“Bring Your Own Greedy”+Max: Near-Optimal 1/2-Approximations for Submodular Knapsack

Grigory Yaroslavtsev
Indiana University
The Alan Turing Institute

Samson Zhou
Indiana University
Carnegie Mellon University

Dmitrii Avdiukhin
Indiana University

Abstract

The problem of selecting a small-size representative summary of a large dataset is a cornerstone of machine learning, optimization and data science. Motivated by applications to recommendation systems and other scenarios with query-limited access to vast amounts of data, we propose a new rigorous algorithmic framework for a standard formulation of this problem as a submodular maximization subject to a linear (knapsack) constraint. Our framework is based on augmenting all partial Greedy solutions with the best additional item. It can be instantiated with negligible overhead in any model of computation, which allows the classic Greedy algorithm and its variants to be implemented. We give such instantiations in the offline (Greedy+Max), multi-pass streaming (Sieve+Max) and distributed (Distributed Sieve+Max) settings. Our algorithms give $(1/2 - \epsilon)$-approximation with most other key parameters of interest being near-optimal. Our analysis is based on a new set of first-order linear differential inequalities and their robust approximate versions. Experiments on typical datasets (movie recommendations, influence maximization) confirm scalability and high quality of solutions obtained via our framework. Instance-specific approximations are typically in the 0.6-0.7 range and frequently beat even the $(1 - 1/e) \approx 0.63$ worst-case barrier for polynomial-time algorithms.

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

1 Introduction

A fundamental problem in many large-scale machine learning, data science and optimization tasks is finding a small representative subset of a big dataset. This problem arises from applications in recommendation systems Leskovec et al. (2007); El-Arini and Guestrin (2011); Bogunovic et al. (2017); Mitrović et al. (2017); Yu et al. (2018); Avdiukhin et al. (2019), exemplar-based clustering Gomes and Krause (2010), facility location Lindgren et al. (2016), image processing Iyer and Bilmes (2019), viral marketing Hartline et al. (2008), principal component analysis Khanna et al. (2015), and document summarization Lin and Bilmes (2011); Wei et al. (2013); Sipos et al. (2012) and can often be formulated as constrained monotone submodular optimization under various constraints such as cardinality Badanidiyuru et al. (2014); Bateni et al. (2018); Kazemi et al. (2019), knapsack Huang et al. (2017), matchings Chakrabarti and Kale (2014), and matroids Călinescu et al. (2011); Anari et al. (2019) due to restrictions demanded by space, budget, diversity, fairness or privacy. As a result, constrained submodular optimization has been recently and extensively studied in various computational models, including centralized Nemhauser et al. (1978), distributed Mirzasoleiman et al. (2013); Kumar et al. (2015); da Ponte Barbosa et al. (2015); Mirrokni and Zadimoghaddam (2015); Mirzasoleiman et al. (2016); da Ponte Barbosa et al. (2016); Liu and Vondrák (2019), streaming Badanidiyuru et al. (2014); Buchbinder et al. (2015); Norouzi-Fard et al. (2018); Agrawal et al. (2019); Kazemi et al. (2019), and adaptive Golovin and Krause (2011); Balkanski and Singer (2018); Balkanski et al. (2019); Fahrbach et al. (2019); Ene and Nguyen (2019b); Chekuri and Quanrud (2019) among others.

In this paper we focus on monotone submodular maximization *under a knapsack constraint*, which captures the scenario when the representative subset should have a small cost or size. While a number of algorithmic techniques exist for this problem, there are few that robustly scale to large data and can be easily implemented in various computing frameworks. This is in contrast with a simpler *cardinality-constrained* version in which only the number of elements is restricted. In this setting the celebrated Greedy algorithm of Nemhauser et al. (1978) enjoys both an optimal approximation ratio and a simplicity that allows easy adaptation in various environments. For knapsack constraints, such a simple and universal algorithm is unlikely. In particular, Greedy does not give any approximation guarantee.

We develop a framework that augments solutions constructed by Greedy and its variations and gives almost $1/2$-approximations[1] in various computational models. For example, in the multi-pass streaming setting we achieve optimal space and almost optimal number of queries and running time. We believe that our framework is robust to the choice of the computational model as it can be implemented with essentially the same complexity as that of running Greedy and its variants.

1.1 Preliminaries and our contributions

A set function $f\colon 2^U \to \mathbb{R}$ is *submodular* if for every $S \subseteq T \subseteq U$ and $e \in U$ it holds that $f(e \cup T) - f(T) \leq f(e \cup S) - f(S)$. Moreover, f is *monotone* if for every $S \subseteq T \subseteq U$ it holds that $f(T) \geq f(S)$. Intuitively, elements in the universe contribute non-negative utility, but their resulting gain is diminishing as the size of the set increases. In the monotone submodular maximization problem subject to a knapsack constraint, each item e has cost $c(e)$. Given a parameter $K > 0$, the task is to maximize a non-negative monotone submodular function $f(S)$ under the constraint $c(S) := \sum_{e \in S} c(e) \leq K$. Without loss of generality, we assume that $\min_{e \in S} c(e) \geq 1$, which can be achieved by rescaling the costs and taking all items with cost 0. Then $\tilde{K} = \min(n, K)$ is an upper bound on the number of elements in any feasible solution.

Any algorithm for submodular maximization requires *query access* to f. As query access can be expensive, the number of queries is typically considered one of the performance metrics. Furthermore, in some critical applications of submodular optimization such as recommendation systems, another constraint often arises from the fact that only queries to feasible sets are allowed (e.g. when click-through rates can only be collected for sets of ads which can be displayed to the users). Practical algorithms for submodular optimization hence typically only make such queries, an assumption commonly used in the literature (see e.g. Norouzi-Fard et al. (2018)). For *any* algorithm that only makes queries on feasible sets, it is easy to show that $\Omega(n^2)$ queries are required to go beyond $1/2$-approximation under various assumptions on f (Theorem 2.14). Hence it is natural to ask whether we can get a $1/2$-approximation, while keeping other performance metrics of interest nearly optimal and hence not compromising on practicality. We answer this question positively.

We first state the following simplified result in the most basic *offline model* (i.e. when an algorithm can access any element at any time) to illustrate the main ideas and then improve parameters in our other results. In this model, we are given an integer knapsack capacity $K \in \mathbb{Z}^+$ and a set E of elements $e_1, \ldots, e_n$ from a finite universe U.[2]

Theorem 1.1 (Offline Greedy+Max) *Let $\tilde{K} = \min(n, K)$. There exists an offline algorithm* Greedy+Max *(Algorithm 1) that gives a $1/2$-approximation for the submodular maximization problem under a knapsack constraint with query complexity and running time $\mathcal{O}\left(\tilde{K}n\right)$ (Theorem 2.6).*

In the *single-pass streaming model*, the algorithm is given K and a stream E consisting of elements $e_1, \ldots, e_n \in U$, which arrive sequentially. The objective is to minimize the auxiliary space used by algorithm throughout the execution. In the *multi-pass streaming model*, the algorithm is further allowed to make multiple passes over E. This model is typically used for modeling storage devices with sequential access (e.g. hard drives) while using a small amount of RAM. In this setting minimizing the number of passes becomes another key priority. Note that since $\Omega(\tilde{K})$ is a trivial lower bound on space and $\Omega(n)$ is a trivial lower bound on time and query complexity of any approximation algorithm that queries feasible sets, our next result is almost optimal in most parameters of interest.

Theorem 1.2 (Multi-pass streaming algorithm Sieve+Max) *Let $\tilde{K} = \min(n, K)$. There exists a multi-pass streaming algorithm* Sieve+Max *(Algorithm 2) that uses $\mathcal{O}\left(\tilde{K}\right)$ space and $\mathcal{O}\left(1/\epsilon\right)$ passes over the stream and outputs a $(1/2 - \epsilon)$-approximation to the submodular maximization problem under a knapsack constraint, with query complexity and running time[3] $\mathcal{O}\left(n(1/\epsilon + \log \tilde{K})\right)$ (see Theorem 2.10).*

We also give an algorithm in the massively-parallel

[1] Algorithm gives an α-approximation if it outputs S such that $f(S) \geq \alpha f(\mathsf{OPT})$, where OPT is optimum solution.

[2] W.l.o.g. for all e we have $1 \leq c(e) \leq K$ as one can rescale the capacity and costs and filter out all items with cost more than K (in all our results this means replacing K with the aspect ratio $K/\min_{e \in E} c(e)$).

[3] Note that when $\frac{1}{\epsilon} \ll K$, in terms of running time our streaming algorithm is more efficient than our offline algorithm. Hence, in the offline setting one can use the best of the two algorithms depending on the parameters.

computation (MPC) model Karloff et al. (2010) used to model MapReduce/Spark-like systems. We use the most restrictive version, which only allows linear total memory, running time and communication per round Andoni et al. (2014). In this model, the input set E of size n is arbitrarily distributed across m machines, each with $s = \mathcal{O}(n/m)$ memory so that the overall memory is $\mathcal{O}(n)$. A standard setting of parameters for submodular optimization is $m = \sqrt{n/\tilde{K}}$ and $s = \mathcal{O}(\sqrt{n\tilde{K}})$ (see e.g. Liu and Vondrák (2019); Avdiukhin et al. (2019)). One of the machines is designated as the *central machine* and outputs the solution in the end. The machines communicate to each other in a number of synchronous rounds. In each round, each machine receives an input of size $\mathcal{O}(\sqrt{n\tilde{K}})$, performs a local linear-time computation, and sends an output of size $\mathcal{O}(\sqrt{n\tilde{K}})$ to other machines before the next round begins. The primary objective in this model is minimizing the number of rounds. Our main result in this model is given below.

Theorem 1.3 (MPC algorithm DISTRIBUTED SIEVE+MAX) *Let $\tilde{K} = \min(n, K)$. There exists an MPC algorithm* DISTRIBUTED SIEVE+MAX *(Algorithm 3)that runs in $\mathcal{O}(1/\epsilon)$ rounds on $\sqrt{n/\tilde{K}}$ machines, each with $\mathcal{O}(\sqrt{n\tilde{K}})$ memory. Each machine uses query complexity and runtime $\mathcal{O}(\sqrt{n\tilde{K}})$ per round. The algorithm outputs a $(1/2 - \epsilon)$-approximation to the submodular maximization problem under a knapsack constraint(see Theorem 2.13).*

In particular, our algorithm uses execution time $\mathcal{O}(\sqrt{n\tilde{K}}/\epsilon)$ and total communication, CPU time and number of queries $\mathcal{O}(n/\epsilon)$.

1.2 Relationship to previous work

The classic version of the problem considered in this work sets $c(e) = 1$ for all $e \in U$ and is known as monotone submodular maximization under a cardinality constraint and has been extensively studied. The celebrated result of Nemhauser et al. (1978) gives a $1 - 1/e \approx 0.63$-approximation using GREEDY, which is optimal unless $P \neq NP$, Feige (1998). The problem of maximizing a monotone submodular function under a knapsack constraint was introduced by Wolsey (1982), who gave an algorithm with ≈ 0.35-approximation. Khuller et al. (1999) gave a simple GREEDYORMAX algorithm with $1 - 1/\sqrt{e} \approx 0.39$-approximation as well as a more complicated algorithm PARTIALENUM+GREEDY which requires a partial enumeration over an initial seed of three items and hence runs in $\mathcal{O}\left(\tilde{K}n^4\right)$ time. PARTIALENUM+GREEDY was later analyzed by Sviridenko (2004) who showed a $(1 - 1/e) \approx 0.63$-approximation, matching the hardness of Feige (1998). The subsequent search for more efficient algorithms has motivated a number of further studies. Badanidiyuru and Vondrák (2014) and Ene and Nguyen (2019a) give algorithms with approximation $1 - 1/e - \epsilon$. However while these algorithms are theoretically interesting, they are self-admittedly impractical due to their exponential dependence on large polynomials in $1/\epsilon$.

