Abstract

Modern Challenges for Machine Learning Applications of Submodularity:

Privacy, Scalability, and Sequences

Marko Mitrovic

2020

In a nutshell, submodularity covers the class of all problems that exhibit some form of diminishing returns. From a theoretical perspective, this notion of diminishing returns is extremely useful as the resulting mathematical properties allow for provably efficient optimization. From a practical perspective, diminishing returns appear in a wide variety of important real-world machine learning problems including data summarization, recommender systems, neural network interpretability, and influence maximization in social networks.

In this thesis, we will focus on three major challenges for modern machine learning applications of submodularity: privacy, scalability, and sequences:

With the emergence of data privacy as one of the foremost controversies in today's society, it is imperative that each individual's personal information be protected. However, machine learning is a field that particularly relies on data. Without data, there is no learning. To tackle this challenge, we adapt the foundational algorithms of submodularity to the framework of differential privacy, which provides mathematically provable protection against information leaks.

On the opposite end of the spectrum of data challenges, we have the problem of too much data. As we will see, many submodular algorithms run in linear time, but with the immense size of modern datasets, even linear time may be too slow. To address this issue, we present novel approaches to scalable submodularity with a specific focus on streaming and distributed frameworks.

Lastly, the vast majority of work and research on submodularity has been focused on set functions where the order of the input and output is not important. While this is perfectly reasonable for many problems, there are also many other machine learning problems (such as recommender systems), where explicitly considering the order of data and solutions can lead to large gains. To this end, we present vital advancements in the field of sequence submodularity.

Modern Challenges for Machine Learning Applications of Submodularity: Privacy, Scalability, and Sequences

A Dissertation

Presented to the Faculty of the Graduate School

Of

Yale University

In Candidacy for the Degree of

Doctor of Philosophy

By

Marko Mitrovic

Dissertation Director: Amin Karbasi

May 2020

Contents

Za moju porodicu. Hvala vam za sve :)

Acknowledgements

I want to extend my deepest gratitude to my advisor, Amin Karbasi. I cannot thank you enough for everything you did for me throughout my time here at Yale. You always put in the extra effort to identify ideas that were exciting to the broader machine learning community, but also a fit for my specific interests. You always knew when to give me a push in the right direction, but I never felt like you placed undue pressure on me. You always encouraged me to pursue new challenges and to never stop learning. Even looking beyond research, these past five years have been some of the best years of my life and I know that a large part of that has been because you didn't just treat me as a student, but as a friend. So thank you for all the advice, all the papers, all the fully-funded conference trips to tropical places, and all the laughs. I won't forget it.

I also want to thank all my colloborators over the years: Andreas Krause, Mark Bun, Moran Feldman, Morteza Zadimoghaddam, and Silvio Lattanzi. None of my work would have been possible without your help. I want to extend a special thank you to Ehsan Kazemi, who has been my co-author on more than half of the papers I managed to publish. Your hard work and guidance truly pulled me through my Ph.D. and I am forever grateful.

I want to thank the rest of my committee members: Daniel Spielman, Dragomir Radev, and Yaron Singer. Thank you for time and your supervision. I know that there are a million other responsibilities on your plates and I am honoured that you all agreed to be on my committee. Most importantly, thank you for letting me pass my defense, I literally could not have done without your backing.

I am also grateful for my undergraduate research advisors: Luis Goddyn, Binay Bhattacharya, and Martin Ester. You took me under your wings when I was still just a teenager and guided me through my first experiences of real research. Without your mentorship, I doubt I would even have made it to graduate school, let alone this far.

Thank you to all the other professors and teachers and mentors and managers and everybody else that has taught me so much. I can't ever hope to pay you all back, but I hope I can at least pay it forward.

A small shout-out to myself, good job Marko.

Finally, the biggest thank you of all goes to my friends and family. I specifically want to mention my parents Snezana and Slobodan, my grandparents Dusan, Spomenka, Mladen, and Radosava, and my siblings Jelena and Dusan. Everything I am and everything I have achieved is a reflection of the love and support I have received from the closest people in my life. I appreciate all of you :)

1. Introduction

Machine learning has seen an enormous amount of success in recent years: from mastery of simple visual tasks such as object recognition (Krizhevsky et al., 2012; He et al., 2016) and image captioning (Vinyals et al., 2015; Xu et al., 2015) to super-human achievements in game-playing (Mnih et al., 2015; Silver et al., 2016; Vinyals et al., 2017). However, particularly with the proliferation of deep learning (LeCun et al., 2015; Goodfellow et al., 2016), there has generally been a lack of theoretical understanding behind many of the most popular approaches and frameworks.

