in terms of the number of oracle calls, we use the same benchmark.

Simple thresholding algorithms. First, we present a simple algorithm for the case of a cardinality constraint, which is faster then the classical greedy algorithm and performs at least as well. A greedy algorithm for selecting r elements uses $O(rn)$ queries. An often used version of greedy known as lazy greedy [28, 23] seems to perform much better in practice. Our algorithm can be viewed as an alternative implementation of lazy greedy which provides the same approximation and a guaranteed faster running time.

THEOREM 1.1. *There is a $(1-1/e-\epsilon)$-approximation algorithm for maximizing a monotone submodular function subject to a cardinality constraint, using $O\left(\frac{n}{\epsilon}\log\frac{n}{\epsilon}\right)$ queries.*

An extension of this algorithm gives the following result for a combination of a p-system and ℓ linear constraints.

THEOREM 1.2. *There is a $1/(p+2\ell+1+\epsilon)$-approximation algorithm for maximizing a monotone submodular function subject to an intersection of a p-system and ℓ knapsack (linear) constraints, using $O(\frac{n}{\epsilon^2}\log^2\frac{n}{\epsilon})$ queries.*

To our knowledge, this is the first known approximation for the combination of a p-system and knapsack constraints. It was known that the greedy algorithm gives a $1/p$-approximation for maximizing a linear function subject to a p-system, and a $1/(p+1)$-approximation for maximizing a monotone submodular function subject to a p-system [12]. We also give a simple proof that the factor of p is optimal for p-systems; see Section 7.

Our algorithm uses a combination of two ideas: a fast implementation of a greedy algorithm, using a decreasing value threshold, and a fixed density threshold, to deal with the knapsack constraints. The ideas used in the above two algorithm also serve as building blocks in the following result.

Accelerated continuous greedy algorithm. Our main result is a new *accelerated continuous greedy*

Constraint	Approximation	Running time	Prior running time
Cardinality	$1-1/e-\epsilon$	$O(\frac{n}{\epsilon}\log\frac{n}{\epsilon})$	$O(nr)$[30]
1 matroid	$1-1/e-\epsilon$	$O(\frac{rn}{\epsilon^4}\log^2\frac{r}{\epsilon})$	$O(nr^3\epsilon^{-3}\log r)$[11]
1 knapsack	$1-1/e-\epsilon$	$O(n^2(\frac{1}{\epsilon}\log\frac{n}{\epsilon})^{poly(1/\epsilon)})$	$O(n^5)$[32]
p-system + ℓ knapsacks	$p+2\ell+1+\epsilon$	$O(\frac{n}{\epsilon^2}\log^2\frac{n}{\epsilon})$	$---$

Figure 1: Overview of results.

algorithm for maximization of a monotone submodular function subject to a matroid constraint. The original continuous greedy algorithm [33, 2] was not optimized for running time and required roughly $\tilde{O}(n^8)$ oracle calls to the objective function. Recently, Filmus and Ward [11] developed a new algorithm, as well as a tighter analysis of the continuous greedy algorithm, which gives $\tilde{O}(r^3n\epsilon^{-3})$ running time. Hence any improvement to the running time should come by introducing new ideas.

We present three modifications to the continuous greedy algorithm that lead to a faster running time. We remark that the continuous greedy algorithm yields a fractional solution that still needs to be rounded. For the rounding stage we appeal to the swap rounding approach of [3]; although we do not develop a new rounding algorithm, the fractional solution that we produce in the optimization stage has additional structure which speeds up the swap rounding algorithm as well.

THEOREM 1.3. *There is an algorithm that achieves a $(1-1/e-\epsilon)$-approximation for maximizing a monotone submodular function subject to a matroid constraint, and requires $O(rn\epsilon^{-4}\log^2 n)$ oracle calls to the objective function.*

As a different application of the accelerated continuous greedy algorithm, we show an asymptotically faster algorithm for a single knapsack constraint.

THEOREM 1.4. *There is an algorithm that achieves a $(1-1/e-\epsilon)$-approximation for maximizing a monotone submodular function subject to a knapsack constraint, and requires $O(n^2(\frac{1}{\epsilon}\log\frac{n}{\epsilon})^{poly(1/\epsilon)})$ oracle calls to the objective function.*

Our techniques. The main ideas involved in accelerating the algorithms are the following.
Smooth interpolation between greedy and continuous greedy: While the classical greedy algorithm [12] gives a 1/2-approximation, continuous greedy [33] gives a $(1-1/e)$-approximation. Continuous greedy works by reducing the problem to maximizing linear functions. This requires it to move in steps of size $\delta = O(\epsilon/n)$. We give an algorithm which is a smooth interpolation between greedy and continuous greedy and gives a $(1-1/(1+\delta)^{1/\delta})$-approximation for any constant step size $\delta \in (0,1]$. We achieve this by sequential updates instead of parallel updates in the continuous greedy algorithm. This allows us to get a $(1-1/e-\epsilon)$-approximation with $\delta = O(\epsilon)$. When $\delta = 1$, we recover the classical greedy 1/2 approximation.
Geometrically decreasing thresholds: This idea comes from the lazy greedy heuristic, which we also use in our algorithm for a cardinality constraint. Lazy greedy can be thought of as an algorithm which maintains a continuously decreasing threshold and takes elements when their marginal value is above the threshold. We show that decreasing the threshold multiplicatively by $1-\epsilon$, and including any element with a marginal value above the threshold, we avoid the need to search for the maximal marginal value and save roughly a factor of n.
Enumeration in value space instead of subsets: The above two ideas are enough for getting our result for the matroid constraint. However, this is not enough for a knapsack constraint since we cannot round a fractional solution to a feasible solution without loss in the approximation ratio. The algorithm of [32] takes care of this by enumerating over subsets of size 3 while [24, 3] enumerates over subsets of size $poly(1/\epsilon)$. We develop an alternate way of enumerating over values of the submodular function instead of subsets of the ground set.

Figure 1 shows a summary of our results, including a comparison with previous best known algorithms giving the same approximation.

Organization. First we present our simple algorithm for a cardinality constraint in Section 3. Our algorithm for the matroid constraint appears in Section 4, and for the knapsack constraint in Section 5. The algorithm for a p-system and ℓ knapsack contraints appears in Section 6 and the lower bound for p-systems in Section 7. We state some basic definitions and facts in Section 2.

2 Preliminaries

Set notation. For compactness, we denote $A+i = A\cup\{i\}$ and $A-i = A\setminus\{i\}$.

Submodular functions. A set function $f: 2^E \to \mathbb{R}$ is submodular, if $f(A)+f(B) \geq f(A\cup B)+f(A\cap B)$ for all $A, B \subseteq E$. It is monotone, if $f(A) \subseteq f(B)$ whenever

$A \subseteq B$. In this paper, we consider only nonnegative set functions, $f : 2^E \to \mathbb{R}_+$.

Matroids. A matroid is a pair $\mathcal{M} = (E, \mathcal{I})$ where $\mathcal{I} \subseteq 2^E$ is a collection of independent sets, satisfying: (i) $A \subseteq B, B \in \mathcal{I} \Rightarrow A \in \mathcal{I}$, and (ii) $A, B \in \mathcal{I}, |A| < |B| \Rightarrow \exists i \in B \setminus A; A + i \in \mathcal{I}$.

p-systems. For a family $\mathcal{I} \subseteq 2^E$ and a set $S \subseteq E$, we call B a base of S, if B is an inclusion-wise maximal subset of S such that $B \in \mathcal{I}$. A p-system is a family $\mathcal{I} \subseteq 2^E$ such that for every $S \subseteq E$ and any two bases B_1, B_2 of S, $|B_2| \leq p|B_1|$.

It is known that for any p matroids $\mathcal{M}_1 = (E, \mathcal{I}_1), \ldots, \mathcal{M}_p = (E, \mathcal{I}_p)$, the intersection $\bigcap_{i=1}^p \mathcal{I}_i$ is a p-system. More generally, p-systems include "p-extendible families", "p-uniform matchoids" and "p-uniform matroid matching"; see [2, 26] for more details.

Knapsack constraints. By a knapsack constraint $\mathcal{K} \subseteq 2^E$, we mean a family of sets defined by a linear constraint $\mathcal{K} = \{I \subseteq E : \sum_{j \in I} c_j \leq 1\}$ for some collection of weights $(c_j : j \in E)$. (We can assume that the capacity is 1 without loss of generality.) Thus the classical knapsack problem is $\max\{\sum_{j \in I} w_j : I \in \mathcal{K}\}$.

The multilinear extension. For a function $f : 2^E \to \mathbb{R}$, we define $F(\mathbf{x}) = \mathbf{E}[f(R(\mathbf{x}))]$, where $R(\mathbf{x})$ is a random set where element i appears independently with probability $\mathbf{x}_i$.

Shorthand notation. We will use the following two shorthand notation.

- For any $S, T \subseteq E$ let $f_S(T) = f(S \cup T) - f(S)$.
- Let E_1 and E_2 be two duplicate copies of ground set E. For any $S \subseteq E_1 \cup E_2$ define contract$(S) \subseteq E$ as the set which contains $e \in E$ iff S contains at least one copy of e. Then define the extension of the monotone submodular function $f : 2^{E_1 \cup E_2} \to \mathbb{R}_+$ as $\forall S \subseteq E_1 \cup E_2, f(S) = f(\text{contract}(S))$.

We will use the following facts.

LEMMA 2.1. (COROLLARY 39.12A [31]) *Let $\mathcal{M} = (N, \mathcal{I})$ be a matroid, and $B_1, B_2 \in \mathcal{B}$ be two bases. Then there is a bijection $\phi : B_1 \to B_2$ such that for every $b \in B_1$ we have $B_1 - b + \phi(b) \in \mathcal{B}$.*

LEMMA 2.2. (THEOREM 1.1 [5]) *Let $X_1, \ldots, X_n$ be independent random variables such that for each i, $X_i \in [0, 1]$. Let $X = \sum_{i=1}^n X_i$. Then*

$$\Pr[X > (1+\epsilon)\mathbf{E}[X]] \leq e^{-\frac{\epsilon^2}{3}\mathbf{E}[X]},$$
$$\Pr[X < (1-\epsilon)\mathbf{E}[X]] \leq e^{-\frac{\epsilon^2}{2}\mathbf{E}[X]}.$$

LEMMA 2.3. (RELATIVE+ADDITIVE CHERNOFF BOUND) *Let $X_1, \ldots, X_m$ be independent random variables such that for each i, $X_i \in [0, 1]$. Let $X = \frac{1}{m}\sum_{i=1}^m X_i$ and $\mu = \mathbf{E}[X]$. Then*

$$\Pr[X > (1+\alpha)\mu + \beta] \leq e^{-\frac{m\alpha\beta}{3}},$$
$$\Pr[X < (1-\alpha)\mu - \beta] \leq e^{-\frac{m\alpha\beta}{2}}.$$

Proof. Using Lemma 2.2,

$$\begin{aligned}
&\Pr[X > (1+\alpha)\mu + \beta] \\
&\leq \min(\Pr[X > (1+\alpha)\mu], \Pr[X > \mu + \beta]) \\
&\leq \min\left(e^{-\frac{\alpha^2}{3}m\mu}, e^{-\frac{\beta^2}{3\mu^2}m\mu}\right) = e^{-\max\left(\frac{\alpha^2}{3}m\mu, \frac{\beta^2}{3\mu}m\right)} \\
&\leq e^{-\left(\frac{\alpha^2}{6}m\mu + \frac{\beta^2}{6\mu}m\right)} \leq e^{-m\sqrt{\frac{\alpha^2\beta^2}{9}}} \leq e^{-\frac{m\alpha\beta}{3}}.
\end{aligned}$$

$$\begin{aligned}
&\Pr[X < (1-\alpha)\mu - \beta] \\
&\leq \min(\Pr[X < (1-\alpha)\mu], \Pr[X < \mu - \beta]) \\
&\leq \min\left(e^{-\frac{\alpha^2}{2}m\mu}, e^{-\frac{\beta^2}{2\mu^2}m\mu}\right) = e^{-\max\left(\frac{\alpha^2}{2}m\mu, \frac{\beta^2}{2\mu}m\right)} \\
&\leq e^{-\left(\frac{\alpha^2}{4}m\mu + \frac{\beta^2}{4\mu}m\right)} \leq e^{-m\sqrt{\frac{\alpha^2\beta^2}{4}}} \leq e^{-\frac{m\alpha\beta}{2}}.
\end{aligned}$$

LEMMA 2.4. (LEMMA 3.7 [2]) *Let $X = X_1 \cup \ldots \cup X_k$, let $f : 2^X \to \mathbb{R}_+$ be a monotone submodular function, and for all $i \neq j$ we have $X_i \cap X_j = \emptyset$. Let $y \in \mathbb{R}_+^E$ such that for each X_j we have $\sum_{i \in X_j} y_i \leq 1$. If T is a random set where we sample independently from each X_i at most one random element, element j with probability y_j, then*

$$E[f(T)] \geq F(y).$$

3 Cardinality constraint

First we present a simple near-linear time algorithm for the problem $\max\{f(S) : |S| \leq r\}$, where f is a monotone submodular function. Observe that a straightforward implementation of the greedy algorithm runs in time $O(rn)$, and a factor of n is necessary just to find the most valuable element (if $r = 1$). Our goal here is to eliminate the dependence on r, while still preserving the approximation properties of the greedy algorithm. Our algorithm works as follows.