Compared to the well-studied cardinality-constrained case, streaming literature on monotone submodular optimization under a knapsack constraint is relatively sparse. A summary of results in the streaming setting is given in Figure 1. Prior to our work, the best results in streaming are by Huang et al. (2017); Huang and Kakimura (2019). While the most recent work of Huang and Kakimura (2019) achieves the $(1/2 - \epsilon)$-approximation, its space, runtime and query complexities are far from optimal and depend on large polynomials of $1/\epsilon$, making it impractical for large data. Compared to this result, our Theorem 1.2 gives an improvement on all main parameters of interest, leading to near-optimal results. On the other hand, for the cardinality-constrained case, an optimal single-pass $(1/2 - \epsilon)$-approximation has very recently been achieved by Kazemi et al. (2019). While using different ideas, our multi-pass streaming result matches theirs in terms of approximation, space and improves slightly on the number of queries and runtime (from $\mathcal{O}\left(n \log \tilde{K}/\epsilon\right)$ to $\mathcal{O}\left(n(1/\epsilon + \log \tilde{K})\right)$) only at the cost of using a constant number of passes for constant ϵ.

In the distributed setting, Mirzasoleiman et al. (2013) give an elegant two round protocol for monotone submodular maximization subject to a knapsack constraint that achieves a subconstant guarantee. Kumar et al. (2015) later give algorithms for both cardinality and matroid constraints that achieve a constant factor approximation, but the number of rounds is $\Theta(\log \Delta)$, where Δ is the maximum increase in the objective due to a single element, which is infeasible for large datasets since Δ even be significantly larger than the size of the entire dataset. da Ponte Barbosa et al. (2015, 2016) subsequently give a framework for both monotone and non-monotone submodular maximization under cardinality, matroid, and p-system constraints. Specifically, the results of da Ponte Barbosa et al. (2016) achieves almost $1/2$-approximation for these settings using two rounds, a result subsequently matched by Liu and Vondrák Liu and Vondrák (2019) without requiring the duplication of items, as well as a $(1 - 1/e - \epsilon)$ approximation using $\mathcal{O}(1/\epsilon)$ rounds. da Ponte Barbosa et al. (2015) also gives a two-round algorithm for a knapsack constraint that achieves roughly 0.17-approximation in

Reference	Approx.	Passes	Space	Runtime and Queries
Huang et al. (2017)	$1/3 - \epsilon$	1	$\mathcal{O}\left(\frac{1}{\epsilon} K \log K\right)$	$\mathcal{O}\left(\frac{1}{\epsilon} n \log K\right)$
Huang et al. (2017)	$4/11 - \epsilon$	1	$\mathcal{O}\left(\frac{1}{\epsilon} K \log K\right)$	$\mathcal{O}\left(\frac{1}{\epsilon} n \log K\right)$
Huang et al. (2017)	$2/5 - \epsilon$	3	$\mathcal{O}\left(\frac{1}{\epsilon^2} K \log^2 K\right)$	$\mathcal{O}\left(\frac{1}{\epsilon} n \log K\right)$
Huang and Kakimura (2019)	$1/2 - \epsilon$	$\mathcal{O}\left(1/\epsilon\right)$	$\mathcal{O}\left(\frac{1}{\epsilon^7} K \log^2 K\right)$	$\mathcal{O}\left(\frac{1}{\epsilon^8} n \log^2 K\right)$
Sieve+Max (Alg. 2)	$1/2 - \epsilon$	$\mathcal{O}\left(1/\epsilon\right)$	$\mathcal{O}\left(K\right)$	$\mathcal{O}\left(n\left(\frac{1}{\epsilon} + \log K\right)\right)$

Fig. 1: Monotone submodular maximization under a knapsack constraint in the streaming model.

expectation.

For extensions to other constraints, non-monotone objectives and other generalizations see e.g. Chakrabarti and Kale (2014); Chekuri et al. (2015); Chan et al. (2017); Elenberg et al. (2017); Epasto et al. (2017); Mirzasoleiman et al. (2018); Feldman et al. (2018); Chekuri and Quanrud (2019).

1.3 Our techniques

Let $f\left(e \mid S\right) = f(e \cup S) - f(S)$ be the *marginal gain* and $\rho\left(e \mid S\right) = f\left(e \mid S\right)/c(e)$ be the *marginal density* of e with respect to S. Greedy starts with an empty set G and repeatedly adds an item that maximizes $\rho\left(e \mid G\right)$ among the remaining items that fit. While by itself this does not guarantee any approximation, the classic result of Khuller et al. (1999) shows that GreedyOrMax algorithm, which takes the best of the greedy solution and the single item with maximum value, gives a 0.39-approximation but cannot go beyond 0.44-approximation. Our algorithm Greedy+Max (Algorithm 1) instead attempts to augment every partial greedy solution with the item giving the largest marginal gain. For each i, let $\mathcal{G}_i$ be the set of the first i items taken by greedy. We augment this solution with the item s_i which maximizes $f\left(s_i \mid \mathcal{G}_i\right)$ among the remaining items that fit. Greedy+Max then outputs the best solution among such augmentations.

Our main technical contribution lies in the analysis of this algorithm and its variants, which shows a 1/2-approximation (this analysis is tight, see Example 2.1). Let o_1 be the item from OPT with the largest cost. The main idea is to consider the last partial greedy solution such that o_1 still fits. Since o_1 has the largest cost in OPT, we can augment the partial solution with any element from OPT, and all of them have a non-greater marginal density than the next selected item. While Greedy+Max augments partial solutions with the best item, for the sake of analysis it suffices to consider only augmentations with o_1 (note that the item itself is unknown to the algorithm).

To simplify the presentation, in the analysis we rescale f and the costs so that $f(\mathsf{OPT}) = 1$ and $K = 1$. Suppose that at some point, the partial greedy solution has collected elements with total cost $x \in [0,1]$. We use a continuous function $g(x)$ to track the performance of Greedy. We also introduce a function $g_1(x)$ to track the performance of augmentation with o_1 and then show that g and g_1 satisfy a differential inequality $g_1(x) + (1 - c\left(o_1\right))g'(x) \geq 1$ (Lemma 2.5), where g' denotes the right derivative. To give some intuition about the proof, consider the case when there exists a partial greedy solution of cost exactly $1 - c\left(o_1\right)$. If $g_1(1 - c\left(o_1\right)) \geq 1/2$, then the augmenation with o_1 gives a 1/2-approximation. Otherwise, by the differential inequality, $g'(1 - c\left(o_1\right)) \geq 1/2(1-c(o_1))$. Since $g(0) = 0$ and g' is non-increasing, $g(1-c\left(o_1\right)) \geq (1-c\left(o_1\right))g'(1-c\left(o_1\right)) \geq 1/2$. See full analysis for how to handle the cases when there is no partial solution of cost exactly $1 - c\left(o_1\right)$.

Our streaming algorithm Sieve+Max and distributed algorithm Distributed Sieve+Max approximately implement Greedy+Max in their respective settings. Sieve+Max makes $\mathcal{O}\left(1/\epsilon\right)$ passes over the data, and for each pass it selects items with marginal density at least a threshold $\frac{cf(\mathsf{OPT})}{K(1+\epsilon)^i}$ in the i-th pass for some constant $c > 0$. This requires having a constant-factor approximation of $f(\mathsf{OPT})$ which can be computed using a single pass. Distributed Sieve+Max combines the thresholding approach with the sampling technique developed by Liu and Vondrák (2019) for the cardinality constraint. The differential inequality which we develop for Greedy+Max turns out to be robust to various sources of error introduced through thresholding and sampling. As we show, it continues to hold with functions and derivatives replaced with their $(1+\epsilon)$-approximations, which results in $(1/2-\epsilon)$-approximation guarantees for both algorithms.

2 Algorithms and analysis

2.1 Offline algorithm Greedy+Max

We introduce the main ideas by first describing our offline algorithm Greedy+Max which is then adapted to the streaming and distributed settings. As this algorithm is a modification of the standard Greedy algorithm we describe Greedy first. Greedy starts with an empty set G and in each iteration selects an item e with the highest marginal density $\rho\,(e\,|\,G)$ that still fits into the knapsack. We refer to the resulting solution as the *greedy solution* and denote it as G. Greedy+Max is based on augmenting each partial solution constructed by Greedy with the item of the largest marginal value (as opposed to density) and taking the best among such augmentations. Recall that $\mathcal{G}_i$ is the set of the first i items in the greedy solution. Greedy+Max finds for each i an augmenting item s_i which maximizes $f(s_i \cup \mathcal{G}_i)$ among all items that still fit, i.e. $c(s_i \cup G_i) \le K$. The final output is the best among all such augmented solutions. Implementation is given as Algorithm 1. In the rest of this section

Algorithm 1: Offline algorithm Greedy+Max

Input: Set of elements $E = e_1, \ldots, e_n$, knapsack capacity K, cost function $c(\cdot)$, non-negative monotone submodular function f;
Output: $\frac{1}{2}$-approximation for submodular maximization under knapsack constraint;

```
G ← ∅, S ← ∅;
while E ≠ ∅ do
    s ← argmax_{e∈E} f(e | G);
    if f(S) < f(G ∪ s) then
        S ← G ∪ s;
    a ← argmax_{e∈E} ρ(e | G);
    G ← G ∪ a;
    K ← K − c(a);
    Remove all elements e ∈ E with c(e) > K;
return S
```

we show that Greedy+Max gives 1/2-approximation. This analysis is tight as illustrated by the following example:

Example 2.1 *Let e_1, e_2, e_3 be three items such that $f(e_1) = f(e_2) = \frac{1}{2}$ and $f(e_3) = \frac{1}{2} + \epsilon$ for any $\epsilon > 0$. Let $c(e_1) = c(e_2) = \frac{1}{2}$ and $c(e_3) = \frac{1+\epsilon}{2}$. Let f be a linear function, i.e. $f(S) = \sum_{e\in S} f(e)$. Then $\mathsf{OPT} = \{e_1, e_2\}$ has value 1 while Greedy+Max outputs $\{e_3\}$ of value $\frac{1}{2} + \epsilon$.*

As discussed in Section 1.3, our analysis is based on a number of differential inequalities for functions tracking the performance of our algorithm. We assume that these functions are continuous and piecewise smooth, and by $\xi'(x)$ we denote the right-hand derivative of ξ at point x. All these inequalities are of the form $\xi(x) + \alpha\xi'(x) \ge \beta$ for some function ξ, applied in a certain range $[u, v]$ and have some initial condition $\xi(u)$. We frequently need to integrate these inequalities to get a lower bound on $\xi(v)$ which can be done as follows:

Our proof proceeds by case analysis on whether o_1, the item of the largest cost in OPT, is included in the greedy solution G or not. We first show that if $o_1 \in G$, then $f(G)$ is at least a 1/2-approximation.

Let OPT be the optimal solution, i.e. the maximizer of $f(\mathsf{OPT})$ under $c(\mathsf{OPT}) \le K$. Let o_1 be the element of the largest cost in OPT. W.l.o.g. and only for the sake of analysis of approximation we rescale the function values and costs so that $f(\mathsf{OPT}) = 1$ and $c(\mathsf{OPT}) = K = 1$[4]. We first define a greedy performance function $g(x)$ which allows us to track the performance of the greedy solution in a continuous fashion. Let G be the greedy solution computed by Algorithm 1 and let $g_1, g_2, \ldots, g_m$ be the elements in G in the order they were added and recall that $\mathcal{G}_i = \{g_1, \ldots, g_i\}$. For a fixed x, let its *greedy index* i be the smallest index such that $c(\mathcal{G}_i) > x$.

Definition 2.2 (Greedy performance function) *For $x \in [0, 1]$ we define $g(x)$ as:*

$$g(x) = f(\mathcal{G}_{i-1}) + (x - c(\mathcal{G}_{i-1}))\rho\,(g_i\,|\,\mathcal{G}_{i-1})\,.$$

Note that g is a continuous and monotone piecewise-linear function such that $g(0) = 0$. Since an important role in the analysis is played by the derivative of this function we further define g' to be the right derivative for g so that g' is defined everywhere on the interval $[0, c(G))$ and is always non-negative.