This has led to a fierce debate in the machine learning community about the role of theory and interpretability in a world where black-box models have traditionally attained the highest performance. In stark contrast, **submodularity** (Fujishige, 2005; Bach, 2013) is one of the rare theoretically grounded disciplines that has achieved practical adoption in the machine learning world. In short, submodularity is a mathematical formalization of the intuitive notion of diminishing returns.

On the theory side, in addition to the fact that submodularity allows us to provide any theoretical guarantees at all, many submodular algorithms are proven to be very efficient, running in linear time or even faster. On the practical side, diminishing returns appear in a wide variety of important real-world problems, thus providing ample area for application of submodular algorithms and frameworks. Although it is also common in fields such as economics and operations research, submodularity has been particularly popular in the machine learning world with applications including data summarization (Mirzasoleiman et al., 2013; Lin and Bilmes, 2011; Kirchhoff and Bilmes, 2014), variable selection (Krause and Guestrin, 2005), sensor placement (Krause et al., 2008), recommender systems (Gabillon et al., 2013), crowd teaching (Singla et al., 2014), neural network interpretability (Elenberg et al., 2017), active learning (Golovin et al., 2010; Guillory and Bilmes, 2010), network inference (Gomez Rodriguez et al., 2010), and influence maximization in social networks (Kempe et al., 2003).

In this thesis, we will focus on three main challenges for modern machine learning applications of submodularity: *privacy*, *scalability*, and *sequences*:

- The topic of data privacy has emerged as one of the most contentious topics in today's society. On one hand, the simplest and most effective way to fully protect a user's privacy is to not collect or use any of their data. On the other hand, the explosion of available data in recent years has been one of the primary reasons for the success of many machine learning models. In section 3, we adapt the foundational algorithms of submodularity to the mathematical

framework of differential privacy (Mitrovic et al., 2017a). Intuitively speaking, an algorithm is said to be differentially private if adding or removing any one individual's data will not cause a significant change to the outcome of the algorithm. The idea is that this protects the privacy of all users by guaranteeing that an adversary cannot learn anything about any one individual's data.

- While the exponential growth in the amount of available data has certainly fueled a great deal of progress in the field of machine learning, we have also begun to encounter the problem of *too much* data. Although most submodular algorithms are based on linear-time greedy subroutines, the sheer amount of data present in today's datasets forces us to look for approaches that are even faster than linear time. In section 4, we present novel advancements towards scalable submodularity. In addition to presenting improved algorithms for both the single-stream and multi-stream settings (Kazemi et al., 2019), we also explore the two-stage submodular framework (Mitrovic et al., 2018b). The goal here is to use some given training functions to reduce the ground set so that optimizing new functions (drawn from the same distribution) over the reduced set provides almost as much value as optimizing them over the entire ground set.

- For many classical submodular problems (such as data summarization) the order of the input and the output is irrelevant. For example, if we want to select just two images to summarize New York City, we might select an image of the Statue of Liberty and an image of Times Square, but regardless of which image is first, our summary is still essentially the same. Conversely, for other problems commonly modelled using submodularity (such as recommender systems), the order of the items selected can be just as important as the items themselves. For example, if we are recommending movies, we might correctly predict that a user would enjoy the Lord of the Rings franchise. However, if we recommend the movies in an incorrect order, the user might end up completely confused and unsatisfied. In section 5, we present vital advancements to the field of sequence submodularity (Mitrovic et al., 2018a, 2019), where we develop algorithms that explicitly consider sequences instead of sets.

(a) (b)

Figure 1: Sensor placement example demonstrating (non-negative and monotone) submodularity. Adding a new sensor s^* to the set $A = \{s_1, s_2\}$ (shown in part a) is more valuable than adding it to the superset $B = \{s_1, s_2, s_3\}$ (shown).

2. Preliminaries

2.1 Definition of Submodularity

Intuitively, submodularity describes the set of functions that exhibit diminishing returns. Mathematically, a set function $f : 2^V \to \mathbb{R}$ is **submodular** if, for every two sets $A \subseteq B \subseteq V$ and element $v \in V \setminus B$, we have $f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B)$. That is, the marginal contribution of any element v to the value of $f(A)$ diminishes as the set A grows.

We will use a simple sensor placement task as an illustratory example (see Figure 1). The goal in this task is to place sensors to cover as much area as possible. Each sensor covers a predetermined area and there is no advantage to covering the same area twice, so overlapping sensors generally want to be avoided. This is where the notion of diminishing returns comes in for sensor placement.

For example, in Figure 1a, we have two existing sensors s_1 and s_2, and their area of coverage is indicated by the blue circles. We see that placing the new sensor s^* (whose area of coverage is indicated by the red circle) will have a little bit of overlap with the existing sensors, but it will mostly be covering new area, meaning it is quite valuable. On the other hand, in Figure 1b, we already have three existing sensors (s_1, s_2, and s_3), and we see that placing the same sensor s^* will result in much more overlap, thus meaning it is not covering as much new area and is therefore less valuable.