Here we prove Theorem 1.1. The running time of the algorithm is easy to check. For the approximation ratio, it is enough to prove the following claim.

CLAIM 3.1. *Let O be an optimal solution. Given a current solution S, the gain of Algorithm 1 in one step is at least $\frac{1-\epsilon}{r}\sum_{a \in O \setminus S} f_S(a)$.*

Proof. Due to submodularity the marginal values can only decrease as we add elements. Hence if the next

Algorithm 1 Cardinality Constraint

Input: $f : 2^E \to \mathbb{R}_+$, $r \in \{1, \ldots, n\}$.
Output: A set $S \subseteq E$ satisfying $|S| \leq r$.

$S \leftarrow \emptyset$.
$d \leftarrow \max_{j \in E} f(j)$
for $(w = d; w \geq \frac{\epsilon}{n} d; w \leftarrow w(1-\epsilon))$ **do**
  for all $e \in E$ **do**
    if $|S \cup \{e\}| \leq r$ and $f_S(\{e\}) \geq w$ **then**
      $S \leftarrow S \cup \{e\}$
    end if
  end for
end for
return S.

element chosen is a and the current threshold value is w, then it implies the following inequalities

$$f_S(x) = \begin{cases} \geq w & \text{if } x = a \\ \leq w/(1-\epsilon) & \text{if } x \in O \setminus (S \cup \{a\}) \end{cases}$$

The above inequalities imply that $f_S(a) \geq (1-\epsilon) f_S(x)$ for each $x \in O \setminus S$. Taking an average over these equations we get $f_S(a) \geq \frac{1-\epsilon}{|O \setminus S|} \sum_{a \in O \setminus S} f_S(a) \geq \frac{1-\epsilon}{r} \sum_{a \in O \setminus S} f_S(a)$.

Now it is straightforward to finish the proof of Theorem 1.1.

Proof. Condition on a solution $S_i = \{a_1, \ldots, a_i\}$ after i steps. By Claim 3.1, we have $f_{S_i}(a_{i+1}) \geq \frac{1-\epsilon}{r} \sum_{a \in O \setminus S_i} f_{S_i}(a)$. By submodularity, $\sum_{a \in O \setminus S_i} f_{S_i}(a) \geq f_{S_i}(O) \geq f(O) - f(S_i)$. Therefore

$$f(S_{i+1}) - f(S_i) = f_{S_i}(a_{i+1}) \geq \frac{1-\epsilon}{r}(f(O) - f(S_i)).$$

$$\begin{aligned} f(S_r) &\geq \left(1 - \left(1 - \frac{1-\epsilon}{r}\right)^r\right) f(O) \\ &\geq \left(1 - e^{-(1-\epsilon)}\right) f(O) \geq (1 - 1/e - \epsilon) f(O) \end{aligned}$$

We remark that the idea of decreasing thresholds, with additional complications, also gives an approximation algorithm for the much more general constraint of a p-system combined with ℓ knapsack constraints. The details are given in Section 6. In addition, this idea will play a role as a building block in the algorithms for a matroid or a knapsack constraint, which we turn to next.

4 Matroid constraint

Here we give a $(1-1/e-\epsilon)$-approximation algorithm for maximizing a monotone submodular function subject to a matroid constraint, using $O\left(rn\epsilon^{-4} \log^2 n\right)$ oracle calls to the objective function and the matroid independence oracle. The general outline of our algorithm follows the continuous greedy algorithm from [33, 2], with a fractional solution being built up gradually from $\mathbf{x} = 0$, and finally using randomized rounding to convert the fractional solution into an integer one (here we use the improved swap rounding procedure from [3]). We are able to achieve a significant improvement in the running time based on the following insights.

While the greedy algorithm is fast, it gives only a 1/2-approximation for a matroid constraint. The continuous greedy algorithm gives $(1-1/e)$-approximation but inherently it is more complicated. Here, we show how one can interpolate between the two via a parameter $\delta \in [0, 1]$ such that $\delta = 1$ corresponds to the greedy algorithm, and $0 < \delta < 1$ corresponds to a discretized version of the continuous greedy algorithm providing a $(1 - 1/(1+\delta)^{1/\delta})$-approximation.

Let us describe this idea in a bit more detail. The greedy algorithm iteratively adds elements to the solution such that you add element e to set S if it maximizes $f_S(e)$ and $S \cup \{e\}$ is a feasible solution. The continuous greedy algorithm from [33, 2] runs in continuous time from 0 to 1. Its implementation discretizes time and increments it in steps of size $\delta = \tilde{O}(1/n)$. In each time step it takes a feasible solution to end up with a convex combination of solutions $\mathbf{x} = \sum_{i=1}^{1/\delta} \delta \mathbf{1}_{B_i}$. When $\delta < \tilde{O}(1/n)$ the improvement by taking j^{th} feasible solution behaves like a linear function, i.e $F(\mathbf{x}_t + \delta \mathbf{1}_S) - F(\mathbf{x}_t) \approx \sum_{e \in S} F(\mathbf{x}_t + \delta 1_e) - F(\mathbf{x}_t)$. This can be proved by a simple Taylor series expansion and bounding the lower-order terms. So the problem boils down to choosing a feasible solution S which maximizes the linear function $\sum_{e \in S} F(\mathbf{x}_t + \delta 1_e) - F(\mathbf{x}_t)$. This in essence is like updating the coordinates "in parallel". We note that for the analysis to work, δ must be inverse polynomial in n.

Our new idea is an algorithm which chooses a solution at any given time step sequentially instead of choosing it in parallel. More specifically, if $B = \{e_1, e_2, \ldots, e_r\}$ is the set chosen at time step t then our sequential update rule chooses e_i which maximizes $F(\mathbf{x}_t + \delta \mathbf{1}_{\{e_1, e_2, \ldots, e_{i-1}, e_i\}}) - F(\mathbf{x}_t + \delta \mathbf{1}_{\{e_1, e_2, \ldots, e_{i-1}\}})$. In effect, we update the fractional solution and the relevant derivatives of F after each selected element instead, rather than after choosing an entire independent set. In contrast, the original continuous greedy algorithm chooses an element e_i which maximizes $F(\mathbf{x}_t + \delta \mathbf{1}_{e_i}) - F(\mathbf{x}_t)$ with $\{e_1, e_2, \ldots, e_i\} \in \mathcal{I}$. One

can easily see why our modification is a smooth interpolation between greedy and continuous greedy — for $\delta = 1$, we obtain the greedy algorithm and for $\delta \to 0$ we obtain the (truly) continuous greedy algorithm. For a fixed $\delta \in [0,1]$, we prove that the approximation ratio with a step of size δ is $1 - 1/(1+\delta)^{\frac{1}{\delta}}$ for every $\delta \in [0,1]$. This results in improving the running time to get a $(1 - 1/e - \epsilon)$-approximation for the following three reasons.
1. It reduces the number of time steps from $O(r/\epsilon)$ to $O(1/\epsilon)$.
2. It reduces the number of samples required per evaluation of the multilinear extension from $O(\frac{1}{\epsilon^2} r^2 \log n)$ to $O\left(\frac{1}{\epsilon^2} r \log n\right)$.
3. It reduces the running time of the rounding procedure, specifically swap-rounding from [3]. If swap-rounding is given a fractional solution as a convex combination of ℓ independent sets, then it runs in time ℓr^2. Since our continuous greedy produces a fractional solution which is a convex combination of $O(1/\epsilon)$ independent sets, the running time is $O(r^2/\epsilon)$.

A key trick in the analysis of this modified procedure is that if a time step begins with a fractional solution $\mathbf{x}$ and finishes with a fractional solution $\mathbf{x}' = \mathbf{x} + \epsilon \mathbf{1}_B$, we compare our gains to the partial derivatives $\frac{\partial F}{\partial x_i}$ evaluated at $\mathbf{x}'$ rather than $\mathbf{x}$. This allows us to eliminate the loss from partial derivatives depending on the elements we have already included in B when selecting the next one. The price for this is that after $1/\epsilon$ steps, our value is at least $(1 - 1/(1+\epsilon)^{1/\epsilon})OPT$ rather than $(1 - (1-\epsilon)^{1/\epsilon})OPT$, but this is only an $O(\epsilon)$ loss in the approximation factor.

4.1 The procedure for one time step In this section we give and analyze a subroutine, which will be used in the final algorithm. This subroutine takes a current fractional solution $\mathbf{x}$ and adds to it an increment corresponding to an independent set B, to obtain $\mathbf{x} + \epsilon \mathbf{1}_B$. The way we find B is similar to the decreasing-threshold algorithm that we developed in Section 3.

Notation. In the following, for $\mathbf{x} \in [0,1]^E$ we denote by $R(\mathbf{x})$ a random set that contains each element $i \in E$ independently with probability x_i. We denote $R(\mathbf{x} + \epsilon \mathbf{1}_S)$ as $R(\mathbf{x}, S)$. By $f_R(e)$, we denote the marginal value $f_R(e) = f(R \cup \{e\}) - f(R)$.

Claim 4.1. *Let O be an optimal solution. Given a fractional solution $\mathbf{x}$, the Decreasing-Threshold procedure produces a new fractional solution $\mathbf{x}' = \mathbf{x} + \epsilon \mathbf{1}_B$ such that*

$$F(\mathbf{x}') - F(\mathbf{x}) \geq \epsilon((1 - 3\epsilon) f(O) - F(\mathbf{x}')).$$

Algorithm 2 Decreasing-Threshold Procedure

Input: $f : 2^E \to \mathbb{R}_+$, $\mathbf{x} \in [0,1]^E$, $\epsilon \in [0,1]$, $\mathcal{I} \subseteq 2^E$.
Output: A set $S \subseteq E$ satisfying $S \subseteq \mathcal{I}$.

$B \leftarrow \emptyset$.
$d \leftarrow \max_{j \in E} f(j)$
for $(w = d; w \geq \frac{\epsilon}{r} d; w \leftarrow w(1-\epsilon))$ **do**
  for all $e \in E$ **do**
    $w_e(B, \mathbf{x}) \leftarrow$ estimate of $\mathbf{E}[f_{R(\mathbf{x}+\epsilon \mathbf{1}_B)}(e)]$ by averaging $\frac{r \log n}{\epsilon^2}$ random samples.
    if $B \cup \{e\} \in \mathcal{I}$ and $w_e(B, \mathbf{x}) \geq w$ **then**
      $B \leftarrow B \cup \{e\}$
    end if
  end for
end for
return B.