We now define a function $g_+(x)$ which tracks the performance of Greedy+Max when the greedy solution collects a set of cost x. Note that the cost of the last item which Greedy+Max uses to augment the solution does not count in the argument of this function.

Definition 2.3 (Greedy+Max performance function) *For any fixed x, let i be the smallest index such that $c(\mathcal{G}_i) > x$. We define $g_+(x) = g(x) + f\,(v\,|\,\mathcal{G}_{i-1})$, where $v = \operatorname{argmax}_{e\in E\setminus\mathcal{G}_{i-1}: c(e\cup\mathcal{G}_{i-1})\le K} f\,(e\,|\,\mathcal{G}_{i-1})$ is the element with the largest marginal gain with respect to the current partial greedy solution $\mathcal{G}_{i-1}$.*

For technical reasons which we describe below instead of working directly with g_+ it is easier to work with a lower bound on it g_1 which has some nicer properties. For g_1 we only consider adding o_1, the largest item from OPT, to the current partial greedy solution. Note that hence g_1 is only defined while this item still fits. Consider the last item added by the greedy solution before the cost of this solution exceeds $1 - c(o_1)$. We define c^* so that $1 - c(o_1) - c^*$ is the cost of the greedy solution before this item is taken.

[4] Note that if $c(\mathsf{OPT}) < K$ then we can set $K = c(\mathsf{OPT})$ first as this does not affect $f(OPT)$.

Definition 2.4 (Greedy+Max performance lower bound) *For $x \in [0, 1 - c(o_1) - c^*]$ we define $g_1(x) = g(x) + f(o_1 \mid \mathcal{G}_{i-1})$ so that $g_1(x) \le g_+(x)$.*

Lemma 2.5 (Greedy+Max inequality) *Let g' denote the right derivative of g. Then for all $x \in [0, 1 - c(o_1) - c^*]$, the following differential inequality holds:*

$$g_1(x) + (1 - c(o_1))g'(x) \ge 1$$

Proof : Similarly to the proof of the standard greedy inequality, it suffices to show the statement only for points where $x = c(\mathcal{G}_{i-1})$ for some $i \ge 1$. Hence, we have $g_1(x) = g(c(\mathcal{G}_{i-1})) + f(o_1 \mid \mathcal{G}_{i-1}) = f(\mathcal{G}_{i-1} \cup o_1)$. Since we normalized $f(\mathsf{OPT}) = 1$, then by monotonicity, $1 = f(\mathsf{OPT}) \le f(\mathcal{G}_{i-1} \cup \mathsf{OPT})$. Hence:

$$\begin{aligned}
1 &\le f(\mathcal{G}_{i-1} \cup \mathsf{OPT}) \\
&= f(\mathcal{G}_{i-1} \cup o_1) + f(\mathsf{OPT} \setminus (o_1 \cup \mathcal{G}_{i-1}) \mid \mathcal{G}_{i-1} \cup o_1) \\
&\le g_1(x) + \sum_{e \in \mathsf{OPT} \setminus (o_1 \cup \mathcal{G}_{i-1})} f(e \mid \mathcal{G}_{i-1} \cup o_1) \\
&= g_1(x) + \sum_{e \in \mathsf{OPT} \setminus (o_1 \cup \mathcal{G}_{i-1})} c(e)\rho(e \mid \mathcal{G}_{i-1} \cup o_1),
\end{aligned}$$

where the second inequality is by submodularity and the definition of g_1 and the last equality is by the definition of marginal density. Since $x \le 1 - c(o_1) - c^*$, then all items in $\mathsf{OPT} \setminus (o_1 \cup \mathcal{G}_{i-1})$ still fit, as o_1 is the largest item in OPT. Since the greedy algorithm always selects the item with the largest marginal density, then $\max_{e \in \mathsf{OPT} \setminus (o_1 \cup \mathcal{G}_{i-1})} \rho(e \mid \mathcal{G}_{i-1} \cup o_1) \le g'(x)$. Hence:

$$\begin{aligned}
1 &\le g_1(x) + \sum_{e \in \mathsf{OPT} \setminus (o_1 \cup \mathcal{G}_{i-1})} c(e)\rho(e \mid \mathcal{G}_{i-1} \cup o_1) \\
&\le g_1(x) + \sum_{e \in \mathsf{OPT} \setminus (o_1 \cup \mathcal{G}_{i-1})} c(e)\rho(e \mid \mathcal{G}_{i-1}) \\
&\le g_1(x) + \sum_{e \in \mathsf{OPT} \setminus (o_1 \cup \mathcal{G}_{i-1})} c(e)g'(x) \\
&= g_1(x) + g'(x) \sum_{e \in \mathsf{OPT} \setminus (o_1 \cup \mathcal{G}_{i-1})} c(e) \\
&= g_1(x) + g'(x)c(\mathsf{OPT} \setminus (o_1 \cup \mathcal{G}_{i-1})) \\
&\le g_1(x) + g'(x)(1 - c(o_1)),
\end{aligned}$$

where the last inequality follows from the normalization of $c(\mathsf{OPT}) \le 1$ and the fact that $o_1 \in \mathsf{OPT}$. □

Theorem 2.6 *Recall that $\tilde{K} = \min(n, K)$ is an upper bound on the number of elements in feasible solutions. Then Greedy+Max gives a 1/2-approximation to the submodular maximization problem under a knapsack constraint and runs in $\mathcal{O}\left(\tilde{K}n\right)$ time.*

Proof : By applying Lemma 2.5 at the point $x = 1 - c(o_1) - c^*$, we have:

$$g_1(1 - c(o_1) - c^*) + (1 - c(o_1))g'(1 - c(o_1) - c^*) \ge 1$$

If $g_1(1-c(o_1)-c^*) \ge \frac{1}{2}$, then we have $\frac{1}{2}$-approximation, because $g_1(1-c(o_1)-c^*)$ is a lower bound on the value of the augmented solution when the cost of the greedy part is $1 - c(o_1) - c^*$. Otherwise:

$$\begin{aligned}
g'(1 - c(o_1) - c^*) &\ge \frac{1 - g_1(1 - c(o_1) - c^*)}{1 - c(o_1)} \\
&> \frac{1}{2(1 - c(o_1))}.
\end{aligned}$$

Note that since $g(0) = 0$ and g' is non-increasing by the definition of Greedy, for any $x \in [0, 1]$ we have $g(x) \ge g'(x) \cdot x$:

$$g(x) \ge \int_{\chi=0}^{x} g'(\chi) d\chi \ge \int_{\chi=0}^{x} g'(x) d\chi = g'(x) \cdot x,$$

Therefore, applying this inequality at $x = 1-c(o_1)-c^*$:

$$\begin{aligned}
g(1 - c(o_1) - c^*) &\ge (1 - c(o_1) - c^*)g'(1 - c(o_1) - c^*) \\
&\ge \frac{1 - c(o_1) - c^*}{2(1 - c(o_1))}.
\end{aligned}$$

Recall that $1 - c(o_1) - c^*$ was the last cost of the greedy solution when we could still augment it with o_1; therefore, the next element e that the greedy solution selects has the cost at least $(1 - c(o_1)) - (1 - c(o_1) - c^*) = c^*$. Thus, the function value after taking e is at least

$$\begin{aligned}
&g(1 - c(o_1) - c^*) + c^* g'(1 - c(o_1) - c^*) \\
&\ge \frac{1 - c(o_1) - c^*}{2(1 - c(o_1))} + \frac{c^*}{2(1 - c(o_1))} = \frac{1}{2}
\end{aligned}$$

Hence, Algorithm 1 gives a $\frac{1}{2}$-approximation to the submodular maximization problem under a knapsack constraint. It remains to analyze the running time and query complexity of Algorithm 1. Since $\tilde{K}$ is the maximum size of a feasible set, Algorithm 1 makes at most $\tilde{K}$ iterations. In each iteration, it makes $\mathcal{O}(n)$ oracle queries, so the total number of queries and runtime is $\mathcal{O}\left(\tilde{K}n\right)$. □

2.2 Streaming algorithm Sieve+Max

Our multi-pass streaming algorithm is given as Algorithm 2. To simplify the presentation, we first give the algorithm under the assumption that it is given a parameter λ, which is a constant-factor approximation of $f(\mathsf{OPT})$. We then show how to remove this assumption using standard techniques in Theorem B.3. As discussed in the description of our techniques Sieve+Max uses $\mathcal{O}(1/\epsilon)$ passes over the data to simulate the execution of Greedy+Max approximately.

Algorithm 2: Multi-pass streaming algorithm Sieve+Max

Input: Stream $e_1, \ldots, e_n$, knapsack capacity K, cost function $c(\cdot)$, non-negative monotone submodular function f, λ which is an α-approximation of $f(\mathsf{OPT})$ for some fixed constant $\alpha>0$, $\epsilon > 0$;
Output: $(1/2 - \epsilon)$-approx. for submodular maximization under a knapsack constraint;

```
T ← ∅, τ ← λ/(αK);
while τ > λ/(2K) do            // Thresholding stage
    Take a new pass over the stream;
    for each read item e do
        if ρ(e | T) ≥ τ and c(e ∪ T) ≤ K then
            T ← T ∪ {e};
    τ ← τ/(1 + ε);
For each i, let G_i be the first i items selected in
  the construction of T above and initialize s_i = ∅
  for the best augmenting item for G_i;
Take a pass over the stream;
for each read item e do // Augmentation stage
    if e ∉ T then
        j = max{i | c(G_i) + c(e) ≤ K};
        if f(G_j ∪ s_j) < f(G_j ∪ e) then
            s_j ← {e};
return argmax f(G_i ∪ s_i)
```

In the analysis, which gives the proof of Theorem 1.2, we define functions t and t_1 analogous to g and g_1 respectively, based on $\mathcal{T}_i$, the first i items collected by the thresholding algorithm. We show that t and t_1 satisfy the same differential inequalities as g and g_1 respectively, up to $(1+\epsilon)$ factors, and similar to before, our analysis then proceeds by casework on whether o_1, the largest item in OPT, is included in the thresholding solution T or not.

We first show that if $o_1 \in T$, then $f(T)$ is at least a $\left(\frac{1}{2} - \epsilon\right)$-approximation.

Let T be the set of items constructed Sieve+Max (as in Algorithm 2) and let $t_1, t_2, \ldots$ be the order that they are collected. We refer to the part of the algorithm which constructs T as "thresholding" and the rest as "augmentation" below. We use $\mathcal{T}_i$ to denote the set containing the i items $\{t_1, t_2, \ldots, t_i\}$. We again use o_1 to denote the item with highest cost in OPT. Similar to the above, we define two functions representing the values of our thresholding algorithm, and augmented solutions given the utilized proportion of the knapsack.

Definition 2.7 (Thresholding performance function) *For any $x \in [0,1]$, let i be the smallest index such that $c(\mathcal{T}_i) > x$. We define $t(x) = f(\mathcal{T}_{i-1}) + (x - c(\mathcal{T}_{i-i}))\rho\left(t_i \,|\, \mathcal{T}_{i-1}\right)$ and $t'(x)$ to be the right derivative of t.*

We define a function $t_1(x)$ that lower bounds the performance of Sieve+Max when the thresholding solution collects a set of cost x:

Definition 2.8 (Sieve+Max performance function and lower bound) *For any fixed x, let i be the smallest index such that $c(\mathcal{T}_i) > x$. Then we define $t_1(x) = t(x) + f\left(o_1 \,|\, \mathcal{T}_{i-1}\right)$, where $o_1 = \operatorname{argmax}_{e\in\mathsf{OPT}} c(e)$.*

In order to analyze the output of the algorithm, we prove a differential inequality for t_1. If $c(T) \geq 1 - c(o_1)$ then let $c^* \geq 0$ be defined so that $1 - c(o_1) - c^*$ is the cost of the thresholding solution before the algorithm takes the item which makes the cost exceed $1 - c(o_1)$.