In mathematical terms, we first define $A = \{s_1, s_2\}$ to be the set of existing sensors in Figure 1a and $B = \{s_1, s_2, s_3\}$ to be the set of existing sensors in Figure 1b. If we use V to denote the set of all possible sensors, then it is clear that $A \subseteq B \subseteq V$. Next, let $f(X)$ indicate the total area

covered by a set of sensors X. Therefore, $f(A \cup \{s^*\}) - f(A)$ denotes the additional area covered by sensor s^* given that the sensors in set A have already been placed. This is commonly known as the marginal value or marginal gain of an item s^* to a set A. As a shorthand, we generally use $f(s^* \mid A) = f(A \cup \{s^*\}) - f(A)$ to denote the marginal gain of an element s^* to the set A.

As discussed above, adding the new sensor s^* is more valuable for set A than it is for set B. In other words, we have $f(s^* \mid A) \geq f(s^* \mid B)$. In fact, for any new sensor v and any two sets $A \subseteq B \subseteq V$, we will have $f(v \mid A) \geq f(v \mid B)$, which is exactly the definition of submodularity. This is because $A \subseteq B$ implies that set B contains all the sensors in set A (and possibly more). As a result, adding any new sensor v to set A will cover at least as much new area (and possibly more) as it would if it were added to set B.

2.2 Types of Submodularity

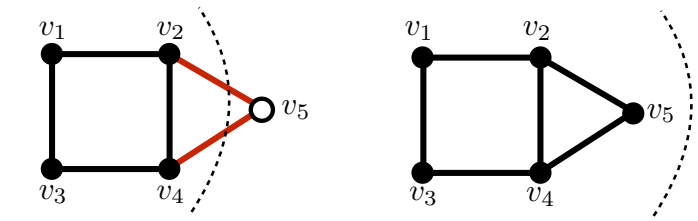
A submodular function f is said to be **monotone** if $f(A) \leq f(B)$ for any two sets $A \subseteq B \subseteq V$. That is, adding items to a set cannot decrease its value. Another way to think about this concept is that the marginal gain of any element is greater than or equal to zero. This also applies to our sensor placement example because adding a new sensor will certainly not decrease the total area of coverage.

On the other hand, if a submodular function f is **non-monotone**, then the marginal gain of an item can be negative. Although this might not sound particularly intuitive, an excellent introductory example is the maximum cut problem. Intuitively, the goal is to partition the vertices of a given graph into two groups such that the number of edges going between the two groups is maximized. In more mathematical terms, we are given a graph $G = (V, E)$ and the goal is to select a set $A \subseteq V$ that maximizes

$$f(A) = \sum_{u \in A, v \in V \setminus A} \mathbb{1}_{u,v}$$

where $\mathbb{1}_{u,v}$ is the indicator function for the existence of the edge (u, v). That is, $\mathbb{1}_{u,v} = 1$ if $(u, v) \in E$, and 0 otherwise.

Figure 2 gives an example of why the maximum cut problem is non-monotone. If we start off by selecting the set $A = \{v_1, v_2, v_3, v_4\}$, we get that $f(A) = 2$ because there are two edges going between the sets A and $V \setminus A$ (highlighted in red in Figure 2a). On the other hand (as shown in Figure 2b), if we add the last vertex v_5 to our set, we get the set $B = \{v_1, v_2, v_3, v_4, v_5\}$ and $f(B) = 0$ because



$A = \{v_1, v_2, v_3, v_4\}$
$f(A) = 2$

(a)

$B = \{v_1, v_2, v_3, v_4, v_5\}$
$f(B) = 0$

(b)

Figure 2: (a) and (b) demonstrate non-monotone submodularity because the set $A = \{v_1, v_2, v_3, v_4\}$ is less valuable than the set $B = \{v_1, v_2, v_3, v_4, v_5\}$. Another way to look at this is that the marginal gain of the vertex v_5 to the set A is negative.

$V \setminus B$ is the empty set and there cannot be any edges crossing into the empty set. In other words, the marginal gain of the item v_5 to the set $A = \{v_1, v_2, v_3, v_4\}$ is negative and thus the function is non-monotone.

In a related vein, a submodular function f is said to be **non-negative** if $f(A) \geq 0$ for any set A. Note that this is different from the concept of monotonicity. If a function is non-negative, there might still be items that have a negative marginal gain, it's just that the value of an entire set cannot be negative. For example, the maximum cut problem is non-negative because you cannot partition the vertices of a graph so that you have a negative number of edges going between the two groups. The sensor placement example is also clearly non-negative because we cannot have a set that covers a negative area.