Proof. Assume for now that the Decreasing-Threshold procedure returns r elements, $B = \{b_1, b_2, \ldots, b_r\}$ (indexed in the order in which they were chosen). The procedure might return fewer than r elements if the threshold w drops below $\frac{\epsilon}{r} d$ before termination; in this case, we formally add dummy elements of value 0, so that $|B| = r$. Let $O = \{o_1, o_2, \ldots, o_r\}$ be an optimal solution, with $\phi(b_i) = o_i$ as specified by lemma 2.1. Additionally, let B_i denote the first i elements of B, and O_i denote the first i elements of O.

Note that from Lemma 2.3 we get that at each step, $\mathbf{E}[f_{R(\mathbf{x}+\epsilon \mathbf{1}_{B_i})}(e)]$ is evaluated to a multiplicative error of ϵ and an additive error of $\frac{\epsilon}{r} f(O)$ by averaging $\frac{1}{\epsilon^2} r \log n$ i.i.d. samples, with high probability. This implies the following bound, with high probability:

$$|w_e(B_i, \mathbf{x}) - \mathbf{E}[f_{R(\mathbf{x}, B_i)}(e)]| \leq \frac{\epsilon}{r} f(O) + \epsilon \mathbf{E}[f_{R(\mathbf{x}, B_i)}(e)]$$

When element b_i is chosen, o_i is a candidate element which could have been chosen instead of b_i. Thus, by design of the Decreasing-Threshold algorithm, we have $w_{b_i}(B_{i-1}, \mathbf{x}) \geq (1-\epsilon) w_{o_i}(B_{i-1}, \mathbf{x}) - \frac{\epsilon}{r} d$ (because either o_i is a potential candidate of value within a factor of $1 - \epsilon$ of the element we chose instead, or the procedure terminated and all remaining elements have value below $\frac{\epsilon}{r} d$). Combining the two equations, and the fact that $f(O) \geq d$, we get

$$\mathbf{E}[f_{R(\mathbf{x}, B_{i-1})}(b_i)] \geq (1-\epsilon) \mathbf{E}[f_{R(\mathbf{x}, B_{i-1})}(o_i)] - \frac{2\epsilon}{r} f(O)$$

Using the above inequality we bound the improvement

at each time step:

$$\begin{aligned}
&F(\mathbf{x}') - F(\mathbf{x}) = F(\mathbf{x} + \epsilon \mathbf{1}_B) - F(\mathbf{x}) \\
&= \sum_{i=1}^{r} (F(\mathbf{x} + \epsilon \mathbf{1}_{B_i}) - F(\mathbf{x} + \epsilon \mathbf{1}_{B_{i-1}})) \\
&= \sum_{i=1}^{r} \epsilon \frac{\partial F}{\partial x_{b_i}}\Big|_{\mathbf{x}+\epsilon \mathbf{1}_{B_{i-1}}} \\
&\geq \sum_{i=1}^{r} \epsilon \mathbf{E}[f_{R(\mathbf{x}+\epsilon \mathbf{1}_{B_{i-1}})}(b_i)] \\
&\geq \sum_{i=1}^{r} \epsilon \left((1-\epsilon)\mathbf{E}[f_{R(\mathbf{x}+\epsilon \mathbf{1}_{B_{i-1}})}(o_i)] - \frac{2\epsilon}{r} f(O) \right) \\
&\geq \epsilon \left((1-\epsilon)\mathbf{E}[f(R(\mathbf{x}') \cup O) - f(R(\mathbf{x}'))] - 2\epsilon f(O) \right) \\
&\geq \epsilon((1-3\epsilon)f(O) - F(\mathbf{x}')).
\end{aligned}$$

Here the second inequality is because o_i is a candidate element when b_i was chosen. The first and last inequalities are due to monotonicity, and the third inequality is due to submodularity.

Claim 4.2. *The Decreasing-Threshold procedure runs in time* $O\left(\frac{1}{\epsilon^3} nr \log^2 \frac{n}{\epsilon}\right)$

Proof. It is simple to note that the running time is a product of the number of iterations in the outer loop ($\frac{1}{\epsilon} \log \frac{n}{\epsilon^2}$), the number of iterations in the inner loop (n), and the number of samples per evaluation of F, which is $\frac{1}{\epsilon^2} r \log n$.

4.2 Final algorithm In this section we give the complete algorithm and its analysis.

Algorithm 3 Accelerated Continuous Greedy

Input: $f : 2^E \to \mathbb{R}_+$, $\mathcal{I} \subseteq 2^E$.
Output: A set $S \subseteq E$ satisfying $S \in \mathcal{I}$.

1: $\mathbf{x} \leftarrow \mathbf{0}$.
2: **for** $(t \leftarrow \epsilon;\ t \leq 1;\ t \leftarrow t + \epsilon)$ **do**
3: $\quad B \leftarrow$ Decreasing-Threshold$(f, \mathbf{x}, \epsilon, \mathcal{I})$.
4: $\quad \mathbf{x} \leftarrow \mathbf{x} + \epsilon \cdot \mathbf{1}_B$.
5: **end for**
6: $S \leftarrow$ Swap-Rounding$(\mathbf{x}, \mathcal{I})$.
7: **return** S.

Claim 4.3. *The algorithm has an approximation ratio of* $1 - 1/e - \epsilon$.

Proof. Define $\Omega = (1 - 3\epsilon)f(O) = (1 - 3\epsilon)OPT$. Substituting this in the result of claim 4.1, we get

$$F(\mathbf{x}(t+\epsilon)) - F(\mathbf{x}) \geq \epsilon(\Omega - F(\mathbf{x}(t+\epsilon))).$$

Rephrasing the equation we get

$$\Omega - F(\mathbf{x}(t+\epsilon)) \leq \frac{\Omega - F(\mathbf{x}(t))}{1+\epsilon}.$$

Now applying induction to this equation, we obtain

$$\Omega - F(\mathbf{x}(t)) \leq \frac{\Omega}{(1+\epsilon)^{t/\epsilon}}.$$

Substituting $t = 1$ and rewriting the equation we get the desired approximation ratio:

$$\begin{aligned}
F(\mathbf{x}(1)) &\geq \left(1 - \frac{1}{(1+\epsilon)^{1/\epsilon}}\right) \Omega \\
&= \left(1 - \frac{1}{(1+\epsilon)^{1/\epsilon}}\right) (1 - 3\epsilon)\, OPT \\
&= (1 - 1/e - O(\epsilon))OPT.
\end{aligned}$$

Claim 4.4. *The algorithm makes* $O\left(\frac{1}{\epsilon^4} nr \log^2 \frac{n}{\epsilon}\right)$ *oracle calls to the submodular function oracle and makes* $O\left(\frac{1}{\epsilon^2} n \log \frac{n}{\epsilon} + \frac{1}{\epsilon} r^2\right)$ *oracle calls to the matroid independence oracle.*

Proof. First note that the oracle to the submodular function is called only by the continuous greedy algorithm and not by the rounding algorithm. Hence the total number of oracle calls to the submodular function is equal the number of time steps multiplied with the number of oracle calls per iteration.

$$\begin{aligned}
\text{Submodular oracle calls} &\leq \epsilon^{-1} \cdot O\left(rn\epsilon^{-3} \log^2 \frac{n}{\epsilon}\right) \\
&= O\left(rn\epsilon^{-4} \log^2 \frac{n}{\epsilon}\right)
\end{aligned}$$

The number of oracle calls to the independence oracle is divided into two parts, one due to continuous greedy procedure and another due to rounding process.

$$\begin{aligned}
\text{Independence oracle calls} &\leq \epsilon^{-1} \cdot O\left(n\epsilon^{-1} \log \frac{n}{\epsilon}\right) \\
&\leq O\left(n\epsilon^{-2} \log \frac{n}{\epsilon}\right)
\end{aligned}$$

The swap rounding proceeds as follows. It takes the fractional solution $\mathbf{x}$ and represents it as a convex combination of independent sets. Then it begins merging the independent sets while maintaining the property of them being independent. Hence the number of oracle calls is number of independent sets times the calls per merge operation.

$$\begin{aligned}
&\text{Independence oracle calls in rounding} \\
&= \text{Number of independent sets} \cdot \text{Calls per merge} \\
&\leq O\left(\epsilon^{-1} r^2\right).
\end{aligned}$$

5 Knapsack constraint

In this section we discuss the main ideas behind an algorithm for maximizing a monotone submodular function subject to knapsack constraint which runs in time $O\left(n^2 \cdot \left(\frac{\log(n)}{\epsilon}\right)^{\frac{1}{\epsilon^8}}\right)$ and has approximation ratio $1 - 1/e - \epsilon$. This is an asymptotic improvement over the current best running time of n^5 from [32] (even though not a practical improvement). Without loss of generality assume that the knapsack constraint has capacity 1.

The n^5-time algorithm from [32, 19] works as follows. It enumerates subsets S of size at most 3 and given S it adds other elements to S greedily based on marginal value to cost ratio while the budget constraint is satisfied. We can immediately improve its running time by enumerating sets of smaller cardinality while resulting in a loss in approximation ratio. But achieving an asymptotically faster $1 - 1/e - \epsilon$ approximation seems challenging. We are able to achieve this using a modification of continuous greedy. The three main ideas behind the new algorithm are summarized below.

- Interpolation between greedy and continuous greedy.
- Geometrically decreasing thresholds.
- **Enumeration in value space instead of subsets**

Since the first two ideas are in common with Section 4 we focus on the third idea here. The primary reason the above two ideas are not enough for knapsack constraint is that when there are "large cost" items, one cannot round a fractional solution to a feasible solution without losing significantly in the approximation ratio. The combinatorial algorithm from [32] (not based on continuous greedy) takes care of large items by enumerating over subsets of size 3. This is the reason for a blow-up of roughly n^3 in its running time. We develop a new way of enumerating over values that the submodular function takes, instead of subsets of the ground set. Roughly speaking we have $poly(1/\epsilon)$ "large cost" items. The submodular function can potentially take $\frac{1}{\epsilon}\log n$ different values with respect to each item. Since we enumerate over all of these, it results in a factor of $\left(\frac{1}{\epsilon}\log n\right)^{poly(1/\epsilon)}$ instead of n^3.

To show the new idea of *Enumeration in value space instead of subsets* we first consider a toy problem where the optimal solution has at most 2 elements. For this toy problem we give a (near)linear time algorithm with approximation ratio $1 - 1/e - \epsilon$ in subsection 5.1. The general case for knapsack constraint is dealt with in subsection 5.2

5.1 Optimal solution with two items In this subsection we illustrate the idea of enumeration in value space on a much simpler special case of knapsack constraint. We consider here the case when optimal solution has at most two elements and we want to design a (near)linear time algorithm.

Let us first consider a special case when we know that the optimal solution contains two items, one with cost $1/2+\epsilon$ and another with cost $1/2-\epsilon$. Note that in such a case we can trivially find the optimal solution in n^2 oracle calls. But the problem becomes more interesting if we restrict the algorithm to make at most $\tilde{O}(n)$ oracle calls. It is simple to see that this special case can be cast as maximizing f over a partition matroid constraint with 2 parts, one with items of cost $1/2 + \epsilon$ and another with items of cost $1/2 - \epsilon$. Then we can use our algorithm from section 4 to get a $1 - 1/e - \epsilon$ approximation with $\tilde{O}(n)$ oracle calls. This is what gave us the idea of using continuous greedy to improve the (asymptotic) running time of the general problem.

Next, consider a slightly more general version of the problem where the optimal solution has two items. Now it is no longer clear how to reduce it to a matroid constraint or invoke continuous greedy (along with a suitable rounding procedure). One of the main ideas here is that we do an enumeration on the values discretized in a specific way instead of enumeration on subsets of items like most other algorithms for a knapsack constraint. This will also help us to force the continuous greedy to get a fractional solution which "mimics" a partition matroid.