Lemma 2.9 (Sieve+Max Inequality) *If $c(T) \geq 1 - c(o_1)$ then for all $x \in [0, 1 - c\left(o_1\right) - c^*]$, then t and t_1 satisfy the following differential inequality:*

$$t_1(x) + (1+\epsilon)(1 - c\left(o_1\right))t'(x) \geq 1.$$

Proof : First, note that for $x \in [0,p]$ where p is the total cost of items taken in the first pass the inequality holds trivially since $t'(x) \geq 1$ (as in the proof of the standard thresholding inequality). Hence assume that $x \in [p, 1 - c(o_1) - c^*]$ is fixed and consider any pass after the first one. Similarly to other proofs it suffices to only consider left endpoints of the intervals of the form $[c(\mathcal{T}_{i-1}), c(\mathcal{T}_i))$ so let $x = c(\mathcal{T}_{i-1})$. Since we normalized $f(\mathsf{OPT}) = 1$, then by monotonicity, $1 = f(\mathsf{OPT}) \leq f(\mathcal{T}_{i-1} \cup \mathsf{OPT})$. Hence:

$$\begin{aligned}
1 &\leq f(\mathcal{T}_{i-1} \cup \mathsf{OPT}) \\
&= f((\mathcal{T}_{i-1} \cup o_1) \cup (\mathsf{OPT} \setminus o_1)) \\
&= f(\mathcal{T}_{i-1} \cup o_1) + f\left(\mathsf{OPT} \setminus (o_1 \cup \mathcal{T}_{i-1}) \,|\, \mathcal{T}_{i-1} \cup o_1\right) \\
&\leq t_1(x) + \sum_{e\in\mathsf{OPT}\setminus(o_1\cup\mathcal{T}_{i-1})} f\left(e \,|\, \mathcal{T}_{i-1} \cup o_1\right) \\
&= t_1(x) + \sum_{e\in\mathsf{OPT}\setminus(o_1\cup\mathcal{T}_{i-1})} c(e)\rho\left(e \,|\, \mathcal{T}_{i-1} \cup o_1\right),
\end{aligned}$$

where the second inequality is by submodularity and the last line is by the definition of marginal density. Since o_1 has the maximum cost in OPT. $x \leq 1 - c\left(o_1\right)$, all items in $\mathsf{OPT} \setminus (o_1 \cup \mathcal{T}_{i-1})$ still fit into the remaining knapsack capacity. In all passes after the first one, the thresholding algorithm always selects an element which gives $\frac{1}{1+\epsilon}$-approximation of the highest possible marginal density:

$$(1+\epsilon)t(x) \geq \max_{e\in\mathsf{OPT}\setminus(o_1\cup\mathcal{T}_{i-1})} \rho\left(e \,|\, o_1 \cup \mathcal{T}_{i-1}\right).$$

Combining with the inequality above:

$$\begin{aligned}1 &\le t_1(x) + \sum_{e\in \mathsf{OPT}\setminus(o_1\cup\mathcal{T}_{i-1})} c(e)\rho\left(e\,|\,\mathcal{T}_{i-1}\cup o_1\right)\\ &\le t_1(x) + (1+\epsilon)t'(x)\sum_{e\in \mathsf{OPT}\setminus(o_1\cup\mathcal{T}_{i-1})} c(e)\\ &= t_1(x) + (1+\epsilon)t'(x)c(\mathsf{OPT}\setminus(o_1\cup\mathcal{T}_{i-1}))\\ &\le t_1(x) + (1+\epsilon)t'(x)(1-c\left(o_1\right)),\end{aligned}$$

where the last equality is by the normalization of $c(\mathsf{OPT}) = 1$ and the fact that $o_1 \in \mathsf{OPT}$.

□

Theorem 2.10 *There exists an algorithm that uses $\mathcal{O}\left(\tilde{K}\right)$ space and $\mathcal{O}\left(1/\epsilon\right)$ passes over the stream, makes $\mathcal{O}\left(n/\epsilon + n\log\tilde{K}\right)$ queries, and outputs a $(1/2-\epsilon)$-approximation to the submodular maximization problem under a knapsack constraint.*

Proof : We can use existing algorithm from Theorem B.3 to obtain a constant factor approximation λ to $f(\mathsf{OPT})$. We thus analyze the correctness of Algorithm 2 given an input λ that is a constant factor approximation to $f(\mathsf{OPT})$. The proof is similar to proof of Theorem 2.6.

By applying Lemma 2.9 at the point $x = 1-c\left(o_1\right)-c^*$, we have:

$$t_1(1-c\left(o_1\right)-c^*)+(1+\epsilon)(1-c\left(o_1\right))t'(1-c\left(o_1\right)-c^*) \ge 1$$

If $t_1(1-c\left(o_1\right)-c^*) \ge \frac{1}{2}$, then we have $\frac{1}{2}$-approximation, because $t_1(1-c(o_1)-c^*)$ is a lower bound on the value of the augmented solution when the cost of the thresholding solution is $1-c(o_1)-c^*$. Otherwise:

$$\begin{aligned}t'(1-c\left(o_1\right)-c^*) &\ge \frac{1-g_1(1-c\left(o_1\right)-c^*)}{(1-c\left(o_1\right))(1+\epsilon)}\\ &> \frac{1}{2(1-c\left(o_1\right))(1+\epsilon)}\end{aligned}$$

Note that since $t(0) = 0$, for any $x \in [0,1]$ we have $t(x) \ge \frac{t'(x)\cdot x}{1+\epsilon}$:

$$t(x) \ge \int_{\chi=0}^{x} t'(\chi)d\chi \ge \int_{\chi=0}^{x}\frac{t'(x)}{1+\epsilon}d\chi = \frac{t'(x)\cdot x}{1+\epsilon},$$

where we used the fact that t' is a $\frac{1}{1+\epsilon}$ approximation of the maximum marginal density, which does not increase. Therefore, applying this at $x = 1-c\left(o_1\right)-c^*$:

$$\begin{aligned}t(1-c\left(o_1\right)-c^*) &\ge (1-c\left(o_1\right)-c^*)t'(1-c\left(o_1\right)-c^*)\\ &\ge \frac{1-c\left(o_1\right)-c^*}{2(1-c\left(o_1\right))(1+\epsilon)}.\end{aligned}$$

Recall that $1-c\left(o_1\right)-c^*$ was the last cost of the thresholding solution when we could still augment it with o_1; therefore, the next element e that the thresholding solution selects has the cost at least $(1-c\left(o_1\right)) - (1-c\left(o_1\right)-c^*) = c^*$. Thus, the function value after taking e is at least

$$\begin{aligned}&g(1-c\left(o_1\right)-c^*) + c^*g'(1-c\left(o_1\right)-c^*)\\ &\ge \frac{1-c\left(o_1\right)-c^*}{2(1-c\left(o_1\right))(1+\epsilon)} + \frac{c^*}{2(1-c\left(o_1\right))(1+\epsilon)}\\ &= \frac{1}{2(1+\epsilon)} = \frac{1}{2} - \frac{\epsilon}{2(1+\epsilon)} \ge \frac{1}{2}-\epsilon.\end{aligned}$$

Hence, Algorithm 2 gives a $\left(\frac{1}{2}-\epsilon\right)$-approximation to the submodular maximization problem under knapsack constraints, given a constant factor approximation to $f(\mathsf{OPT})$. Note that it suffices to consider only thresholds up to $\frac{\tau}{2K}$ since $t'(x) < \frac{1}{2}$ implies that $t(x) > \frac{1}{2}$ by Lemma A.3.

Using existing algorithms to obtain a constant factor approximation λ (e.g., by setting $\epsilon = \frac{1}{6}$ in Theorem B.3) that use additional $\mathcal{O}\left(n\log\tilde{K}\right)$ queries, then correctness of Algorithm 2 follows. It remains to analyze the space and query complexity of Algorithm 2. Since each item has cost at least 1, at most K items are stored by the thresholding algorithm, and at most K items are stored by the augmented solution S. Hence, the space complexity of Algorithm 2 is $\mathcal{O}\left(\tilde{K}\right)$. If τ is an α-approximation to $f(\mathsf{OPT})$ for some constant α, then the algorithm makes $\log_{1+\epsilon}\frac{1}{2\alpha} = \mathcal{O}\left(\frac{1}{\epsilon}\right)$ passes over the input stream. Each pass makes at most n queries, so the number of queries is at most $\mathcal{O}\left(\frac{n}{\epsilon}\right)$. □

2.3 Distributed algorithm Distributed Sieve+Max

In this section, we assume that there are $m = \sqrt{n/\tilde{K}}$ machines $M_1,\ldots,M_m$, each with $O\left(\frac{n}{m}\right) = O(\sqrt{n\tilde{K}})$ amount of local memory. Our distributed algorithm (Algorithm 3) follows a similar thresholding approach as our streaming algorithm: at each round, machines collect items whose marginal densities exceed the threshold corresponding to the round. Our algorithm and proof are based on Liu and Vondrák (2019).

We require the following form of Azuma's inequality for submartingales.

Theorem 2.11 (Azuma's Inequality) *Suppose $X_0, X_1, \ldots, X_n$ is a submartingale and $|X_i - X_{i+1}| \le c_i$. Then*

$$\Pr\left[X_n - X_0 \le -t\right] \le exp\left(\frac{-t^2}{2\sum_i c_i^2}\right).$$

Algorithm 3: DISTRIBUTED SIEVE+MAX: A $O(1/\epsilon)$-round MapReduce algorithm for submodular maximization under knapsack constraints.

Input: Set of elements $E = e_1, \ldots, e_n$, knapsack capacity K, cost function $c(\cdot)$, non-negative monotone submodular function f, τ that is α-approximation of $f(\mathsf{OPT})$ for some constant $\alpha > 0$;
Output: A set S that is a $(\frac{1}{2} - \epsilon)$-approximation for submodular maximization with a knapsack constraint;

```
T ← ∅, t ← τ/(αK), K̃ ← min(n, K);
while t > τ/(2K) do
    On the central machine C:
    Form Γ by sampling each e ∈ E with
      probability 4√(K̃/n);
    for each item e ∈ Γ do
        if ρ(e|T) > t and c(e ∪ T) ≤ K then
            T ← T ∪ {e};
    if c(T) ≥ K then
        break

    Partition E randomly into sets V_1, V_2, ... V_m;
    Send T and V_i to machine M_i for all i;
    On each machine M_i:
    X_i ← T;
    for each item e ∈ V_i do
        if ρ(e|X_i) > t and c(e ∪ X_i) ≤ K then
            X_i ← X_i ∪ {e};
    X_i ← X_i \ T;
    Send X_i to C;

    On the central machine C:
    for each item e ∈ ∪X_i do
        if ρ(e|T) > t and c(e ∪ T) ≤ K then
            T ← T ∪ {e};
    t ← t/(1+ε)
Send T to all machines;

On each machine M_i:
For each i, let G_i denote the first i items that a
  greedy algorithm would select from T and
  initialize s_i ← ∅;
for each item e ∈ V_i \ T do
    j ← max {i | c(e) + c(G_i) ≤ K};
    if f(G_j ∪ s_j) < f(G_j ∪ e) then
        s_j ← e;
Send argmax {f(G_i ∪ s_i)} to C;
return argmax of solutions received in C
```

We first bound the total number of elements sent to the central machine.

Lemma 2.12 *In Algorithm 3, with probability $1 - e^{-\Omega(\tilde{K})}$, the total number of elements sent to the central machine is $\sqrt{n\tilde{K}}$.*

Proof : Since each element is sampled with probability $4\sqrt{\tilde{K}/n}$, the expected number of elements in Γ is $4\sqrt{n\tilde{K}}$ for any round i. Hence $|\Gamma| \geq 3\sqrt{n\tilde{K}}$ with probability at least $1 - e^{-\Omega(\tilde{K})}$ by a standard Chernoff bound. Let N denote the total number of elements with marginal density at least $\frac{f(\mathsf{OPT})}{(1+\epsilon)^i\tilde{K}}$ with respect to T, so that the number of elements sent to the central unit in one round is $N + |\Gamma|$.