2.3 Constraints on Submodular Functions

Another important piece of submodular optimization is the associated constraint. That is, if we are trying to choose a set A to optimize a function f, what kind of restrictions do we have for the selected set A? As a simple example, if f is monotone and we have no constraints on our set A, we could simply just select all possible items $A = V$ and trivially maximize f. As a result, the most common (and probably simplest) constraint is the **cardinality constraint**. With the cardinality constraint, we are simply limiting the size of the selected set A to some chosen k (i.e. $|A| \leq k$).

Another relatively common constraint in the literature of submodularity is the **matroid constraint** (Calinescu et al., 2011; Gharan and Vondrák, 2011). A matroid $\mathcal{M}$ on a ground set V can be thought of as a family of subsets of V. If a set $A \subseteq V$ is a part of this family of subsets, we say

that it is independent and we write $A \in \mathcal{M}$. Every matroid $\mathcal{M}$ must satisfy the following three properties:

1. $\emptyset \in \mathcal{M}$. That is, the empty set is independent.
2. If $B \in \mathcal{M}$ and $A \subseteq B$, then $A \in \mathcal{M}$. This is commonly known as the downward closed (or hereditary) property because it essentially says that if a set $B \subseteq V$ is independent, then all subsets $A \subseteq B$ are also independent.
3. If $A, B \in \mathcal{M}$ and $|A| \leq |B|$, then there exists an item $v \in B \setminus A$ such that $A \cup \{v\} \in \mathcal{M}$. This is commonly known as the augmentation property because it essentially says that if we look at two independent sets where one is larger than the other, then there is at least one item in the larger set that can be added to the smaller set such that the smaller set remains independent.

To better understand these constraints, consider the problem of movie recommendation, which is commonly modeled under some sort of framework involving diminishing returns. With a cardinality constraint, we are simply limited to recommending a maximum of k movies. This is similar to how Netflix, and most other recommender systems in the real world, will only show a certain number of suggestions.

The cardinality constraint is technically a type of matroid constraint, but matroid constraints can be much more general. One example of a matroid constraint in the context of movie recommendation would be limiting the number of movies from certain genre. To further illustrate why this qualifies as a matroid constraint, suppose we limit ourselves to recommending at most 4 drama movies and at most 5 comedy movies. We will go through the matroid properties one by one:

1. Not recommending any movies (i.e. the empty set) would clearly satisfy this constraint.
2. If $B \in \mathcal{M}$ then there are at most 4 drama movies and at most 5 comedy movies in B. Therefore, any subset $A \subseteq B$ will also have at most 4 drama movies and 5 comedy movies and thus $A \in \mathcal{M}$, satisfying the downward closed property.
3. For the augmentation property, we start off with sets $A, B \in \mathcal{M}$ and $A \subseteq B$ and we want to show that there exists a movie in B that can be added to A such that A is still an independent set. Here we can consider two simple cases:
   (a) If A has fewer comedy or drama movies than B, then we can simply choose one of the comedy or drama movies $v \in B$ and add them to A. Since $B \in M$ and B had more comedy or drama movies than A to start with, then $A \cup \{v\}$ will still be independent.

(b) If A and B have the same number of comedy and drama movies, then that means B has at least one movie v from an unrestricted genre, and thus $A \cup \{v\}$ will also remain independent.

2.4 The Greedy Algorithm and Submodularity

2.4.1 Classical Greedy Algorithm

Algorithm 1 Greedy Algorithm

1: **Input:** Ground set of items V, submodular function $f : 2^V \to \mathbb{R}$, cardinality constraint k
2: **Output:** Set $S \subseteq V$ of size k
3: Initialize $S = \emptyset$
4: **for** $i = 1, \ldots, k$ **do**
5: $\quad v_i = \arg\max_v f(v \mid S)$ ▷ Find the item with the highest marginal gain
6: $\quad S = S \cup \{v_i\}$ ▷ Add that item to S
7: Return S

The **greedy** algorithm (Algorithm 1) is an intuitive iterative algorithm that simply selects the best option given the current information. In the context of discrete optimization (and submodularity more specifically), this means that the greedy algorithm simply selects the item with the highest marginal gain.

A seminal result in submodularity states that if our utility function f is non-negative, monotone and submodular, then the classical greedy algorithm maximizes f subject to a cardinality constraint up to an approximation ratio of $1 - 1/e$ (Nemhauser et al., 1978) (see Appendix A.1 for a proof).

In addition to this strong theoretical guarantee on the utility, the greedy algorithm is also desirable for its computational efficiency. In particular, if we want to select k items out of a ground set of size n, then the traditional greedy algorithm requires just $O(nk)$ function evaluations, making it linear in the size of the ground set.