Algorithm overview: Let the optimal solution be $O = \{o_1, o_2\}$. Our main algorithm 4 guesses a set of values$(\forall t, w_t^1, w_t^2)$ and sends it to the continuous greedy algorithm 5. These values represent guesses for marginal values of o_1, o_2 with respect to the fractional solution at any given time step t. The continuous greedy algorithm first duplicates the set of items E to create two copies E_1 and E_2. Then it constrains the continuous greedy algorithm at each time step to pick a solution $\{e_1, e_2\}$ to satisfy the following two properties.

- $e_1 \in E_1$ and $e_2 \in E_2$. Note that since E_1 and E_2 are copies of E this does not change the optimal solution.

- $c(e_1) \leq c(o_1)$ and $c(e_2) \leq c(o_2)$. Since the algorithm does not know the cost of items in the optimal solution it cannot force this property on the solution. However, it makes sure that there exists at least one set of guessed values$(\forall t, w_t^1, w_t^2)$ which satisfy this property. We will design such a set of guess values in claim 5.2.

Further the continuous greedy algorithm updates the vectors $\mathbf{y}_1 = \mathbf{y}_1 + \epsilon \mathbf{1}_{e_1}$ and $\mathbf{y}_2 = \mathbf{y}_2 + \epsilon \mathbf{1}_{e_2}$ at each time step along with some bookkeeping. At the end the algorithm does a simple independent rounding to convert to a feasible solution.

Similarity to partition matroid: Note that if the above two bullet points hold, then any $\{e', e''\}$ with $e' \in P_1 = \{e | \mathbf{y}_1(e) > 0\}$ and $e'' \in P_2 = \{e | \mathbf{y}_2(e) > 0\}$ form a solution which satisfies the capacity. Here P_1, P_2 behave exactly like 2 parts of a partition matroid constraint.

Algorithm 4 Knapsack, 2 item opt

Input: $f : 2^E \to \mathbb{R}_+$, cost function $c : E \to \mathbb{R}_+$.
Output: A set $S \subseteq E$ satisfying $c(S) \leq 1$.

$d \leftarrow \max_{j \in E} f(j)$
Guess-set $\leftarrow \{d, d(1-\varepsilon), d(1-\varepsilon)^2, \ldots, \varepsilon^2 d/n\} \cup \{0\}$
Time-set $\leftarrow \{\epsilon, 2\epsilon, 3\epsilon, \ldots, 1\}$
for all $t \in$ Time-set and $w_t^1, w_t^2 \in$ Guess-set **do**
  $S =$ Guessing-Continuous-Greedy(f, w, c)
  $R = R \cup \{S\}$
end for
return $\max_{S \in R} f(S)$

Algorithm 5 Guessing-Continuous-Greedy (GCG)

Input: $f : 2^E \to \mathbb{R}_+$, cost function $c : E \to \mathbb{R}_+$, guess values $w \in \mathbb{R}_+^{1/\epsilon \times 2}$
Output: A set $S \subseteq E$.

$E_1, E_2 \leftarrow E$ (Duplicate the items).
Define $f : 2^{E_1 \cup E_2} \to \mathbb{R}_+$ as $f(S_1, S_2) = f(S_1 \cup S_2)$.
$\mathbf{x} \leftarrow \overrightarrow{0} \in [0,1]^{E_1 \cup E_2}$, $\mathbf{y}_1 \leftarrow \overrightarrow{0} \in [0,1]^{E_1}$, $\mathbf{y}_2 \leftarrow \overrightarrow{0} \in [0,1]^{E_2}$
for $(t \leftarrow \epsilon; t \leq 1; t \leftarrow t + \epsilon)$ **do**
  for $(i \leftarrow 1; i \leq 2; i \leftarrow i + 1)$ **do**
    For each $e \in E_i$ compute marginal$(\mathbf{x}, e) = \mathbf{E}[f_{R(\mathbf{x})}(e)]$ in time $O\left(2^{\frac{2}{\epsilon}}\right)$.
    Let $e = \text{argmin}\{c(e) | e \in E_i, \text{marginal}(\mathbf{x}, e) \geq w_t^i\}$
    $\mathbf{x} \leftarrow \mathbf{x} + \epsilon \mathbf{1}_e$.
    $\mathbf{y}_i \leftarrow \mathbf{y}_i + \epsilon \mathbf{1}_e$.
  end for
end for
Let $U(\mathbf{y}_i)$ denote one random element from E_i, element j with probability $\mathbf{y}_i(j)$.
$S \leftarrow U(\mathbf{y}_1) \cup U(\mathbf{y}_2)$
if $c(S) > 1$ **then**
  $S \leftarrow \emptyset$.
end if
return S.

Claim 5.1. *Algorithm 4 has running time* $O\left(n \cdot \left(\frac{\log(n)}{\epsilon}\right)^{\frac{2}{\epsilon}}\right)$.

Proof. The running time of Guessing-Continuous-Greedy is a product of number of iterations in the outer loop, number of iterations in the inner loop, number of elements in the ground set and number of evaluations of f per evaluation of F.

$$\text{Running time of GCG} = \frac{1}{\epsilon} \cdot 2 \cdot n \cdot 2^{\frac{2}{\epsilon}} = O\left(n \cdot \frac{2^{\frac{2}{\epsilon}}}{\epsilon}\right).$$

Now the total running time is the number of different possible choices for each of w_t^1, w_t^2 times the running time of the Guessing-Continuous-Greedy.

$$\begin{aligned}\text{Total running time} &= \left(\frac{\log(n)}{\epsilon}\right)^{\frac{2}{\epsilon}} \cdot O\left(n \cdot \frac{2^{\frac{2}{\epsilon}}}{\epsilon}\right) \\ &= O\left(n \cdot \left(\frac{\log(n)}{\epsilon}\right)^{\frac{2}{\epsilon}}\right).\end{aligned}$$

Claim 5.2. *Algorithm 4 has an approximation ratio of* $1 - 1/e - \epsilon$.

Proof. The proof of approximation goes by showing specific values for w_t^1, w_t^2 for which the Guessing-Continuous-Greedy achieves a $1 - 1/e - \epsilon$ approximation. Let $\mathbf{x}(t)$ be the fractional solution at the end of time step t. Define w_t^1 as the largest value in Guess-Set smaller than $\mathbf{E}[f_{R(\mathbf{x}(t))}(o_1)]$. Then let $e_t^1 \in E_1$ be the choice of the continuous greedy. Then by the choice of e_t^1 we have the following two inequalities.

- $\mathbf{E}[f_{R(\mathbf{x}(t))}(e_t^1)] \geq w_t^1 \geq (1-\epsilon)\mathbf{E}[f_{R(\mathbf{x}(t))}(o_1)]$. This follows from the fact that o_1 is a candidate for e_t^1.
- $c(e_t^1) \leq c(o_1)$. This is because both e_t^1 and o_1 are candidate elements to be chosen and we choose the one with minimum cost.

Similarly, define w_t^2 as the largest value in Guess-Set smaller than $\mathbf{E}[f_{R(\mathbf{x}(t)+\epsilon 1_{e_t^1})}(o_2)]$.

- $\mathbf{E}[f_{R(\mathbf{x}(t)+\epsilon 1_{e_t^1})}(e_t^2)] \geq w_t^2 \geq (1-\epsilon)\mathbf{E}[f_{R(\mathbf{x}(t)+\epsilon 1_{e_t^1})}(o_2)]$. This follows from the fact that o_2 is a candidate for e_t^2.
- $c(e_t^2) \leq c(o_2)$. This is because both e_t^2 and o_2 are candidate elements to be chosen and we choose the one with minimum cost.

For the set of w_t^1, w_t^2 thus defined we have the following property. For every $e_t^1 \in E_1$ and $e_t^2 \in E_2$ with

$y_1(e_t^1) > 0$ and $y_2(e_t^2) > 0$, $\{e_t^1, e_t^2\}$ forms a feasible solution. Additionally the equations allow us to bound the increase at each time step.

$$\begin{aligned}
&F(\mathbf{x}(t+\epsilon)) - F(\mathbf{x}(t)) = F(\mathbf{x}(t) + \epsilon \mathbf{1}_{\{e_t^1, e_t^2\}}) - F(\mathbf{x}(t)) \\
&= F(\mathbf{x}(t) + \epsilon \mathbf{1}_{\{e_t^1, e_t^2\}}) - F(\mathbf{x}(t) + \epsilon \mathbf{1}_{e_t^1}) \\
&\qquad + F(\mathbf{x}(t) + \epsilon \mathbf{1}_{e_t^1}) - F(\mathbf{x}(t)) \\
&\geq \epsilon(\mathbf{E}[f_{R(\mathbf{x}(t)+\epsilon 1_{e_t^1})}(e_t^2)] + \mathbf{E}[f_{R(\mathbf{x}(t))}(e_t^1)]) \\
&\geq \epsilon(1-\epsilon)(\mathbf{E}[f_{R(\mathbf{x}(t)+\epsilon 1_{e_t^1})}(o_2)] + \mathbf{E}[f_{R(\mathbf{x}(t))}(o_1)]) \\
&\geq \epsilon(1-\epsilon)(\mathbf{E}[f_{R(\mathbf{x}(t+\epsilon)+\epsilon 1_{o_1})}(o_2)] + \mathbf{E}[f_{R(\mathbf{x}(t+\epsilon))}(o_1)]) \\
&= \epsilon(1-\epsilon)(\mathbf{E}[f(R(\mathbf{x}(t+\epsilon) \cup O)) - f(R(\mathbf{x}(t+\epsilon)))]) \\
&\geq \epsilon(1-\epsilon)(f(O) - F(\mathbf{x}(t+\epsilon))).
\end{aligned}$$

Now we proceed as in Section 4. Rephrasing the inequality above, we get $f(O) - F(\mathbf{x}(t+\epsilon)) \leq \frac{f(O) - F(\mathbf{x}(t))}{1+\epsilon(1-\epsilon)}$. Then by induction we can prove

$$f(O) - F(\mathbf{x}(t)) \leq \frac{1}{(1+\epsilon(1-\epsilon))^{\frac{t}{\epsilon}}} f(O).$$

Substituting $t = 1$ and rephrasing once again

$$\begin{aligned}
F(\mathbf{x}(1)) &\geq \left(1 - \frac{1}{(1+\epsilon(1-\epsilon))^{\frac{1}{\epsilon}}}\right) f(O) \\
&\geq (1 - 1/e - O(\epsilon)) f(O).
\end{aligned}$$

By Lemma 2.4 we get that the rounding in step 12 of algorithm 5 results in a solution S such that $\mathbf{E}[f(S)] \geq F(\mathbf{x}(1)) \geq (1 - 1/e - \epsilon) f(O)$.

5.2 General case of knapsack constraint In this section we consider the general case of a knapsack constraint without any restrictions. The final algorithm requires several ideas developed in this paper.

Define the set of "small items", items which satisfy $f(e) \leq \epsilon^6 f(O)$ and $c(e) \leq \epsilon^4$ (we only need an estimate for $f(O)$, so we can discretize the values in between $\max_{i \in E} f(i)$ and $n \max_{i \in E} f(i)$ into $\log n/\epsilon$ values and check each one of them). We call the set of small items E_s. Define the set of "large items" as $E \backslash E_s$. Let the optimal solution be $O = O_l \cup O_s$ where $O_l = \{o_1, o_2, \ldots, o_{1/\epsilon^6}\} = O \backslash E_s$ is the set of large items in O and O_s is the set of small items in O.

The basic ingredient of our algorithm is a variant of continuous greedy. Algorithm 7 just guesses a set of values represented as w_t^i in a brute force manner and sends to continuous greedy algorithm (Algorithm 8). The continuous greedy at each time step updates the fractional solution $\mathbf{x}$ with a feasible solution S to get $\mathbf{x} + \epsilon \mathbf{1}_S$ in the following two steps.

• **Handling large items** At any given time step the continuous greedy handles the large items similar to algorithm from Section 5.1.

• **Handling small items** Handling small items in a given time step is much trickier. This is because unlike large items we can have up to $O(n)$ small items. Hence we cannot afford to have a separate part for each small item in the partition matroid structure (as our running time is proportional to roughly $\log n/\epsilon$ raised to the number of parts). At a high level our algorithm handles all the small items as a single part and is essentially a greedy algorithm with respect to the marginal value / cost ratios.