For the sake of analysis, suppose that Γ is randomly partitioned into at least $3\tilde{K}$ chunks of size $\sqrt{n/\tilde{K}}$ elements, with the chunks being processed sequentially. If there are fewer than $\sqrt{n\tilde{K}}$ remaining elements whose marginal density with respect to T_{i-1} exceeds $\frac{f(\mathsf{OPT})}{(1+\epsilon)^i\tilde{K}}$, then at most $\sqrt{n\tilde{K}}$ elements are sent to the central machine.

Otherwise, there are at least $\sqrt{n\tilde{K}}$ remaining elements whose marginal density with respect to T exceeds $\frac{f(\mathsf{OPT})}{(1+\epsilon)^i\tilde{K}}$. Then for each block, with probability at least $1 - \left(1 - \sqrt{\tilde{K}/n}\right)^{\sqrt{n/\tilde{K}}} > 1/2$, an additional element is added to N_i. To use a martingale argument to bound the number of elements selected in Γ_i, we let X_i be the indicator random variable for the event that at least one element is selected from the i^{th} block so that we have $E[X_i \mid X_1, \ldots, X_{i-1}] \geq \frac{1}{2}$. Let $Y_i = \sum_{j=1}^{i}(X_i - 1/2)$ so that the sequence $Y_1, Y_2, \ldots$ is a submartingale, i.e., $E[Y_i \mid Y_1, \ldots, Y_{i-1}] \geq Y_{i-1}$ and $|Y_i - Y_{i-1}| \leq 1$. By Azuma's inequality (Theorem 2.11), $\Pr[Y_{3\tilde{K}} < -\frac{1}{2}\tilde{K}] < e^{-\Omega(\tilde{K})}$, so that $\sum_{j=1}^{3\tilde{K}} X_j = Y_K + \frac{3}{2}\tilde{K} \geq \tilde{K}$ with probability at least $1 - e^{-\Omega(\tilde{K})}$, in which case no elements are sent to the central machine. □

We now analyze the approximation guarantee and performance of Algorithm 3.

Theorem 2.13 *There exists an algorithm* DISTRIBUTED SIEVE+MAX *which uses $\mathcal{O}(1/\epsilon)$ rounds of communication between $\sqrt{n/\tilde{K}}$ machines, each with $\mathcal{O}\left(\sqrt{n\tilde{K}}\right)$ memory. With high probability, the total number of elements sent to the central machine is $\sqrt{n\tilde{K}}$ and the algorithm outputs a $(1/2 - \epsilon)$-approximation to the submodular maximization problem with a knapsack constraint.*

Proof : Correctness follows from the observation that the algorithm performs thresholding in the same manner as Algorithm 2. The space bounds follow from Lemma 2.12. □

2.4 Query lower bound

We show a simple query lower bound under the standard assumption Norouzi-Fard et al. (2018); Kazemi et al. (2019) that the algorithm only queries f on feasible sets.

Theorem 2.14 *For $\alpha > 1/2$, any α-approximation algorithm for maximizing a function f under a knapsack constraint that succeeds with constant probability and only queries values of the function f on feasible sets (i.e. sets of cost at most K) must make at least $\Omega(n^2)$ queries if f is either: 1) non-monotone submodular, 2) monotone and submodular on the feasible sets, 3) monotone subadditive.*

Proof : Let $e_1, \ldots, e_n$ be the set of elements and set $c(e_i) = K/2$ for all i. By Yao's principle it suffices to consider two hard distributions $\mathcal{D}_{1/2}$ and $\mathcal{D}_1$ such that the optimum for every instance in the support of these distributions is $1/2$ and 1 respectively and then show that no algorithm making $o(n^2)$ deterministic queries can distinguish the two distributions with constant probability. The distributions $\mathcal{D}_{1/2}$ and $\mathcal{D}_1$ are as follows:

- $\mathcal{D}_{1/2}$ has $f(S) = 1/2$ for all $S \neq \emptyset$.
- $\mathcal{D}_1$ is constructed by picking two items $e_i \neq e_j$ uniformly at random and assigning $f(S) = 1$ for $S = \{e_i, e_j\}$. Otherwise, set $f(S) = 1/2$ for all $S \neq \emptyset$ and $S \neq \{e_i, e_j\}$.

Fix the set of deterministic queries Q that the algorithm makes. Since the algorithm is only allowed to make queries to sets of cost at most K, all sets in Q have size at most two. Furthermore, note that $f(e_i) = 1/2$ for all i under both $\mathcal{D}_{1/2}$ and $\mathcal{D}_1$. Thus, only queries to sets of size exactly two can help the algorithm distinguish the two distributions. All such queries give value $1/2$ under both distributions except for a single query (i, j) under $\mathcal{D}_1$ which gives value 1. Since (i, j) is chosen uniformly at random under $\mathcal{D}_1$ the probability that a fixed set Q contains it is given as $|Q|/\binom{n}{2}$. Hence if the algorithm succeeds with a constant probability then it must be the case that $|Q| = \Omega(n^2)$.

Note that the construction of f results in a non-monotone submodular function but f is monotone when restricted to feasible sets of size at most two items. By changing $\mathcal{D}_1$ so that the functions in this distribution take value 1 on all sets of size more than 2 one can ensure monotonicity of f. However, f is still submodular on the feasible sets and subadditive everywhere (recall that a subadditive function satisfies $f(S) + f(T) \geq f(S \cup T)$ for all $S, T \subseteq U$). □

3 Experimental results

We compare our offline algorithm Greedy+Max and our streaming algorithm Sieve+Max with baselines, answering the following questions: (1) What are the approximation factors we are getting on real data? (2) How do the objective values compare? (3) How do the runtimes compare? (4) How do the numbers of queries compare? We compare Greedy+Max to the following baselines:

(1) **PartialEnum+Greedy** Sviridenko (2004). Given an input parameter d, this algorithm creates a separate knapsack for each combination of d items, and then runs the Greedy algorithm on each of the knapsacks. At the end, the algorithm outputs the best solution among all knapsacks, so that the total runtime is $\Omega(Kn^{d+1})$. In fact, PartialEnum+Greedy is only feasible for $d = 1$ and our smallest dataset.

(2) **Greedy**. This algorithm starts with an empty knapsack and repeatedly adds the item with the highest marginal density with respect to the collected items in the knapsack, until no more item can be added to the knapsack.

(3) **GreedyOrMax** Khuller et al. (1999). This algorithm compares the value of the best item with the value of the output of the Greedy algorithm and outputs the better of the two.

In streaming we compare Sieve+Max to Sieve Badanidiyuru et al. (2014) and SieveOrMax Huang et al. (2017), which are similar thresholding-based algorithms. Sieve starts with an empty knapsack and collects all items whose marginal density with respect to the items in the knapsack exceed a given threshold (which is initially equal to $\frac{1}{2}$), while SieveOrMax uses a similar approach, but compares the items collected by the thresholding algorithm to the best single item, and outputs the better of the two solutions. We also implemented a single-pass BranchingMRT by Huang et al. (2017) that uses thresholding along with multiple branches and gives a $4/11 \approx 0.36$-approximation. We did not implement Huang and Kakimura (2019) as their algorithms are orders of magnitude slower than BranchingMRT which is already several orders of magnitude slower than other algorithms.

Our code is available at https://github.com/

`aistats20submodular/aistats20submodular`.

3.1 Objectives and Datasets

Graph coverage. For a graph $G(V, E)$ and $Z \subset V$, the objective is to maximize the neighborhood vertex coverage function $f(Z) := |Z \cup N(Z)|/|V|$, where $N(Z)$ is the set of neighbors of Z. The cost of each node is roughly proportional to the value of the node. Specifically, the cost of each node $v \in V$ is $c(v) = \frac{\beta}{|V|}(|N(v)| - \alpha)$, where $\alpha = \frac{1}{20}$ and β is a normalizing factor so that $c(v) \geq 1$, so that the cost of each node is roughly proportional to the value of the node. We ran experiments on two graphs from SNAP Leskovec and Krevl (2014): 1) `ego-Facebook`(4K vertices, 81K edges), 2) `com-DBLP` (317K vertices, 1M edges).

Movie ratings. We also analyze a dataset of movies to model the scenario of movie recommendation. The objective function, defined as in Avdiukhin et al. (2019), is maximized for a set of movies that is similar to a user's interests and the cost of a movie is set to be roughly proportional to its value. Each movie is assigned a rating in the range $[1, 5]$ by users. Let $r_{x,u}$ be the rating assigned by user u to movie x and r_{avg} be the average rating across all movies. For each movie x, we normalize the ratings to produce a vector v_x by setting $v_{x,u} = 0$ if user u did not rate movie x and $v_{x,u} = r_{x,u} - r_{avg}$ otherwise. We then define the similarity between two movies x_1 and x_2 as the dot product $\langle v_{x_1}, v_{x_2} \rangle$ of their vectors. Given a set X of movies, to quantify how representative a subset of movies Z is, we consider a parameterized objective function $f_X(Z) = \sum_{x \in X} \max_{z \in Z} \langle v_z, v_x \rangle$. Hence, the maximizer of $f_X(Z)$ corresponds to a set of movies that is similar to the user's interests. We analyze the `ml-20` MovieLens dataset GroupLens (2015), which contains approximately $27K$ movies and $20M$ ratings.

3.2 Results

We first give instance-specific approximation factors for different values of K for offline (Fig. 2) and streaming (Fig. 3) algorithms. These approximations are computed using upper bounds on $f(\mathsf{OPT})$ which can be obtained using the analysis of Greedy. Greedy+Max and Sieve+Max typically perform at least 20% better than their $1/2$ worst-case guarantees. In fact, our results show that the output value can be improved by up to 50%, both by Greedy+Max upon Greedy (Figure 4) and by Sieve+Max upon Sieve (Figure 5).

Running time. We point out that the runtimes of Greedy+Max and GreedyOrMax algorithms are similar, being at most 20% greater than the runtime of Greedy, as shown in Figure 6. On the other hand, even though PartialEnum+Greedy does not outperform Greedy+Max, it is only feasible for $d = 1$ and the `ego-Facebook` dataset and uses on average almost 500 times as much runtime for $K = 10$ across ten iterations of each algorithm, as shown in Figure 6. The runtimes of Sieve+Max, SieveOrMax, and Sieve are generally similar; however in the case of the `com-dbpl` dataset, the runtime of Sieve+Max grows with K. This can be explained by the fact that oracle calls on larger sets typically require more time, and augmented sets typically contain more elements than sets encountered during execution of Sieve. On the other hand, the runtime of BranchingMRT was substantially slower, and we did not include its runtime for scaling purposes, as for $K = 5$, the runtime of BranchingMRT was already a factor 80K more than Sieve. Error bars for the standard deviations of the runtimes of the streaming algorithms are given in Figure 8.

Oracle calls. We also compare the number of oracle calls performed by the algorithms. Greedy+Max, GreedyOrMax and Greedy require the same amount of oracle calls, since computing marginal gains and finding the best element for augmentation compute the objective on the same set. On the other hand, PartialEnum+Greedy requires 544x more calls than Greedy for $K = 8$. For the streaming algorithms, the number of oracle calls made by Sieve, Sieve+Max, and Sieve, never differed by more than a factor of two, while BranchingMRT requires a factor 125K more oracle calls than Sieve for $K = 8$. We illustrate the number of oracle calls made by these algorithms in Figure 9.