2.4.2 Lazy Greedy Algorithm

One heuristic for improving the greedy algorithm is known as the **lazy greedy** algorithm (Minoux, 1978). This heuristic works by keeping an upper bound $\beta(v)$ on the marginal gain of each item, initialized to marginal gain of that item relative to the empty set, i.e. $\beta(v) = f(v \mid \emptyset) = f(v)$.

At the beginning of each iteration of the lazy greedy algorithm, we have a list of all available items sorted according to these upper bounds. We then take the item v_1 with the current highest upper

bound and update the upper bound to reflect the marginal gain relative to the current set of selected items S. That is, we set $\beta(v_1) = f(v_1 \mid S)$

Notice that, due to submodularity, the marginal gain of any item cannot increase. Therefore, if the updated upper bound of v_1 is still larger than the upper bound of v_2 (the next item in our sorted list) then we certainly know that v_1 will have the highest marginal gain and there is no reason to evaluate $f(v_i \mid S)$ for all the other items. In case we find that $\beta(v_1) < \beta(v_2)$ after the update, we simply insert v_1 into the proper spot to maintain the sorted order of our list and repeat the process with v_2 as our new highest upper bound.

The lazy greedy algorithm does not provide any theoretical improvements because in the worst case it is perfectly possible that we need to update the upper bounds $\beta(v_i)$ for all items v_i in every single iteration. However, in practice we do not usually see these worst case-scenarios and the lazy greedy algorithm has been shown to result in runtimes that are orders of magnitude faster than the classical greedy algorithm.

2.4.3 Stochastic Greedy Algorithm

The **stochastic greedy algorithm** (Mirzasoleiman et al., 2015) combines the best of both worlds and produces an improved greedy algorithm with both provable theoretical guarantees and strong practical performance. The main idea of this approach is to use subsampling. That is, instead of re-evaluating the marginal gain of all items in every iteration like the classical greedy algorithm and (in the worst case) the lazy greedy algorithm, the stochastic greedy algorithm simply randomly looks at s items in every iteration and selects the one that has the highest marginal gain.

Perhaps surprisingly, this algorithm guarantees a minimum utility that is independent of n (the total size of the ground set) and k (the total number of items we are selecting) and instead depends only on s (the number of items subsampled in each iteration). In particular, if we set $s = \frac{n}{k} \log \frac{1}{\epsilon}$ (where ϵ is any arbitrarily small constant) then the solution output by the stochastic greedy algorithm is guaranteed to be within a factor of $(1 - \frac{1}{e} - \epsilon)$ of the true optimal solution, while requiring only $O(n \log \frac{1}{\epsilon})$ function evaluations.

Compare this to the classical greedy algorithm, which guarantees that the output is within a factor of $(1 - \frac{1}{e})$ of the true optimal solution, but it requires $O(nk)$ function evaluations. It is clear that by changing the value of ϵ (which is a free hyperparameter), one can trade-off algorithmic runtime against utility guarantees.

Other research on stochastic submodular maximization include works from Karimi et al. (2017), Hassani et al. (2017), Mokhtari et al. (2018), and Hassidim and Singer (2017).

3. Submodularity and Differential Privacy

3.1 Introduction to Differentially Private Submodular Maximization

Many of the most compelling use cases of submodularity (such as data summarization, recommender systems, and feature selection) commonly concern sensitive data about individuals. As a simple running example, let us consider the specific problem of determining which of a massive number of features (e.g. age, height, weight, etc.) are most relevant to a binary classification task (e.g. predicting whether an individual is likely to have diabetes). In this problem, a sensitive training set takes the form $D = \{(x_i, y_i)\}_{i=1}^n$ where each individual i's data consists of a features $x_{i,1}, \ldots, x_{i,m}$ together with a class label y_i. The goal is to identify a small subset $S \subseteq [m]$ of features which can then be used to build a good classifier for y. Many techniques exist for feature selection, including one based on maximizing a submodular function which captures the mutual information between a subset of features and the class label of interest (Krause and Guestrin, 2005). However, for both legal (e.g. compliance with HIPAA regulations) and ethical reasons, it is important that the selection of relevant features does not compromise the privacy of any individual who has contributed to the training data set. Unfortunately, the theory of submodular maximization does not in itself accommodate such privacy concerns.

To this end, we propose a systematic study of *differentially private submodular maximization* to enable these applications based on submodular maximization, while provably guaranteeing individual-level privacy. The definition of differential privacy Dwork et al. (2006), which emerged from the theoretical computer science literature, offers a strong protection of individual-level privacy. Nevertheless, differential privacy has been shown to permit useful data analysis and machine learning tasks. In a nutshell, the definition formalizes a guarantee that no individual's data should have too significant an effect on the outcome of a computation. We provide the formal definition in Section 3.4.