Once we compute the fractional solution, we need to round it to an integral one. Although our rounding is based on known techniques, it is somewhat non-trivial to use them in our setting. We explain the rounding step in Section 5.2.3.

5.2.1 Subroutine for handling small items We give a sketch of the subroutine here. A formal implementation can be found in Algorithm 6. The subroutine here is called at each time step and it updates the current fractional solution $\mathbf{x}$ with a set $S_s \subseteq E_s$ of small items. Furthermore we desire such a solution to have the following properties.

- $F(\mathbf{x} + \epsilon \mathbf{1}_{S_s}) - F(\mathbf{x}) \geq \epsilon(1 - O(\epsilon)) \mathbf{E}[f_{R(\mathbf{x})}(O_s)]$,
- $c(S_s) \leq c(O_s)$.

To this end we have a guess W of the value of $\mathbf{E}[f_{R(\mathbf{x})}(O_s)]$ which is sent to the subroutine. At a high level our algorithm achieves the desired set of properties (when our guessed value is correct) by just running a greedy algorithm on the marginal value / cost ratios. The only problem with a naive implementation of such an algorithm is that it requires n^2 evaluations of F which results in a running time of $\tilde{O}(n^3)$. We decrease this to $\tilde{O}(n^2)$ and solve several other issues by the following set of ideas.

• A key idea is that the marginal value of an item need not be reevaluated unless its value decreases by more than a $1 - \epsilon$ factor. Further if an item's value gets reevaluated more than $2 \log n/\epsilon$ times then we can throw it away from the current time step as it no longer can add a significant value to our solution. This results in needing us to evaluate F only $O(n \log n/\epsilon)$ times. We implement this by managing a sorted list Q.

• We achieve $c(S_s) \leq c(O_s)$ by stopping the algorithm slightly before it reaches our target guess value.

Algorithm 6 Decreasing-Density

Input: A monotone submodular function $f : 2^E \to \mathbb{R}_+$, cost function $c : E \to \mathbb{R}_+$, guess value W, current fractional value $\mathbf{x}$.

Output: A set $S \subseteq E$.

$S \leftarrow \emptyset$
$Q \leftarrow \emptyset$ (sorted list)
for all $e \in E$ **do**
  $w_e(\emptyset, x) \leftarrow$ estimate of $\mathbf{E}[f_{R(\mathbf{x})}(e)]$ by averaging $\frac{n \log n}{\epsilon^3}$ random samples..
  Let $v(e) = \max\{(1-\epsilon)^t \mid t \in \text{Integers}, v(e) \leq w_e(\emptyset, x)/c(e)\}$.
  Add $(e, v(e))$ to Q.
end for
Sort Q in decreasing order of $v(e)$.
while Q is non-empty **do**
  Let $(e, v(e))$ be the element at the top of Q. Delete it from Q.
  $w_e(S, x) \leftarrow$ estimate of $\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_S)}(e)]$ by averaging $\frac{n \log n}{\epsilon^2}$ random samples.
  Let $v'(e) = \max\{(1-\epsilon)^t \mid t \in \text{Integers}, v'(e) \leq w_e(S, x)/c(e)\}$.
  if $v'(e) \geq (1-\epsilon)v(e)$ **then**
    $S \leftarrow S \cup \{e\}$
  else if e has been reinserted back into Q less than $\log(n)/\epsilon$ times **then**
    Reinsert $(e, v'(e))$ into Q.
  end if
  TotalMarginal $\leftarrow$ estimate of $F(\mathbf{x} + \epsilon\mathbf{1}_S) - F(\mathbf{x})$ by averaging $\frac{n \log n}{\epsilon^3}$ random samples.
  if TotalMarginal $\geq (1 - 10\epsilon)W$ **then**
    break loop;
  end if
end while
return S.

Then we proceed to show that if this condition were violated, then we would have stopped the algorithm at a later point.

• To make the analysis work we once again use the trick from Section 4 of comparing the gains to partial derivatives evaluated at $\mathbf{x}' = \mathbf{x} + \epsilon\mathbf{1}_{S_s}$ instead of at $\mathbf{x}$. This will also require some careful counting.

CLAIM 5.3. *If there exists a set T such that $(1+\epsilon)W \geq \epsilon\mathbf{E}[f_{R(\mathbf{x})}(T)] \geq W$ and $W \geq \epsilon^2 f(O)$ then Algorithm 6 finds a set S such that $F(\mathbf{x}+\epsilon\mathbf{1}_S) - F(\mathbf{x}) \geq (1-12\epsilon)W$. Additionally we will have that $c(S) \leq c(T)$. (Here T denotes a proxy for O_s when W is appropriately defined.)*

Proof. From lemma 2.3 we get that $F(\mathbf{x}+\epsilon\mathbf{1}_S) - F(\mathbf{x})$ is evaluated to a multiplicative error of ϵ and an additive error of $\epsilon^3 f(O)$. By the stopping rule we stop when TotalMarginal $\geq (1 - 10\epsilon)W$. Hence when we stop, we have

$$\begin{aligned} F(\mathbf{x} + \epsilon\mathbf{1}_S) - F(\mathbf{x}) &\geq (1-\epsilon)\text{TotalMarginal} - \epsilon^3 f(O) \\ &\geq (1-\epsilon)(1-10\epsilon)W - \epsilon W \geq (1-12\epsilon)W. \end{aligned}$$

Now we just need to prove that $c(S) \leq c(T)$. To prove it first we prove an upper bound on $F(\mathbf{x}+\epsilon\mathbf{1}_S) - F(\mathbf{x})$. Note that by the stopping condition we stop when Total-Marginal $\geq (1 - 10\epsilon)W$. Hence Total-Marginal $\leq (1 - 10\epsilon)W + \max_{e \in E_s} f(e) \leq (1 - 10\epsilon)W + \epsilon^3 f(O) \leq (1 - 9\epsilon)W$. Now by Lemma 2.3 we have that

$$\begin{aligned} F(\mathbf{x} + \epsilon\mathbf{1}_S) - F(\mathbf{x}) &\leq (1+\epsilon)\text{TotalMarginal} + \epsilon^3 f(O) \\ &\leq (1+\epsilon)(1-9\epsilon)W + \epsilon W \leq (1-7\epsilon)W. \end{aligned}$$

Hence we get that $F(\mathbf{x}+\epsilon\mathbf{1}_S) - F(\mathbf{x}) \leq (1-7\epsilon)W$. Now we prove $c(S) \leq c(T)$ by contradiction. Assume $c(S) > c(T)$; then we show that $F(\mathbf{x}+\epsilon\mathbf{1}_S) - F(\mathbf{x}) \geq (1-4\epsilon)W$ which is a contradiction.

Let $S = \{s_1, s_2, \ldots, s_\ell\}$ be the elements in the order they are chosen. Sort $T = \{t_1, t_2, \ldots, t_k\}$ in the order of decreasing $\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_S)}(t_i)]/c(t_i)$. Consider any number $0 \leq \zeta \leq c(T)$. Let $N(\zeta) = \min\{j : \sum_{i=1}^{j} c(s_i) \geq \zeta\}$. Similarly let $M(\zeta) = \min\{j : \sum_{i=1}^{j} c(t_i) \geq \zeta\}$. Then we show that

$$\frac{\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_{S_{N(\zeta)-1}})}(s_{N(\zeta)})] + \frac{\epsilon^3}{n} f(O)}{c(s_{N(\zeta)})} \geq (1-\epsilon)\frac{\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_S)}(t_{M(\zeta)})] - \frac{\epsilon^3}{n} f(O)}{c(t_{M(\zeta)})}.$$

The above equation is true because when $s_{N(\zeta)}$ was chosen, some element from $\{t_1, t_2, \ldots, t_{M(\zeta)}\}$ was a

candidate which could have been chosen instead. Now we can bound the total gain from S.

$$
\begin{aligned}
&F(\mathbf{x}+\epsilon\mathbf{1}_S) - F(\mathbf{x}) \\
=&\sum_{i=1}^{\ell}(F(\mathbf{x}+\epsilon\mathbf{1}_{S_i}) - F(\mathbf{x}+\epsilon\mathbf{1}_{S_{i-1}})) \\
\geq&\epsilon\sum_{i=1}^{\ell}\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_{S_{i-1}})}(s_i)] \\
=&\epsilon\int_0^{c(S)}\frac{\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_{S_{N(\zeta)-1}})}(s_{N(\zeta)})]}{c(s_{N(\zeta)})}\,\mathrm{d}\zeta \\
\geq&\epsilon\int_0^{c(T)}\frac{\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_{S_{N(\zeta)-1}})}(s_{N(\zeta)})]}{c(s_{N(\zeta)})}\,\mathrm{d}\zeta \\
\geq&(\epsilon-\epsilon^2)\int_0^{c(T)}\Big(\frac{\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_S)}(t_{M(\zeta)})] - \frac{\epsilon^3}{n}f(O)}{c(t_{M(\zeta)})} \\
&\qquad\qquad - \frac{\frac{\epsilon^3}{n}f(O)}{c(s_{N(\zeta)})}\Big)\,\mathrm{d}\zeta \\
=&\sum_{t_i\in T}(\epsilon-\epsilon^2)\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_S)}(t_i)] - 2\epsilon^3 f(O) \\
\geq&\epsilon(1-\epsilon)\left(\mathbf{E}[f(R(\mathbf{x}+\epsilon\mathbf{1}_S)\cup T) - f(R(\mathbf{x}+\epsilon\mathbf{1}_S))\right]) \\
&\qquad - 2\epsilon^3 f(O) \\
\geq&\epsilon(1-\epsilon)\left(\mathbf{E}[f(R(\mathbf{x})\cup T)] - F(\mathbf{x}+\epsilon\mathbf{1}_S)\right) - 2\epsilon^3 f(O) \\
\geq&(1-\epsilon)\left(W - \epsilon(F(\mathbf{x}+\epsilon\mathbf{1}_S) - F(\mathbf{x}))\right) - 2\epsilon W.
\end{aligned}
$$

Rephrasing the equation we get $F(\mathbf{x}+\epsilon\mathbf{1}_S)-F(\mathbf{x}) \geq (1-3\epsilon)W/(1+\epsilon) \geq (1-4\epsilon)W$ which is a contradiction.

CLAIM 5.4. *Algorithm 6 runs in time* $O\left(\frac{1}{\epsilon^4}n^2\log^2 n\right)$.

Proof. Note that each item is reinserted back into the sorted list at most $\log(n)/\epsilon$ times. Hence the running time is at most the product of the total number of items (n), the maximum number of times an item is reinserted ($\log(n)/\epsilon$), and the number of samples per evaluation ($n\log(n)/\epsilon^3$).

5.2.2 Complete algorithm The main algorithm, Algorithm 7, is similar to the main algorithm from Section 5.1 in the sense that it guesses a set of values and sends it to the continuous greedy algorithm (Algorithm 8). The continuous greedy algorithm at each time step finds a feasible solution $S = S_s \cup S_l$ as follows. Let $S_l = \{s_1, s_2, \ldots, s_{1/\epsilon^6}\}$.

• **Handling large items** At any given time step the continuous greedy handles the large items similar to the algorithm from Section 5.1. We need the fact that each item $s_i \in S_l$ satisfies the following constraints.

Algorithm 7 Knapsack, General Case

Input: $f : 2^E \to \mathbb{R}_+$, cost function $c : E \to \mathbb{R}_+$.
Output: A set $S \subseteq E$ satisfying $c(S) \leq 1$.

$d \leftarrow n \cdot \max_{j\in E} f(j)$
Guess-set $\leftarrow \{d, d(1-\varepsilon), d(1-\varepsilon)^2, \ldots, \varepsilon^2 d/n^2\} \cup \{0\}$
Time-set $\leftarrow \{\epsilon, 2\epsilon, 3\epsilon, \ldots, 1\}$
for all $t \in$ Time-set and $w_t^1, w_t^2, \ldots, w_t^{1/\epsilon^6}, W_t \in$ Guess-set **do**
  $S =$ Greedy-With-Guessing(f, c, w, W)
  $R = R \cup \{S\}$
end for
return $\max_{S\in R} f(S)$.