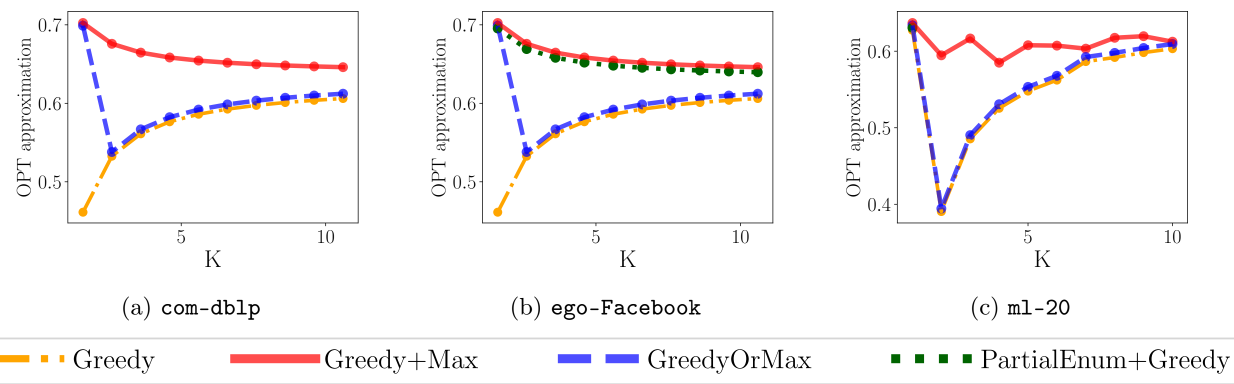


Fig. 2: Instance-specific approximations for different K. Greedy+Max performs substantially better than its worst-case $1/2$-approximation guarantee and typically beats even the $(1 - 1/e) \approx 0.63$ bound. Despite much higher runtime, PartialEnum+Greedy does not beat Greedy+Max even on the only dataset where its runtime is feasible (`ego-Facebook`).

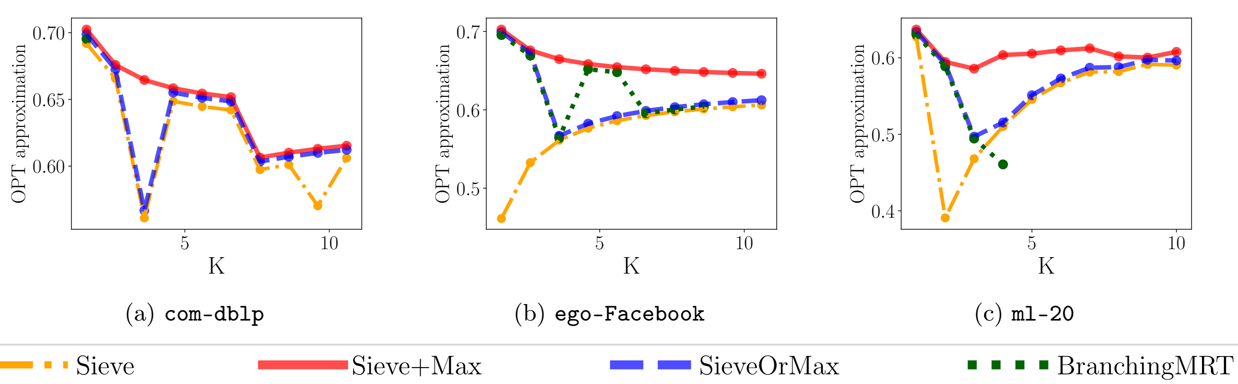
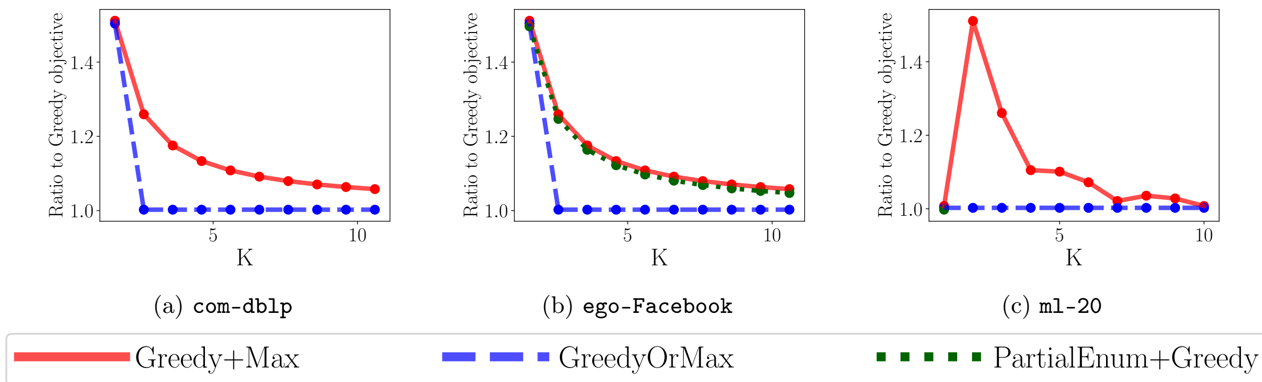
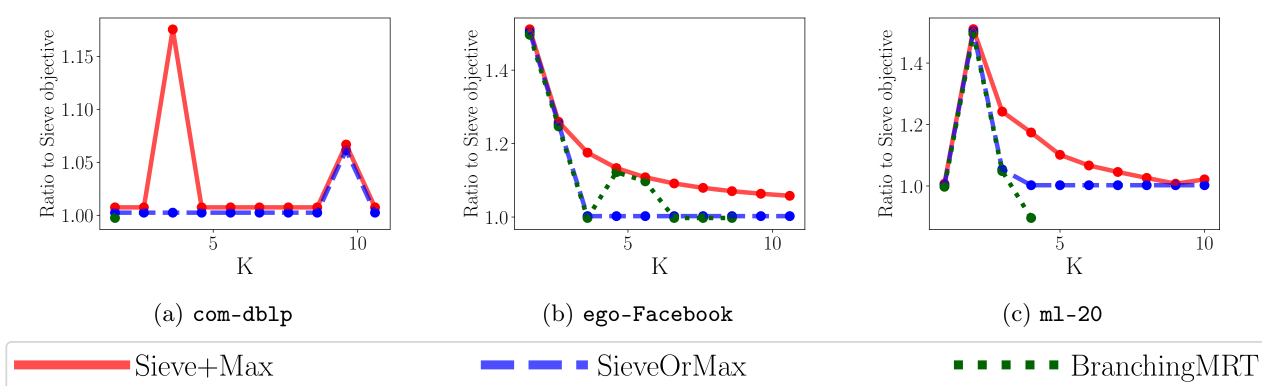


Fig. 3: Instance-specific approximations for different K. Sieve+Max performs substantially better than its worst-case $(1/2 - \epsilon)$-approximation guarantee and robustly dominates all other approaches. It can improve by up to 40% upon Sieve. Despite much higher runtime, BranchingMRT does not beat Sieve+Max (some data points not shown for BranchingMRT as it did not terminate under a 200-second time limit).



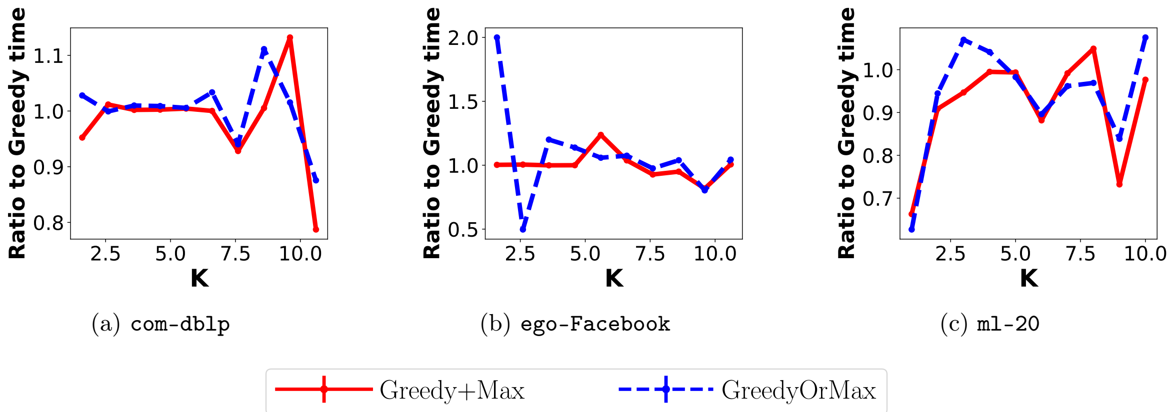
(a) `com-dblp` (b) `ego-Facebook` (c) `ml-20`

Fig. 4: Ratio of the objective of offline algorithms to the objective of GREEDY for different values of K. GREEDY+MAX can improve by almost 50% upon GREEDY, but by definition, GREEDY+MAX and GREEDYORMAX cannot perform worse than GREEDY. Despite its runtime, PARTIALENUM+GREEDY does not outperform GREEDY+MAX on the `ego-Facebook` dataset.



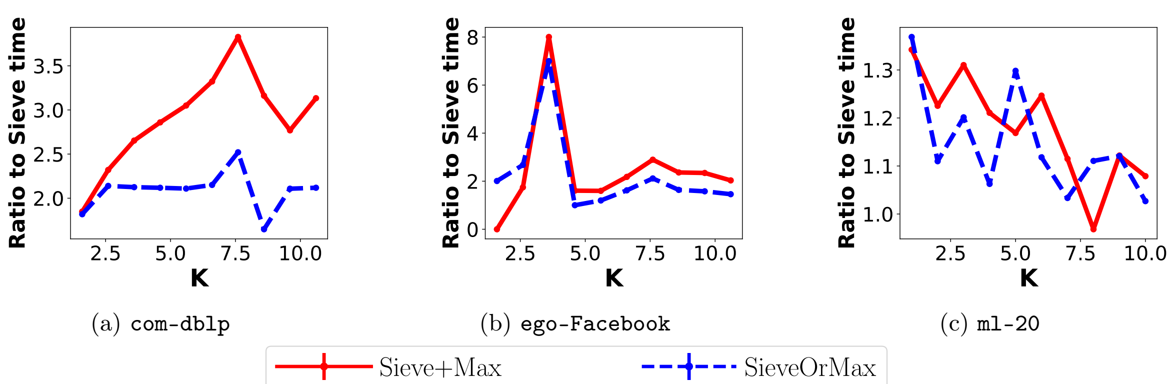
(a) `com-dblp` (b) `ego-Facebook` (c) `ml-20`

Fig. 5: Ratio of the objective of streaming algorithms to the objective of SIEVE for different values of K. SIEVE+MAX can improve by almost 40% upon SIEVE, but by definition, SIEVE+MAX and SIEVEORMAX cannot perform worse than SIEVE. Despite its runtime, BRANCHINGMRT does not outperform SIEVE+MAX.



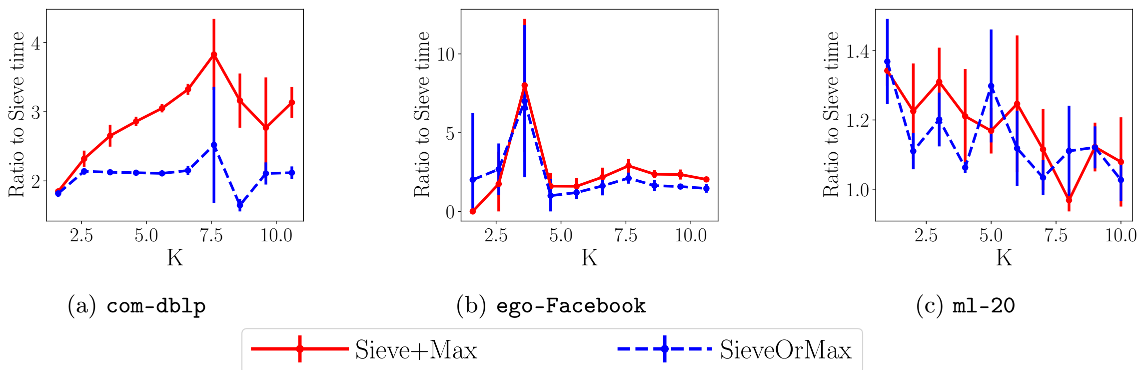
(a) `com-dblp` (b) `ego-Facebook` (c) `ml-20`

Fig. 6: Ratio of runtime of offline algorithms to the runtime of Greedy, for different values of K. Observe that Greedy+Max and GreedyOrMax show similar running time, which is at most 20% greater than Greedy running time. The ratio of PartialEnum+Greedy runtime is not displayed, due to it being several orders of magnitude larger, e.g., 1000 times larger for $K = 15$.