The problem of differentially private submodular maximization can be summarized as follows: Given a sensitive dataset D associated to a submodular function $f_D : 2^V \to \mathbb{R}$: Find a subset $S \in \mathcal{C} \subseteq 2^V$ that approximately maximizes $f_D(S)$ in a manner that guarantees differential privacy with respect

to the input dataset D. In this chapter, we study this problem under various conditions on the submodular objective function f (monotone vs. non-monotone), and various choices of the constraint $\mathcal{C}$ (cardinality, matroid, or p-system).

An important special case of this problem was studied in prior work of Gupta et al. (2010). They considered the "combinatorial public projects" problem (Papadimitriou et al., 2008), where given a dataset $D = (x_1, \ldots, x_n)$, the function f_D takes the particular form $f_D(S) = \frac{1}{n}\sum_{i=1}^{n} f_{x_i}(S)$ for monotone submodular functions $f_{x_i} : 2^V \to [0, 1]$, and is to be maximized subject to a cardinality constraint $|S| \leq k$. We call functions of this form *decomposable*. They presented a simple greedy algorithm, which will be central to our work, together with a tailored analysis which achieves strong accuracy guarantees in this special case.

However, there are many cases which do not fall into the combinatorial public projects framework. For some problems, including feature selection via mutual information, the submodular function f_D of interest depends on the dataset D in ways much more complicated than averaging functions associated to each individual. The focus of our work is to capture a broader class of useful applications in machine learning. We summarize our specific contributions in Section 3.3.

3.2 The Greedy Paradigm

Even without concern for privacy, the problem of submodular maximization poses computational challenges. In particular, exact submodular maximization subject to a cardinality constraint is **NP**-hard. One of the principle approaches to designing efficient approximation algorithms is to use a greedy strategy (Nemhauser et al., 1978). Consider the problem of maximizing a set function $f(S)$ subject to the cardinality constraint $|S| \leq k$. In each of rounds $i = 1, \ldots, k$, the basic greedy algorithm constructs S_i from S_{i-1} by adding the element $v_i \in (V \setminus S_{i-1})$ which maximizes the marginal gain $f(S_{i-1} \cup \{v_i\}) - f(S_{i-1})$. Nemhauser et al. (1978) famously showed that this algorithm yields a $(1 - 1/e)$-approximation to the optimal value of $f(S)$ whenever f is a monotone submodular function.

In the combinatorial public projects setting, Gupta et al. (2010) showed how to make the greedy algorithm compatible with differential privacy by randomizing the procedure for selecting each v_i. This selection procedure is specified by the differentially private *exponential mechanism* of McSherry and Talwar (2007), which (probabilistically) guarantees that the v_i selected in each round is almost as good as the true marginal gain maximizer. Remarkably, Gupta et al. (2010) show that the cumulative

	Cardinality	Matroid	p-System
Com. Pub. Pro.	$\left(1-\frac{1}{e}\right)OPT - O\left(\frac{k\log\lvert V\rvert}{n}\right)$	$\frac{1}{2}OPT - O\left(\frac{k\log\lvert V\rvert}{n}\right)$	$\frac{1}{p+1}OPT - O\left(\frac{k\log\lvert V\rvert}{n}\right)$
Monotone	$\left(1-\frac{1}{e}\right)OPT - O\left(\frac{k^{3/2}\log\lvert V\rvert}{n}\right)$	$\frac{1}{2}OPT - O\left(\frac{k^{3/2}\log\lvert V\rvert}{n}\right)$	$\frac{1}{p+1}OPT - O\left(\frac{k^{3/2}\log\lvert V\rvert}{n}\right)$
Non-monotone	$\frac{1}{e}\left(1-\frac{1}{e}\right)OPT - O\left(\frac{k^{3/2}\log\lvert V\rvert}{n}\right)$	–	–

Table 1: Guarantees of expected solution quality for privately maximizing a sensitivity-$(1/n)$ submodular function f_D. The parameter k represents either a cardinality constraint, or the size of the set returned (for matroid or p-system constraints). Full expressions with explicit dependencies on differential privacy parameters ε, δ appear in this section. The cardinality result for the Combinatorial Public Projects problem is due to Gupta et al. (2010), but the rest are our novel contributions.

privacy guarantee of the resulting randomized greedy algorithm is not much worse than that of a single run of the exponential mechanism. This analysis is highly tailored to the structure of the combinatorial public projects problem. However, it is not hard to see that by replacing this tailored analysis with the more generic "advanced composition theorem" for differential privacy Dwork et al. (2010), one still obtains useful results for the more general class of "low-sensitivity" submodular functions.