Algorithm 8 Greedy-With-Guessing

Input: $f : 2^E \to \mathbb{R}_+$, cost function $c : E \to \mathbb{R}_+$, guessed values $w \in \mathbb{R}_+^{1/\epsilon\times 1/\epsilon^6}, W \in \mathbb{R}_+^{1/\epsilon}$.
Output: A set $S \subseteq E$.

$E_1, E_2, \ldots, E_{1/\epsilon^6}, E_s \leftarrow E$ (Duplicate the items).
$\mathbf{x} \leftarrow \overrightarrow{0} \in [0,1]^{\cup E_i}$
For each E_i, $\mathbf{y}_i \leftarrow \overrightarrow{0} \in [0,1]^{E_i}$
$\mathbf{z} \leftarrow \overrightarrow{0} \in [0,1]^{E_s}$
$S \leftarrow \emptyset$
for $(t \leftarrow \epsilon; t \leq 1; t \leftarrow t+\epsilon)$ **do**
  for $(j \leftarrow 1; j \leq 1/\epsilon^6; j \leftarrow j+1)$ **do**
    For each $e \in E$ compute marginal$(\mathbf{x}, e) = \mathbf{E}[f_{R(\mathbf{x})}(e)]$ by averaging $\frac{n\log n}{\epsilon^2}$ samples.
    Find $e \in E$ such that marginal$(\mathbf{x}, e) \geq w_t^j$ and $c(e)$ is minimized.
    $\mathbf{x} \leftarrow \mathbf{x} + \epsilon\mathbf{1}_e$.
    $\mathbf{y}_j \leftarrow \mathbf{y}_j + \epsilon\mathbf{1}_e$
  end for
  $V \leftarrow$ Decreasing-Density$(f, \mathbf{x}, W_t)$
  $\mathbf{z} \leftarrow \mathbf{z} + \epsilon\mathbf{1}_V$
  $\mathbf{x} \leftarrow \mathbf{x} + \epsilon\mathbf{1}_V$
end for
$S \leftarrow$ Rounding-Algorithm$(\mathbf{x}, \mathbf{y}_1, \ldots, \mathbf{y}_{1/\epsilon^6}, \mathbf{z})$
if $c(S) > 1$ **then**
  $S \leftarrow \emptyset$
end if
return S.

- $c(s_i) \leq c(o_i)$,
- $\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_{S_{i-1}})}(s_i)] \geq (1-O(\epsilon))\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_{S_{i-1}})}(o_i)]$.

In Claim 5.6 we design a set of guess values (w) for which the above two properties are satisfied.

• **Handling small items** The subroutine to handle small items is described in Section 5.2.1. We need the set returned by the subroutine to satisfy the following constraints.
- $c(S_s) \leq c(O_s)$,
- $F(\mathbf{x}+\epsilon\mathbf{1}_S) - F(\mathbf{x}+\epsilon\mathbf{1}_{S_l}) \geq (1-O(\epsilon))\mathbf{E}[f_{R(\mathbf{x}+\epsilon\mathbf{1}_{S_l})}(O_s)]$.

Once again we design a set of guess values (W) in Claim 5.6 for which the above properties are satisfied.

Then the algorithm at the end of each time step does certain bookkeeping on the vectors $\mathbf{x}, \mathbf{y}_i$ and $\mathbf{z}$. Finally it invokes a rounding algorithm which we describe in Section 5.2.3.

CLAIM 5.5. *Algorithm 7 has running time* $O\left(n^2 \cdot \left(\frac{\log(n)}{\epsilon}\right)^{\frac{1}{\epsilon^8}}\right)$.

Proof. It is straightforward to see that the running time of each instantiation of continuous greedy is $O(n^2 \log^2 n/\epsilon^9)$. The total running time is a product of this with number of possible choices for w.

$$\begin{aligned}\text{Running time} \leq& \left(\frac{\log(n)}{\epsilon}\right)^{1/\epsilon^7} \cdot \frac{n^2 \log^2 n}{\epsilon^9} \\ =& O\left(n^2 \cdot \left(\frac{\log(n)}{\epsilon}\right)^{1/\epsilon^8}\right).\end{aligned}$$

CLAIM 5.6. *Algorithm 7 has approximation ratio* $1 - 1/e - O(\epsilon)$.

Proof. The proof is by designing a sequence of w for which the algorithm Greedy-With-Guessing gives a $1 - 1/e - \epsilon$ approximation.

For each step define w_t^i as the maximum number in the guess-set smaller than $\mathbf{E}[f_{R(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_{i-1}})}(o_i)]$. Then by the choice of e_i we have the following two properties.
• $\mathbf{E}[f_{R(\mathbf{x}(t),S_{i-1})}(e_i)] + \frac{\epsilon}{n}f(O) \geq w_t^i \geq (1-\epsilon)\mathbf{E}[f_{R(\mathbf{x}(t),S_{i-1})}(o_i)]$. This follows from the fact that o_i is a candidate for e_i.
• $c(e_i) \leq c(o_i)$.

Define W_t as the largest value in Guess-Set smaller than $\epsilon\mathbf{E}[f_{R(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_l})}(O_s)]$. Then from Claim 5.3 of the section for handling small items we have that
• $F(\mathbf{x}(t) + \epsilon\mathbf{1}_{S_l\cup S_s}) - F(\mathbf{x}(t) + \epsilon\mathbf{1}_{S_l}) \geq \epsilon(1-12\epsilon)\mathbf{E}[f_{R(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_l})}(O_s)]$.
• $c(S_s) \leq c(O_s)$.

For the set of w_t^i, W_t thus defined we will see that a simple rounding in Section 5.2.3 returns a feasible solution. Additionally we can bound the increase at each time step as follows.

$$\begin{aligned}
& F(\mathbf{x}(t+\epsilon)) - F(\mathbf{x}(t)) \\
=& \sum_{i=1}^{1/\epsilon^6} \left(F(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_{i-1}\cup\{e_i\}}) - F(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_{i-1}})\right) \\
& + F(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_l\cup S_s}) - F(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_l}) \\
\geq& \epsilon\sum_{i=1}^{1/\epsilon^6} \mathbf{E}[f_{R(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_{i-1}})}(e_i)] \\
& + F(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_l\cup S_s}) - F(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_l}) \\
\geq& \epsilon(1-\epsilon)\left(\sum_{i=1}^{1/\epsilon^6} \mathbf{E}[f_{R(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_{i-1}})}(o_i)] - \frac{\epsilon}{n}f(O)\right) \\
& + \epsilon(1-12\epsilon)\mathbf{E}[f_{R(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_l})}(O_s)] - \epsilon^2 f(O) \\
=& \epsilon(1-\epsilon)\sum_{i=1}^{1/\epsilon^6}(\mathbf{E}[f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_{i-1}})\cup\{o_i\}) \\
& \quad - f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_{i-1}}))]) - \epsilon^2 f(O) \\
& + \epsilon(1-12\epsilon)\mathbf{E}[f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_l})\cup O_s) \\
& \quad - f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_{S_l}))] - \epsilon^2 f(O) \\
\geq& \epsilon(1-\epsilon)\sum_{i=1}^{1/\epsilon^6}(\mathbf{E}[f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S)\cup O_i) \\
& \quad - f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S)\cup O_{i-1})]) \\
& + \epsilon(1-12\epsilon)\mathbf{E}[f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S)\cup O_s) \\
& \quad - f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S))] - 2\epsilon^2 f(O) \\
=& \epsilon(1-\epsilon)(\mathbf{E}[f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S)\cup O_l) \\
& \quad - f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S))]) \\
& + \epsilon(1-12\epsilon)\mathbf{E}[f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S)\cup O_s) \\
& \quad - f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S))] - 2\epsilon^2 f(O) \\
\geq& \epsilon(1-\epsilon)(\mathbf{E}[f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S)\cup O) \\
& \quad - f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S)\cup O_s)]) \\
& + \epsilon(1-12\epsilon)\mathbf{E}[f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S)\cup O_s) \\
& \quad - f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S))] - 2\epsilon^2 f(O) \\
\geq& \epsilon(1-12\epsilon)\mathbf{E}[f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S)\cup O) \\
& \quad - f(R(\mathbf{x}(t)+\epsilon\mathbf{1}_S))] - 2\epsilon^2 f(O) \\
\geq& \epsilon(1-14\epsilon)(f(O) - F(\mathbf{x}(t+\epsilon))).
\end{aligned}$$

Now we proceed similarly to the analysis of continuous greedy for a matroid constraint. Rephrasing we get $f(O) - F(\mathbf{x}(t+\epsilon)) \leq \frac{f(O)-F(\mathbf{x}(t))}{1+\epsilon(1-14\epsilon)}$. Then by induction we can prove

$$f(O) - F(\mathbf{x}(t)) \leq \left(\frac{1}{(1+\epsilon(1-14\epsilon))^{\frac{t}{\epsilon}}}\right) f(O).$$

Now substituting $t = 1$ and rephrasing once again

$$F(\mathbf{x}(1)) \geq \left(1 - \frac{1}{(1+\epsilon(1-14\epsilon))^{\frac{1}{\epsilon}}}\right) f(O)$$
$$\geq (1 - 1/e - O(\epsilon))f(O).$$

5.2.3 Rounding In this subsection we show how to convert a fractional solution as produced by Algorithm 7 to an integral solution. We restrict our attention to the fractional solution $\{\mathbf{x}, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{1/\epsilon^6}, \mathbf{z}\}$ corresponding to the values w, W designed in Lemma 5.6 for which we have the following properties.

- $F(\mathbf{x}) \geq (1 - 1/e - O(\epsilon))f(O)$.
- $\mathbf{x} = \sum_{i=1}^{1/\epsilon} \epsilon \mathbf{1}_{S_{\epsilon i}}$ where $S_{\epsilon i}$ is the set chosen at time ϵi.
- The set chosen at time t, $S_t = S_{l,t} \cup S_{s,t}$ can be represented as a union of large items ($S_{l,t} = \{s_{1,t}, s_{2,t}, \ldots, s_{1/\epsilon^6,t}\}$) and small items ($S_{s,t}$). Let $O = O_l \cup O_s$ with $O_l = \{o_1, o_2, \ldots, o_{1/\epsilon^6}\}$
- For each time t and each $s_{i,t} \in S_{l,t}$ we have that $s_{i,t} \in E_i$. Additionally $S_{s,t} \subseteq E_s$.
- For each time t and each $s_{i,t} \in S_{l,t}$ we have that $c(s_{i,t}) \leq c(o_i)$.
- For each time t we have that $c(S_{s,t}) \leq c(O_s)$.

Now from the above properties one can see that for any given i since $c(s_{i,t}) \leq c(o_i)$ we can choose any of the elements $\cup_t \{s_{i,t}\}$ and it will be a good substitute for o_i. Now applying Lemma 2.4 we shall not lose anything in the approximation ratio. The rounding is formally described in Algorithm 9.

Algorithm 9 Rounding-Algorithm

Input: Fractional solution $\mathbf{x} = \mathbf{z} + \sum_{i=1}^{\frac{1}{\epsilon^6}} \mathbf{y}_i = \sum_t \epsilon \mathbf{1}_{S_t}$.
Output: A set $S \subseteq E$ satisfying $c(S) \leq 1$ and $\mathbf{E}[f(S)] \geq (1-\epsilon)F(\mathbf{x})$

1: $S_l, S_s \leftarrow \emptyset$
2: Define $\mathbf{z}' \in [0,1]^{E_s}$ as $\mathbf{z}'(e) = \mathbf{z}(e)$ if $c(e) < \epsilon^3 \max_i c(S_{s,t})$ and $\mathbf{z}'(e) = 0$ otherwise.
3: **for** $i = 1, 2, \ldots, 1/\epsilon^6$ **do**
4: Include one of $e \in E_i$ to S_l with e being included with probability $\mathbf{y}_i(e)$.
5: **end for**
6: For each $e \in E_s$, include in S_s independently with probability $(1-\epsilon)\mathbf{z}'(e)$.
7: $S \leftarrow S_l \cup S_s$.
8: **return** S.