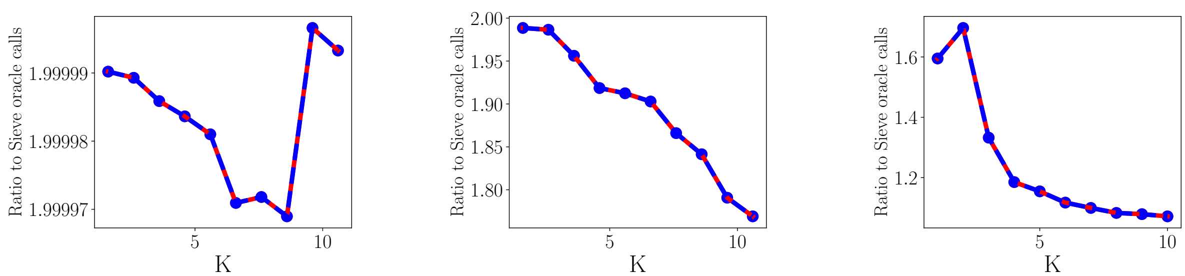
(a) `com-dblp` (b) `ego-Facebook` (c) `ml-20`

Fig. 7: Ratio of average runtime of streaming algorithms to the average runtime of Sieve for different values of K, across ten iterations. The larger ratios can be explained from the oracle calls made on larger sets by Sieve+Max being more expensive than the average oracle call made by Sieve. The ratio of BranchingMRT runtime is not displayed, due to being several orders of magnitude larger, e.g., 80K times larger for $K = 5$.



(a) `com-dblp` (b) `ego-Facebook` (c) `ml-20`

Fig. 8: Ratio of average runtime of streaming algorithms compared to the average runtime of Sieve, with error bars representing one standard deviation for each algorithm on the corresponding knapsack constraint across ten iterations.



(a) `com-dblp` (b) `ego-Facebook` (c) `ml-20`

Fig. 9: Smoothed ratio of average number of oracle calls made by streaming algorithms compared to the average number of oracles calls made by Sieve, across ten iterations.

References

Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular secretary problem with shortlists. In *10th Innovations in Theoretical Computer Science Conference, ITCS*, 2019.

Nima Anari, Nika Haghtalab, Seffi Naor, Sebastian Pokutta, Mohit Singh, and Alfredo Torrico. Structured robust submodular maximization: Offline and online algorithms. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 3128–3137. PMLR, 16–18 Apr 2019. URL http://proceedings.mlr.press/v89/anari19a.html.

Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC*, pages 574–583, 2014.

Dmitrii Avdiukhin, Slobodan Mitrovic, Grigory Yaroslavtsev, and Samson Zhou. Adversarially robust submodular maximization under knapsack constraints. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD.*, pages 148–156, 2019.

Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1497–1514, 2014.

Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 671–680. ACM, 2014.

Eric Balkanski and Yaron Singer. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1138–1151, 2018.

Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An exponential speedup in parallel running time for submodular maximization without loss in approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 283–302, 2019.

MohammadHossein Bateni, Hossein Esfandiari, and Vahab S. Mirrokni. Optimal distributed submodular optimization via sketching. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 1138–1147, 2018.

Ilija Bogunovic, Slobodan Mitrović, Jonathan Scarlett, and Volkan Cevher. Robust submodular maximization: A non-uniform partitioning approach. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, pages 508–516, 2017.

Niv Buchbinder, Moran Feldman, and Roy Schwartz. Online submodular maximization with preemption. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1202–1216, 2015.

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.

Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: Matchings, matroids, and more. In *Integer Programming and Combinatorial Optimization - 17th International Conference, IPCO. Proceedings*, pages 210–221, 2014.

T.-H. Hubert Chan, Zhiyi Huang, Shaofeng H.-C. Jiang, Ning Kang, and Zhihao Gavin Tang. Online submodular maximization with free disposal: Randomization beats 0.25 for partition matroids. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1204–1223, 2017.

Chandra Chekuri and Kent Quanrud. Submodular function maximization in parallel via the multilinear relaxation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 303–322, 2019.

Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP, Proceedings, Part I*, pages 318–330, 2015.

Rafael da Ponte Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. The power of randomization: Distributed submodular maximization on massive datasets. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, pages 1236–1244, 2015.

Rafael da Ponte Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. A new framework for distributed submodular maximization. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS*, pages 645–654, 2016.

Khalid El-Arini and Carlos Guestrin. Beyond keyword search: discovering relevant scientific literature. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 439–447. ACM, 2011.

Ethan R. Elenberg, Alexandros G. Dimakis, Moran Feldman, and Amin Karbasi. Streaming weak submodularity: Interpreting neural networks on the fly. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 4047–4057, 2017.

Alina Ene and Huy L. Nguyen. A nearly-linear time algorithm for submodular maximization with a knapsack constraint. In *46th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 53:1–53:12, 2019a.

Alina Ene and Huy L. Nguyen. Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 274–282, 2019b.

Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. Submodular optimization over sliding windows. In *Proceedings of the 26th International Conference on World Wide Web, WWW*, pages 421–430, 2017.

Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 255–273, 2019.

Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS*, pages 730–740, 2018.

Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

Ryan Gomes and Andreas Krause. Budgeted nonparametric learning from data streams. In *Proceedings of the 27th International Conference on Machine Learning, ICML*, pages 391–398, 2010.

GroupLens. https://grouplens.org/datasets/movielens, 2015. MovieLens Datasets.

Jason Hartline, Vahab Mirrokni, and Mukund Sundararajan. Optimal marketing strategies over social networks. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 189–198, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-085-2. doi: 10.1145/1367497.1367524. URL http://doi.acm.org/10.1145/1367497.1367524.

Chien-Chung Huang and Naonori Kakimura. Multipass streaming algorithms for monotone submodular function maximization. In *Algorithms and Data Structures - 16th International Symposium, WADS*, 2019. (to appear). https://arxiv.org/abs/1802.06212.

Chien-Chung Huang, Naonori Kakimura, and Yuichi Yoshida. Streaming algorithms for maximizing monotone submodular functions under a knapsack constraint. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 11:1–11:14, 2017.

Rishabh Iyer and Jeffrey Bilmes. Near optimal algorithms for hard submodular programs with discounted cooperative costs. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 276–285. PMLR, 16–18 Apr 2019. URL http://proceedings.mlr.press/v89/iyer19a.html.

Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 938–948, 2010.

Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, pages 3311–3320, 2019.

Rajiv Khanna, Joydeep Ghosh, Russell Poldrack, and Oluwasanmi Koyejo. Sparse Submodular Probabilistic PCA. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 453–461, San Diego, California, USA, 09–12 May 2015. PMLR. URL http://proceedings.mlr.press/v38/khanna15.html.

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

Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. *ACM Transactions on Parallel Computing (TOPC)*, 2(3):14, 2015.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2007.

Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 510–520. Association for Computational Linguistics, 2011.

Erik M. Lindgren, Shanshan Wu, and Alexandros G. Dimakis. Leveraging sparsity for efficient submodular data summarization. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems*, pages 3414–3422, 2016.

Paul Liu and Jan Vondrák. Submodular optimization in the mapreduce model. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019*, pages 18:1–18:10, 2019.

Vahab S. Mirrokni and Morteza Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC*, pages 153–162, 2015.

Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems*, pages 2049–2057, 2013.

Baharan Mirzasoleiman, Morteza Zadimoghaddam, and Amin Karbasi. Fast distributed submodular cover: Public-private data summarization. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems*, pages 3594–3602, 2016.

Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18)*, pages 1379–1386, 2018.

Slobodan Mitrović, Ilija Bogunovic, Ashkan Norouzi-Fard, Jakub Tarnawski, and Volkan Cevher. Streaming robust submodular maximization: A partitioned thresholding approach. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 4560–4569, 2017.

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

Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, pages 3826–3835, 2018.

Ruben Sipos, Adith Swaminathan, Pannaga Shivaswamy, and Thorsten Joachims. Temporal corpus summarization using submodular word coverage. In *Proceedings of the 21st ACM international conference* *on Information and knowledge management*, pages 754–763. ACM, 2012.

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

Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. Using document summarization techniques for speech data subset selection. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 721–726, 2013.

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

Qilian Yu, Easton Li Xu, and Shuguang Cui. Streaming algorithms for news and scientific literature recommendation: Monotone submodular maximization with a d -knapsack constraint. *IEEE Access*, 6: 53736–53747, 2018.

A Standard greedy and thresholding inequalities

In this section we prove the standard greedy inequality $g(x) \ge 1 - e^{-x}$, where x is the cost of a partial greedy solution. To prove it, we first show that a differential inequality $g(x) + g'(x) \ge 1$ holds, and then integrate it using Proposition A.1. For the thresholding algorithm a similar approximate inequality holds.

Proposition A.1 *Let ξ be a continuous and piecewise smooth function $[u, v] \to \mathbb{R}^+$. If for some $\alpha, \beta > 0$ we have $\xi(x) + \alpha\xi'(x) \ge \beta$ for $u \le x \le v$, then $\xi(v) \ge \beta + (\xi(u) - \beta)e^{\frac{u-v}{\alpha}}$.*

Proof : First, consider the case when ξ is smooth. $\xi(x) + \alpha\xi'(x) \ge \beta$ implies that $\xi(x)e^{\frac{x}{\alpha}} + \alpha\xi'(x)e^{\frac{x}{\alpha}} \ge \beta e^{\frac{x}{\alpha}}$ through multiplication by $e^{\frac{x}{\alpha}}$. Observe that $\xi(x)e^{\frac{x}{\alpha}} + \alpha\xi'(x)e^{\frac{x}{\alpha}}$ is the derivative of $\xi(x)\alpha e^{\frac{x}{\alpha}}$. Hence, $\frac{d(\xi(x)\alpha e^{\frac{x}{\alpha}})}{dx} \ge \beta e^{\frac{x}{\alpha}}$ implies

$$\int_u^v d(\xi(x)\alpha e^{\frac{x}{\alpha}}) \ge \int_u^v \beta e^{\frac{x}{\alpha}} dx$$
$$(\xi(x)\alpha e^{\frac{x}{\alpha}})\Big|_u^v \ge \alpha\beta e^{\frac{x}{\alpha}}\Big|_u^v$$
$$\xi(v)\alpha e^{\frac{v}{\alpha}} - \xi(u)\alpha e^{\frac{u}{\alpha}} \ge \alpha\beta e^{\frac{v}{\alpha}} - \alpha\beta e^{\frac{u}{\alpha}}.$$

Dividing both sides by α,

$$\xi(v)e^{\frac{v}{\alpha}} - \xi(u)e^{\frac{u}{\alpha}} \ge \beta e^{\frac{v}{\alpha}} - \beta e^{\frac{u}{\alpha}}$$
$$\xi(v) \ge \beta + (\xi(u) - \beta)e^{\frac{u-v}{\alpha}}.$$

For a piecewise smooth ξ, let $u = x_0 < x_1 < \cdots < x_t = v$, such that ξ is smooth on a segment (x_i, x_{i+1}) for any i. By induction, we prove that the inequality holds for x_0, x_i for any i:

$$\xi(x_i) \ge \beta + (\xi(x_0) - \beta)e^{\frac{x_i - x_0}{\alpha}}.$$

The statement is true for $i = 0$. Induction step:

$$\begin{aligned}\xi(x_{i+1}) &\ge \beta + (\xi(x_i) - \beta)e^{\frac{x_i - x_{i+1}}{\alpha}} \\ &\ge \beta + (\xi(x_0) - \beta)e^{\frac{x_0 - x_i}{\alpha}} e^{\frac{x_i - x_{i+1}}{\alpha}} \\ &\ge \beta + (\xi(x_0) - \beta)e^{\frac{x_0 - x_{i+1}}{\alpha}}\end{aligned}$$

□

Theorem A.2 (Standard greedy inequality) *For all $x \in [0, 1 - c(o_1)]$, the greedy performance function g satisfies the following differential inequality:*

$$g(x) + g'(x) \ge 1,$$

and hence also its integral version: $g(x) \ge 1 - e^{-x}$.