3.3 Our Contributions

Table 1 summarizes the approximation guarantees we obtain under increasingly more general classes of submodular functions f_D (read top to bottom), and increasingly more general types of constraints (read left to right). In each entry, OPT denotes the value of the optimal non-private solution. Below we draw attention to a few particular contributions, including some that are not expressed in Table 1.

Non-monotone objective functions. Submodular maximization for non-monotone functions is significantly more challenging than it is for monotone objectives. In particular, the basic greedy algorithm of Nemahauser et al. fails dramatically, and cannot guarantee any constant-factor approximation. Several works (Buchbinder et al., 2014; Feldman et al., 2011) have identified variations of the greedy algorithm that do yield constant-factor approximations for non-monotone objectives. However, it is not clear how to modify any of these algorithms to accommodate differential privacy.

Our starting point is instead the "stochastic greedy" algorithm of Mirzasoleiman et al. (2015), which was originally designed to perform *monotone* submodular maximization in nearly linear time. Drawing ideas from Buchbinder et al. (2014), we give a new analysis of the stochastic greedy algorithm

to show that it also gives a $\frac{1}{e}(1-1/e)$-approximation for non-monotone submodular functions. To our knowledge, this is the first algorithm running in time exactly $|V|$ which achieves any constant-factor approximation for non-monotone objectives. Moreover, it is immediately clear how to use the exponential mechanism to make this algorithm differentially private.

This phenomenon is quite analogous to how stochastic variants of gradient descent are more amenable to providing differential privacy than their deterministic counterparts (Bassily et al., 2014). That is, our results illustrate how techniques for making algorithms *fast* are also helpful in making them *privacy-preserving*.

General constraints. While a cardinality constraint is perhaps the most natural to place on a submodular maximization problem, some machine learning problems require the use of more general types of constraints such as personalized data summarization Mirzasoleiman et al. (2016a). For instance, one may wish to maximize a submodular function $f(S)$ subject to $S \in \mathcal{I}$ for an arbitrary matroid $\mathcal{I}$, or subject to S being contained in an intersection of p matroids (more generally, a p-extenible system). For these types of constraints, the greedy algorithm still yields a constant factor approximation for monotone objective functions Fisher et al. (1978); Jenkyns (1976); Calinescu et al. (2011). We show in this work that the analysis in Calinescu et al. (2011) for matroids and p-systems can be adapted to handle additional error introduced for differential privacy.

General selection procedures. For worst-case datasets, the exponential mechanism is optimal within each round of private maximization. However, it may be sub-optimal for datasets enjoying additional structural properties. Fortunately, the greedy framework we use is flexible with regard to the choice of the selection procedure. For instance, one can replace the exponential mechanism in a black-box manner with the "large margin mechanism" of Chaudhuri et al. (2014) to obtain error bounds that replace the explicit dependence on $\log |V|$ in Table 1 with a term that may be significantly smaller for real datasets. To do so, we came up with a simplified version and tailored analysis of large margin mechanism suitable for greedy algorithms that used the same data set multiple times. For submodular functions exhibiting additional structure, one may also be able to perform each maximization step with the "choosing mechanism" of Bun et al. (2015) and Beimel et al. (2016).

3.4 Differential Privacy Definitions

Let V be finite set which we will refer to as a "ground set," and let X be a finite set which we will refer to as a "data universe". A dataset is an n-tuple $D = (x_1, \ldots, x_n) \in X^n$. Suppose each dataset D is associated to a set function $f_D : 2^V \to \mathbb{R}$. The manner in which f_D depends on D will be application-specific, but it is assumed that the association between D and f_D is public information.

We are interested in the problem of approximately maximizing a submodular function subject to differential privacy. The definition of differential privacy relies on the notion of *neighboring* datasets, which are simply tuples $D, D' \in X^n$ that differ in at most one entry. If D, D' are neighboring, we write $D \sim D'$.

Definition 3.1. *A randomized algorithm $M : X^n \to \mathcal{R}$ satisfies* (ε, δ)-differential privacy *if for all measurable sets $T \subseteq \mathcal{R}$ and all neighboring datasets $D \sim D'$.*

$$\Pr[M(D) \in T] \leq e^{\varepsilon} \Pr[M(D') \in T] + \delta.$$

Differentially private algorithms must be calibrated to the sensitivity of the function of interest with respect to small changes in the input dataset, defined formally as follows.