Claim 5.7. *Algorithm 9 rounds the fractional solution to a set S such that $\mathbf{E}[f(S)] \geq (1-\epsilon)F(\mathbf{x})$.*

Proof. First let us relate value of the fractional solution $\mathbf{x} = \mathbf{z} + \sum_t \mathbf{y}_t$ to that of $\mathbf{x}' = \mathbf{z}' + \sum_t \mathbf{y}$.

$$\begin{aligned}
F(\mathbf{x}') =& F\left(\mathbf{z}' + \sum_t \mathbf{y}_t\right) \\
\geq& F\left(\mathbf{z} + \sum_t \mathbf{y}_t\right) - \sum_{c(e) \geq \epsilon^3 \max_i c(S_{s,t})} \mathbf{z}(e) f(e) \\
\geq& F(\mathbf{x}) - \max_{e \in E_s} f(e) \sum_{c(e) \geq \epsilon^3 \max_i c(S_{s,t})} \mathbf{z}(e) \\
\geq& F(\mathbf{x}) - \left(\epsilon^6 f(O)\right) \frac{1}{\epsilon^3} \\
\geq& F(\mathbf{x}) - \epsilon^3 f(O) \\
\geq& (1-\epsilon)F(\mathbf{x}).
\end{aligned}$$

Next we note that we get S by selecting exactly one random element from each E_i, and sampling independently from E_s. Hence applying Lemma 2.4 we get

$$\mathbf{E}[f(S)] \geq F((1-\epsilon)\mathbf{x}') \geq (1-\epsilon)F(\mathbf{x}') \geq (1-O(\epsilon))F(\mathbf{x}).$$

Claim 5.8. *Algorithm 9 rounds the fractional solution to a set S such that with high probability $c(S) \leq 1$. i.e S is a feasible solution to the knapsack constraint.*

Proof. Note that by the condition that for each i, $c(s_{i,t}) \leq c(o_i)$ the large items after the rounding will have cost less than the large items of the optimal solution. Hence it is enough to prove that with high probability $c(S_s) \leq c(O_s)$. First note that

$$\begin{aligned}
\mathbf{E}[c(R((1-\epsilon)\mathbf{z}'))] &= (1-\epsilon)\mathbf{E}[c(R(\mathbf{z}'))] \\
&\leq (1-\epsilon)\mathbf{E}[c(R(\mathbf{z}))] \\
&\leq (1-\epsilon) \max_t c(S_{s,t}) \leq (1-\epsilon)c(O_s).
\end{aligned}$$

Additionally we have that for each element e with $\mathbf{z}'(e) > 0$ we have that $c(e) \leq \epsilon^3 c(O_s)$. Hence we bound the probability that $c(S_s) > c(O_s)$ by a simple Chernoff bound (Lemma 2.2) to show this happens with very low probability. We set up the Chernoff bound with $X_e = c(e)/(\epsilon^3 c(O_s))$ if $e \in S_s$ and $X_e = 0$ otherwise. Then $\mathbf{E}[X] = \sum_e \mathbf{E}[X_e] = \mathbf{E}[c(R((1-\epsilon)\mathbf{z}'))]/(\epsilon^3 c(O_s)) \leq (1-\epsilon)/\epsilon^3$. Therefore, we obtain

$$\Pr[c(S_s) > c(O_s)] = \Pr[X \geq \epsilon^{-3}]$$
$$\leq e^{-\frac{\epsilon^2}{3}\frac{1}{\epsilon^3}} \leq e^{-\frac{1}{3\epsilon}} \leq 3\epsilon.$$

6 p-system and ℓ knapsack constraints

Here we consider a more general type of constraint: a p-system combined with ℓ knapsack constraints. (We

refer the reader to Section 2 for the definition of a p-system.) This is the most general type of constraint that we handle in this paper. As special cases, this contains the intersection of p matroids and ℓ knapsacks, as well as p-set packing combined with ℓ knapsack constraints. We assume without loss of generality that each knapsack constraint has unit capacity.

Algorithm 10 p-system and ℓ knapsack constraints

Input: $f : 2^E \to \mathbb{R}_+$, a membership oracle for p-system $\mathcal{I} \subset 2^E$, and ℓ knapsack-cost functions $c_i : E \to [0,1]$.

Output: A set $S \subseteq E$ satisfying $S \in \mathcal{I}$ and $c_i(S) \leq 1 \forall i$.

$M \leftarrow \max_{j \in E} f(j)$
repeat the following for $\rho \in \{\frac{M}{p+\ell}, (1+\epsilon)\frac{M}{p+\ell}, (1+\epsilon)^2\frac{M}{p+\ell}, \ldots, \frac{2nM}{p+\ell}\}$ *(density threshold)*
  $\tau \leftarrow M_\rho \leftarrow \max\{f(j) : \frac{f(j)}{\sum_{i=1}^{\ell} c_{ij}} \geq \rho\}$ *(value threshold)*
  $S \leftarrow \emptyset$
  while $\tau \geq \frac{\epsilon}{n} M_\rho$ and $c_i(S) \leq 1 \forall i$ **do**
    for $\forall j \in E$ **do**
      if $S + j \in \mathcal{I}$ and $f_S(j) \geq \tau$ and $\frac{f_S(j)}{\sum_{i=1}^{\ell} c_{ij}} \geq \rho$ **then**
        $S \leftarrow S + j$
        if $\exists i; c_i(S) > 1$ **then**
          $S_\rho \leftarrow S$, $T_\rho \leftarrow S \setminus \{j\}$, $T'_\rho \leftarrow \{j\}$
          restart with the next value of ρ
        end if
      end if
    end for
    $\tau \leftarrow \frac{1}{1+\epsilon}\tau$
  end while
  $T_\rho \leftarrow S_\rho \leftarrow S$, $T'_\rho \leftarrow \emptyset$
  restart with the next value of ρ
return the set of maximum value among T_ρ and T'_ρ for all enumerations of ρ.

Algorithm overview: We define the density of an element j with respect to a set S as $\frac{f_S(j)}{\sum_{i=1}^{\ell} c_{ij}}$. Our algorithm combines two ideas:

- a fixed density threshold ρ, which is somewhat below the value/size ratio of the optimal solution; this is guessed (i.e. enumerated) by the algorithm,

- a decreasing value threshold τ, which mimics the greedy algorithm for p-systems but leads to a faster running time.

In each stage, the algorithm picks all items that are above the density threshold and also above the value threshold. The value threshold decreases slightly after each stage. This is effectively a greedy algorithm with respect to marginal values, while discarding all elements of density below some threshold. The greedy rule guarantees an approximation ratio for a p-system constraint, while the density threshold guarantees that we do not exceed the knapsack constraints without collecting enough value.

For a formal description, see Algorithm 10. We call the execution of the algorithm for a given value of ρ a "run of the algorithm". We call the inner loop for a given value of τ a "stage of the algorithm".

THEOREM 6.1. *Algorithm 10 runs in time $O(\frac{n}{\epsilon^2}\log^2\frac{n}{\epsilon})$ and provides a $(1+\epsilon)(p+2\ell+1)$-approximation for the problem of maximizing a monotone submodular function subject to a p-system and ℓ knapsack constraints.*

We note that we consider p, ℓ to be constant; i.e., the O notation hides a constant depending on p, ℓ. This constant is actually very mild - linear in ℓ, and independent of p (this is due to our model which assumes a membership oracle for the p-system). The following claim gives a more accurate statement of running time.

CLAIM 6.1. *The running time of Algorithm 10 is $O(\ell\frac{n}{\epsilon^2}\log^2\frac{n}{\epsilon})$.*

Proof. The enumeration loop consists of $\log_{1+\epsilon}(2n)$ values of ρ. The decreasing-threshold loop consists of $\log_{1+\epsilon}\frac{n}{\epsilon}$ values of τ. The inner loop checks each element of E, querying its marginal value and performing an $O(\ell)$ time computation. Therefore, the total running time is

$$O\left(\ell n \log_{1+\epsilon}(2n)\log_{1+\epsilon}\frac{n}{\epsilon}\right) = O\left(\ell\frac{n}{\epsilon^2}\log^2\frac{n}{\epsilon}\right).$$

For the analysis of the approximation ratio, let us introduce some notation: We fix an optimal solution O. Given a density threshold ρ and the resulting solution S_ρ, we denote

$$O_{<\rho} = \{j \in O : \frac{f_{S_\rho}(j)}{\sum_{i=1}^{\ell} c_{ij}} < \rho\}$$

and $O_{\geq\rho} = O \setminus O_{<\rho}$.

CLAIM 6.2. *Given ρ and the respective algorithmic solution S_ρ, we have*

$$f_{S_\rho}(O_{<\rho}) \leq \ell\rho.$$

Proof. By the definition of $O_{<\rho}$, for each $j \in O_{<\rho}$ we have $f_{S_\rho}(j) < \rho \sum_{i=1}^{\ell} c_{ij}$. By submodularity, $f_{S_\rho}(O_{<\rho}) \leq \sum_{j \in O_{<\rho}} f_{S_\rho}(j) \leq \rho \sum_{i=1}^{\ell} \sum_{j \in O_{<\rho}} c_{ij} \leq \rho\ell$ by the feasibility of O and $O_{<\rho}$ being a subset of O.

Claim 6.3. *If the algorithm terminates without exceeding any knapsack constraint, we have*

$$f_{S_\rho}(O_{\geq\rho}) \leq ((1+\epsilon)p + \epsilon) \cdot f(S_\rho).$$

The proof of this claim is essentially the analysis of the greedy algorithm for a p-system, as in [2]. We supply the proof for completeness.

Proof. Consider the elements of S_ρ in the order they are added by the algorithm: denote them by $e_1, e_2, \ldots, e_r$. For $i \in \{1, \ldots, r\}$, we define a set $A_i \subseteq O_{\geq\rho} \setminus S_\rho$, consisting of the elements that could have been added instead of e_i:

$$A_i = \{j \in O_{\geq\rho} \setminus S_\rho : \{e_1, \ldots, e_{i-1}, j\} \in \mathcal{I}\}.$$

We also define A_{r+1} in the same way; these are the elements in $O_{\geq\rho} \setminus S_\rho$ that could still be added to S_ρ at the end of the run. Since $O_{\geq\rho} \setminus S_\rho \in \mathcal{I}$, we have $A_1 = O_{\geq\rho} \setminus S_\rho$. By the down-monotonicity of $\mathcal{I}$, it follows that $O_{\geq\rho} \setminus S_\rho = A_1 \supseteq A_2 \supseteq A_3 \supseteq \ldots A_{r+1}$.

Next, we use the defining property of p-systems. Consider the set $Q_i = \{e_1, \ldots, e_i\} \cup (A_1 \setminus A_{i+1})$. We have $A_1 \setminus A_{i+1} \in \mathcal{I}$, hence Q_i has a base of size at least $|A_1 \setminus A_{i+1}|$. On the other hand, $\{e_1, \ldots, e_i\}$ is also a base of Q_i, since no element of $A_1 \setminus A_{i+1}$ can be added to $\{e_1, \ldots, e_i\}$ (otherwise by definition it would be in A_{i+1}). Since $\mathcal{I}$ is a p-system, the cardinality of the two bases cannot differ by a factor larger than p; equivalently, $|A_1 \setminus A_{i+1}| \leq pi$.