Proof : Let $x \in [0, 1 - c(o_1)]$ and recall that by definition $\mathcal{G}_{i-1}$ is the largest set of elements selected by the greedy solution without exceeding total cost of x. Note that it suffices to show the inequality only for the left endpoints of the piecewise linear intervals of the form $[c(\mathcal{G}_{i-1}), c(\mathcal{G}_i))$ as inside these intervals g' stays constant while g can only increase and hence the inequality holds. Hence we can assume that $x = c(\mathcal{G}_{i-1})$ in the proof below which implies that $g(x) = f(\mathcal{G}_{i-1})$.

Since we normalized $f(\mathsf{OPT}) = 1$, by monotonicity:

$$\begin{aligned}1 &= f(\mathsf{OPT}) \le f(\mathsf{OPT} \cup \mathcal{G}_{i-1}) \\ &= f(\mathcal{G}_{i-1}) + f(\mathsf{OPT} \setminus \mathcal{G}_{i-1} \,|\, \mathcal{G}_{i-1}).\end{aligned}$$

Then by submodularity and using the fact that by definition $f(\mathcal{G}_{i-1}) = g(x)$:

$$\begin{aligned}1 &\le f(\mathcal{G}_{i-1}) + f(\mathsf{OPT} \setminus \mathcal{G}_{i-1} \,|\, \mathcal{G}_{i-1}) \\ &\le g(x) + \sum_{e \in \mathsf{OPT} \setminus \mathcal{G}_{i-1}} f(e \,|\, \mathcal{G}_{i-1}).\end{aligned}$$

Since $f(e \,|\, \mathcal{G}_{i-1}) = c(e)\rho(e \,|\, \mathcal{G}_{i-1})$:

$$\begin{aligned}1 &\le g(x) + \sum_{e \in \mathsf{OPT} \setminus \mathcal{G}_{i-1}} f(e \,|\, \mathcal{G}_{i-1}) \\ &= g(x) + \sum_{e \in \mathsf{OPT} \setminus \mathcal{G}_{i-1}} c(e)\rho(e \,|\, \mathcal{G}_{i-1}) \\ &\le g(x) + \sum_{e \in \mathsf{OPT} \setminus \mathcal{G}_{i-1}} c(e)g'(x),\end{aligned}$$

where the last inequality follows because greedy always picks the item with the largest marginal density and since $x \leq 1 - c(o_1)$ every item in $OPT \setminus \mathcal{G}_{i-1}$ can still fit into the knapsack. Hence,

$$\begin{aligned} 1 &\leq g(x) + g'(x) \sum_{e \in \mathsf{OPT} \setminus \mathcal{G}_{i-1}} c(e) \\ &= g(x) + g'(x) c(\mathsf{OPT} \setminus \mathcal{G}_{i-1}). \end{aligned}$$

The desired differential inequality follows from the observation that $c(\mathsf{OPT} \setminus \mathcal{G}_{i-1}) \leq c(\mathsf{OPT}) \leq 1$. Finally, by integrating from 0 to x using the initial condition $g(0) = 0$, it follows that $g(x) \geq 1 - e^{-x}$ (by Proposition A.1). □

Theorem A.3 (Standard thresholding inequality) *For all $x \in [0, 1 - c(o_1)]$, the thresholding performance function t satisfies the following differential inequality:*

$$t(x) + (1 + \epsilon) t'(x) \geq 1.$$

And hence also its integral version: $t(x) \geq 1 - e^{-\frac{x}{1+\epsilon}}$.

Proof : Let $p \in [0, 1]$ be the total cost of the elements collected by the thresholding algorithm in the first pass. First, note that for the first pass when $x \in [0, p]$ the differential inequality follows trivially as $t'(x) \geq \frac{\lambda}{\alpha K} \geq 1$ since $\lambda \geq \alpha f(\mathsf{OPT})$ and by our normalization $f(\mathsf{OPT}) = K = 1$. Fix $x \in [p, 1 - c(o_1)]$ and recall that by definition $\mathcal{T}_{i-1}$ is the largest set of elements selected by the thresholding algorithm without exceeding total cost of x. Similarly to the previous proofs it suffices to consider only the left endpoints of the intervals of the form $[c(\mathcal{T}_{i-1}, \mathcal{T}_i))$ so we assume $x = c(\mathcal{T}_{i-1})$. Since we normalized $f(\mathsf{OPT}) = 1$, then by monotonicity:

$$\begin{aligned} 1 = f(\mathsf{OPT}) &\leq f(\mathsf{OPT} \cup \mathcal{T}_{i-1}) \\ &= f(\mathcal{T}_{i-1}) + f(\mathsf{OPT} \setminus \mathcal{T}_{i-1} \mid \mathcal{T}_{i-1}). \end{aligned}$$

Then by submodularity and using the fact that by definition $t(x) = f(\mathcal{T}_{i-1})$:

$$\begin{aligned} 1 &\leq f(\mathcal{T}_{i-1}) + f(\mathsf{OPT} \setminus \mathcal{T}_{i-1} \mid \mathcal{T}_{i-1}) \\ &\leq t(x) + \sum_{e \in \mathsf{OPT} \setminus \mathcal{T}_{i-1}} f(e \mid \mathcal{T}_{i-1}). \end{aligned}$$

Since $f(e \mid \mathcal{T}_{i-1}) = c(e) \rho(e \mid \mathcal{T}_{i-1})$:

$$\begin{aligned} 1 &\leq t(x) + \sum_{e \in \mathsf{OPT} \setminus \mathcal{T}_{i-1}} f(e \mid \mathcal{T}_{i-1}) \\ &= t(x) + \sum_{e \in \mathsf{OPT} \setminus \mathcal{T}_{i-1}} c(e) \rho(e \mid \mathcal{T}_{i-1}) \\ &\leq t(x) + \sum_{e \in \mathsf{OPT} \setminus \mathcal{T}_{i-1}} c(e) t'(x)(1 + \epsilon), \end{aligned}$$

where the last inequality follows because after the first pass $t'(x) \geq \frac{\rho(e \mid \mathcal{T}_{i-1})}{1+\epsilon}$ for all $e \in \mathsf{OPT} \setminus \mathcal{T}_{i-1}$. Indeed, note that in all passes except the first one the thresholding algorithm always selects an item whose marginal density is at least $(1 + \epsilon)^{-1}$ times the best marginal density available. Since $t'(x)$ is the density of this item and all items in $\mathsf{OPT} \setminus \mathcal{T}_{i-1}$ still fit (as $x \leq 1 - c(o_1)$) we have $(1+\epsilon) t'(x) \geq \max_{e \in \mathsf{OPT} \setminus \mathcal{T}_{i-1}} \rho(e \mid \mathcal{T}_{i-1})$ as desired. Hence:

$$\begin{aligned} 1 &\leq t(x) + \sum_{e \in \mathsf{OPT} \setminus \mathcal{T}_{i-1}} c(e) t'(x)(1 + \epsilon) \\ &= t(x) + (1 + \epsilon) t'(x) c(\mathsf{OPT} \setminus \mathcal{T}_{i-1}). \end{aligned}$$

The desired differential inequality follows from the observation that $c(\mathsf{OPT} \setminus \mathcal{T}_{i-1}) \leq c(\mathsf{OPT}) = 1$.

For the integral version we integrate the differential inequality between 0 and x with the initial condition $t(0) = 0$ (formally, apply Proposition A.1 with $\alpha = 1 + \epsilon, \beta = 1, u = 0, v = x$) and get $t(x) \geq 1 - e^{-\frac{x}{1+\epsilon}}$, as desired. □

B Omitted proofs

Fact B.1 *For all $0 \leq x \leq 1$,*

$$(1 - x) e^{2x-1} \leq \frac{1}{2}.$$

Proof : Let $r(x) = (1 - x) e^{2x-1}$ and note that $r'(x) = (1 - 2x) e^{2x-1}$ so that $r'(x) > 0$ for $x \in \left[0, \frac{1}{2}\right]$ and $r'(x) \leq 0$ for $x \in \left[\frac{1}{2}, 1\right]$. Hence, it follows that $r\left(\frac{1}{2}\right) = \frac{1}{2}$ is a local maximum and so $(1 - x) e^{2x-1} \leq \frac{1}{2}$ for all $0 \leq x \leq 1$. □

Fact B.2

$$\left(1 - \frac{c(o_1)}{1 + \epsilon}\right) e^{\frac{2c(o_1)-1}{1+\epsilon}} \leq \frac{1}{2} + \epsilon.$$

Proof : By Fact B.1,

$$\left(1 - \frac{c(o_1)}{1 + \epsilon}\right) e^{\frac{2c(o_1)-1}{1+\epsilon}} \leq \frac{1}{2} e^{\frac{\epsilon}{1+\epsilon}}.$$

Hence it suffices to show that $e^{\frac{\epsilon}{1+\epsilon}} \leq 1 + 2\epsilon$, which follows from the fact that $\frac{d}{dx} e^{\frac{x}{1+x}} \leq 2$ for $0 \leq x \leq 1$. □

We now describe a generalization to a knapsack constraint of the algorithm of Kazemi et al. (2019) that computes a constant factor approximation to maximum submodular maximization under a cardinality constraint, using small space and a small number of queries.

Theorem B.3 *There exists a one-pass streaming algorithm that outputs a $\left(\frac{1}{3} - \epsilon\right)$-approximation to the submodular maximization under knapsack constraint that uses $O\left(\frac{K}{\epsilon}\right)$ space and $O\left(\frac{n \log K}{\epsilon}\right)$ total queries.*

Algorithm 4: Space efficient constant factor approximation

Input: Stream of elements $E = e_1, \ldots, e_n$, knapsack capacity K, cost function $c(\cdot)$, non-negative monotone submodular function f, and an approximation parameter $\epsilon > 0$;
Output: A set S that is a $\left(\frac{1}{3} - \epsilon\right)$-approximation for submodular maximization with a knapsack constraint;

```
τ_min, Δ, LB ← 0;
for each item e_i do
    if f(e_i) > Δ then
        e ← e_i, Δ ← f(e_i)
    τ_min = max(2LB, 2Δ) / (3K);
    Discard all sets with S_τ with τ < τ_min;
    for τ ∈ {(1+ε)^i | τ_min/(1+ε) ≤ (1+ε)^i ≤ Δ} do
        if τ is a new threshold then
            S_τ ← ∅
        if c(S_τ) < K and ρ(e | S_τ) ≥ τ then
            S_τ ← S_τ ∪ {e} and
            LB ← max{LB, f(S_τ)}
return argmax{f(S_τ), f(e)}
```

Proof : Since Algorithm 4 uses the same threshold as Algorithm 2 in Huang et al. (2017), it outputs a $\frac{1}{3} - \epsilon$-approximation. On the other hand, by Theorem 1 in Kazemi et al. (2019), Algorithm 4 uses space $O\left(\frac{K}{\epsilon}\right)$ and query complexity $O\left(\frac{n \log K}{\epsilon}\right)$. $\square$

Hence, by setting $\epsilon = \frac{1}{6}$, we obtain the following:

Corollary B.4 *There exists a one-pass streaming algorithm that outputs a $\frac{1}{6}$-approximation to the submodular maximization under knapsack constraint that uses $O(K)$ space and $O(n \log K)$ total queries.*