Now consider the stages of the algorithm defined by decreasing τ. For $1 \leq i \leq r$, define τ_i to be the value of the threshold τ when e_i was included in S_ρ. In particular, the marginal value of e_i at that point is $f_{\{e_1,\ldots,e_{i-1}\}}(e_i) \geq \tau_i$. For each $j \in A_i$, if j were considered at a stage earlier than e_i, j would have been added to S_ρ because its density would have been above ρ (by submodularity) and adding it to $\{e_1, \ldots, e_{i-1}\}$ would not violate feasibility in $\mathcal{I}$. (Also, we assumed that no knapsack constraint overflows in this run of the algorithm). However, $j \notin S_\rho$, and hence j could not have been considered before e_i. This implies that $f_{\{e_1,\ldots,e_{i-1}\}}(j) \leq (1+\epsilon)\tau_i$, for each $j \in A_i$.

By a similar argument, for each $j \in A_{r+1}$, we have $f_{S_\rho}(j) < \frac{\epsilon}{n} M_\rho$ (the lowest possible value of the threshold τ; see the algorithm for definition of M_ρ), otherwise j would still be added to $S_\rho = \{e_1, \ldots, e_r\}$. In fact, it can be seen that the first element chosen by the algorithm is of value exactly M_ρ, and $\tau_1 = M_\rho$. Therefore, $f_{S_\rho}(j) \leq \frac{\epsilon}{n}\tau_1$ for all $j \in A_{r+1}$.

It remains to add up the values of all the elements in $A_1 = O_{\geq\rho} \setminus S_\rho$, marginally with respect to S_ρ. By submodularity, we have

$$\begin{aligned} f_{S_\rho}(A_1) &\leq \sum_{j \in A_1} f_{S_\rho}(j) \\ &= \sum_{i=1}^{r} \sum_{j \in A_i \setminus A_{i+1}} f_{S_\rho}(j) + \sum_{j \in A_{r+1}} f_{S_\rho}(j). \end{aligned}$$

As we proved, $f_{S_\rho}(j) \leq f_{\{e_1,\ldots,e_{i-1}\}}(j) \leq (1+\epsilon)\tau_i$ for $j \in A_i \setminus A_{i+1}$, and $f_{S_\rho}(j) \leq \frac{\epsilon}{n}\tau_1$ for $j \in A_{r+1}$. So we obtain

$$f_{S_\rho}(A_1) \leq (1+\epsilon) \sum_{i=1}^{r} |A_i \setminus A_{i+1}|\tau_i + |A_{r+1}|\frac{\epsilon}{n}\tau_1.$$

We proved above that $|A_1 \setminus A_{i+1}| \leq pi$. Also, observe that due to the operation of the algorithm, $\tau_1 \geq \tau_2 \geq \tau_3 \geq \ldots$. Under these conditions, the right-hand side is maximized when $|A_i \setminus A_{i+1}| = p$ for each $1 \leq i \leq r$. Thus we obtain

$$\begin{aligned} f_{S_\rho}(A_1) &\leq (1+\epsilon) \sum_{i=1}^{r} p\tau_i + |A_{r+1}|\frac{\epsilon}{n}\tau_1 \\ &\leq (1+\epsilon) \sum_{i=1}^{r} p\tau_i + \epsilon\tau_1. \end{aligned}$$

Recall that τ_i was the value of the threshold τ when e_i was included; this implies that $f_{e_1,\ldots,e_{i-1}}(e_i) \geq \tau_i$, i.e. $f(S_\rho) \geq \sum_{i=1}^{r} \tau_i$. Therefore,

$$f_{S_\rho}(O_{\geq\rho}) = f_{S_\rho}(A_1) \leq (1+\epsilon)p \cdot f(S_\rho) + \epsilon \cdot f(S_\rho).$$

Let us proceed to the proof of Theorem 6.1.

Proof. Consider an optimal solution O and set $\rho^* = \frac{2}{p+2\ell+1} f(O)$. Since $M \leq f(O) \leq nM$ (by submodularity), we have $\rho^* \in [\frac{M}{p+\ell}, \frac{2nM}{p+\ell}]$. Thus there is a run of the algorithm with density threshold ρ such that $\rho^* \in [\rho, (1+\epsilon)\rho]$. In the following we consider this run of the algorithm. We consider two cases.

(1) If the algorithm terminates by exceeding some knapsack capacity, then we obtain a set S_ρ such that $c_i(S_\rho) > 1$ for some $i \in [\ell]$. I.e., certainly $\sum_{i=1}^{\ell} c_i(S_\rho) > 1$. Since every element that we include satisfies $f_S(j) \geq \rho \sum_{i=1}^{\ell} c_{ij}$, with respect to the current solution S, we obtain

$$f(S_\rho) \geq \rho \sum_{i=1}^{\ell} \sum_{j \in S_\rho} c_{ij} > \rho.$$

This solution is infeasible; however, we keep either $T_\rho = S_\rho \setminus \{j\}$ or $T'_\rho = \{j\}$ where j is the last element

included in S_ρ. Each of these solutions is feasible and by submodularity, one of them has value at least $\frac{1}{2}\rho$. By our choice of $\rho \geq \frac{1}{1+\epsilon}\rho^* = \frac{2}{(1+\epsilon)(p+2\ell+1)}f(O)$, we obtain

$$\max\{f(T_\rho), f(T'_\rho)\} \geq \frac{1}{2}\rho \geq \frac{1}{(1+\epsilon)(p+2\ell+1)}f(O).$$

(2) If the algorithm terminates without exceeding any knapsack capacity, then by the above claims,

$$f_{S_\rho}(O_{<\rho}) \leq \ell\rho \leq \ell\rho* = \frac{2\ell}{p+2\ell+1}f(O)$$

and

$$f_{S_\rho}(O_{\geq\rho}) \leq ((1+\epsilon)p+\epsilon)\cdot f(S_\rho).$$

By submodularity,

$$\begin{aligned} f_{S_\rho}(O) &\leq f_{S_\rho}(O_{<\rho}) + f_{S_\rho}(O_{\geq\rho}) \\ &\leq \frac{2\ell}{p+2\ell+1}f(O) + ((1+\epsilon)p+\epsilon)\cdot f(S_\rho). \end{aligned}$$

Since $f_{S_\rho}(O) = f(S_\rho \cup O) - f(S_\rho) \geq f(O) - f(S_\rho)$, this means

$$f(O) \leq \frac{2\ell}{p+2\ell+1}f(O) + (1+\epsilon)(p+1)\cdot f(S_\rho)$$

and from here

$$(1+\epsilon)f(S_\rho) \geq \frac{1}{p+2\ell+1}f(O).$$

Again, $f(S_\rho) \geq \frac{1}{(1+\epsilon)(p+2\ell+1)}f(O)$, which proves the theorem.

7 A lower bound for p-systems

THEOREM 7.1. *For any $\epsilon > 0$, a $1/(p+\epsilon)$-approximation for the problem $\max\{|S| : S \in \mathcal{I}\}$, where $\mathcal{I}$ is a p-system, requires an exponential number of queries to $\mathcal{I}$.*

Proof. The proof is by constructing two p-systems $P_1, P_2 \subset 2^E$ randomly, with $p\cdot\max\{|S| : S \in P_1\} = \max\{|S| : S \in P_2\}$, such that it requires an exponential number of queries to distinguish between them with constant probability. We define the two set systems as follows.

- Define $t = n/(4p)$ (where $n = |E|$) and let $S \subseteq E$ be in P_1 iff $|S| \leq t$.
- Let T be a set of size pt chosen uniformly at random from $\binom{E}{pt}$. Then let S be in P_2 iff either $|S| \leq t$ or $S \subseteq T$.

It is easy to verify that P_1 and P_2 are both p-systems: for P_1 it is trivial, and for P_2 all maximal bases have size between t and pt. Also, $\max\{|S| : S \in P_1\} = t$ and $\max\{|S| : S \in P_2\} = pt$.

Now consider any query S. S can distinguish between P_1 and P_2 iff $t < |S| \leq pt$ and $S \subseteq T$. Fix any S with $t < |S| \leq pt$. Then $S \subseteq T$ with probability $\frac{\binom{pt}{|S|}}{\binom{n}{|S|}}$ over the random choice of T. Furthermore, as long as $S \not\subseteq T$, we do not learn anything about whether the set system is P_2 or the identity of T. Hence we need to make $\Omega(\binom{n}{|S|}/\binom{pt}{|S|})$ queries to distinguish P_1 from P_2 with constant probability. From here,

$$\begin{aligned} \#\text{ Queries required} &= \Omega\left(\frac{\binom{n}{|S|}}{\binom{pt}{|S|}}\right) \geq \Omega\left(\frac{(n-|S|)^{|S|}}{(pt)^{|S|}}\right) \\ &\geq \Omega\left(\left(\frac{n-pt}{pt}\right)^t\right) = \Omega\left(3^{n/(4p)}\right). \end{aligned}$$

References

[1] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *FOCS*, 2012. 2

[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. 2, 3, 4, 5, 16

[3] Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *STOC*, pages 783–792, 2011. 2, 3, 5, 6

[4] Shahar Dobzinski and Michael Schapira. An improved approximation algorithm for combinatorial auctions with submodular bidders. In *SODA*, pages 1064–1073, 2006. 1

[5] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. 4

[6] Uriel Feige. On maximizing welfare when utility functions are subadditive. In *STOC*, pages 41–50, 2006. 1

[7] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. In *FOCS*, pages 461–471, 2007. 2

[8] Uriel Feige and Jan Vondrák. Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. In *FOCS*, pages 667–676, 2006. 1

[9] Moran Feldman, Joseph Naor, and Roy Schwartz. Nonmonotone submodular maximization via a structural continuous greedy algorithm - (extended abstract). In *ICALP*, pages 342–353, 2011. 2

[10] Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *FOCS*, pages 570–579, 2011. 2

[11] Yuval Filmus and Justin Ward. A tight combinatorial algorithm for submodular maximization subject to a matroid constraint. *CoRR*, abs/1204.4526, 2012. 2, 3

[12] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions - II. *Math. Prog. Study*, 8:73–87, 1978. 1, 2, 3

[13] Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *SODA*, pages 1098–1116, 2011. 2

[14] Carlos Guestrin, Andreas Krause, and Ajit Paul Singh. Near-optimal sensor placements in gaussian processes. In *ICML*, pages 265–272, 2005. 1

[15] Anupam Gupta, Viswanath Nagarajan, and R. Ravi. Robust and maxmin optimization under matroid and knapsack uncertainty sets. *CoRR*, abs/1012.4962, 2010. 2

[16] Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *WINE*, pages 246–257, 2010. 2

[17] David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003. 1

[18] David Kempe, Jon M. Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *ICALP*, pages 1127–1138, 2005. 1

[19] Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999. 8

[20] Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM TIST*, 2(4):32, 2011. 1

[21] Andreas Krause, Carlos Guestrin, Anupam Gupta, and Jon M. Kleinberg. Near-optimal sensor placements: maximizing information while minimizing communication cost. In *IPSN*, pages 2–10, 2006. 1

[22] Andreas Krause, Carlos Guestrin, Anupam Gupta, and Jon M. Kleinberg. Robust sensor placements at informative and communication-efficient locations. *TOSN*, 7(4):31, 2011. 1

[23] 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. 1, 2

[24] Ariel Kulik, Hadas Shachnai, and Tami Tamir. Maximizing submodular set functions subject to multiple linear constraints. In *SODA*, pages 545–554, 2009. 2, 3

[25] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *STOC*, pages 323–332, 2009. 2

[26] Jon Lee, Maxim Sviridenko, and Jan Vondrák. Matroid matching: the power of local search. In *STOC*, pages 369–378, 2010. 2, 4

[27] Benny Lehmann, Daniel J. Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006. 1

[28] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. VanBriesen, and Natalie S. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007. 1, 2

[29] G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set functions. *Math. Oper. Research*, 3(3):177–188, 1978. 1

[30] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Prog.*, 14:265–294, 1978. 1, 3

[31] Alexander Schrijver. Combinatorial Optimization, *Volume B*. Springer, 2003. 4

[32] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004. 2, 3, 8

[33] Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*, pages 67–74, 2008. 1, 2, 3, 5

[34] Jan Vondrák. Symmetry and approximability of submodular maximization problems. In *FOCS*, pages 651–670, 2009. 2