Definition 3.2. *The* sensitivity *of a set function $f_D : 2^V \to \mathbb{R}$ (depending on a dataset D) is defined as*

$$\max_{D \sim D'} \max_{S \subseteq V} |f_D(S) - f_{D'}(S)|.$$

Composition of Differential Privacy. The analyses of our algorithms rely crucially on *composition theorems* for differential privacy. For a sequence of privacy parameters $\{(\varepsilon_i, \delta_i)\}_{i=1}^k$, we informally refer to the *k-fold adaptive composition* of $(\varepsilon_i, \delta_i)$-differentially private algorithms as the output of a mechanism M^* that behaves as follows on an input D: In each of rounds $i = 1, \ldots, k$, the algorithm M^* selects an $(\varepsilon_i, \delta_i)$-differentially private algorithm M_i possibly depending on the previous outcomes $M_1(D), \ldots, M_i(D)$ (but *not* directly on the sensitive dataset D itself), and releases $M_i(D)$. For a formal treatment of adaptive composition, see Dwork et al. (2010); Dwork and Roth (2014).

Theorem 3.1. *(Dwork and Lei, 2009; Dwork et al., 2010; Bun and Steinke, 2016) The k-fold adaptive composition of $(\varepsilon_0, \delta_0)$-differentially private algorithms satisfies (ε, δ)-differential privacy where*

1. *$\varepsilon = k\varepsilon_0$ and $\delta = k\delta_0$. (Basic Composition).*
2. *$\varepsilon = \frac{1}{2}k\varepsilon_0^2 + \sqrt{2\log(1/\delta')}\varepsilon_0$ and $\delta = \delta' + k\delta$, for any $\delta' > 0$. (Advanced Composition)*

Exponential Mechanism. The exponential mechanism McSherry and Talwar (2007) is a general purpose primitive for solving discrete optimization functions. Let $q : V \times X^n \to \mathbb{R}$ be a "quality" function measuring how good a solution $v \in V$ is with respect to a dataset $D \in X^n$. We say a quality function q has *sensitivity* λ if for all $v \in V$ and all neighboring datasets $D \sim D'$, we have $|q(v, D) - q(v, D')| \leq \lambda$.

Proposition 3.2. *McSherry and Talwar (2007) Let $\varepsilon > 0$ and let $q : V \times X^n$ be a quality function with sensitivity λ. Define the* exponential mechanism *as the algorithm which selects every $v \in V$ with probability proportional to $\exp(\varepsilon q(v, D)/2\lambda)$.*

- *The exponential mechanism provides $(\varepsilon, 0)$-differential privacy.*
- *For every $D \in X^n$, let $\mathrm{OPT} = \arg\max_{v \in V} q(v, D)$. Then*

$$\mathbb{E}[q(\hat{v}, D)] \geq \mathrm{OPT} - \frac{2\lambda \cdot \log |V|}{\varepsilon},$$

where $\hat{v}$ is the output of the exponential mechanism on dataset D.

The privacy guarantee and a "with high probability" utility guarantee of the exponential mechanism are due to McSherry and Talwar McSherry and Talwar (2007). A simple proof of the utility guarantee in expectation appears in Bassily et al. (2016).

Large Margin Mechanism The accuracy guarantee of the exponential mechanism can be pessimistic on datasets where $q(\cdot, D)$ exhibits additional structure. For example, suppose that when the elements of V are sorted so that $q(v_1, D) \geq q(v_2, D) \geq \cdots \geq q(v_{|V|}, D)$, there exists an ℓ such that $q(v_1, D) \gg q(v_{\ell+1}, D)$. Then only the top ℓ ground set items are relevant to the optimization problem, so running the exponential mechanism on these should maintain differential privacy, but with error proportional to $\log \ell$ rather than to $\log |V|$. The *large margin mechanism* of Chaudhuri et al. (2014), like the exponential mechanism, generically solves discrete optimization problems. However,

it automatically leverages this additional margin structure whenever it exists. Asymptotically, the error guarantee of the large margin mechanism is always at most that of the exponential mechanism, but can be much smaller when the data exhibits a margin for small ℓ. Further details about the large margin mechanism are given in Appendix B.1.

3.5 Monotone Submodular Maximization with Differential Privacy

In this section, we present a variant of the basic greedy algorithm which will enable maximization of monotone submodular functions. This algorithm simply replaces each greedy selection step with a privacy-preserving selection algorithm denoted $\mathcal{O}$. The selection function $\mathcal{O}$ takes as input a quality function $q : U \times X^n \to \mathbb{R}$ and a dataset D, as well as privacy parameters ε_0, δ_0, and outputs an element $u \in U$. We begin in the simplest case of monotone submodular maximization with a cardinality constraint (Algorithm 2). The algorithm for more general constraints appears in Section 3.5.1.

Algorithm 2 was already studied by Gupta et al. (2010) in the special case where f_D is decomposable, and $\mathcal{O}$ is the exponential mechanism. We generalize their result to the much broader class of low-sensitivity monotone submodular functions.

Algorithm 2 Diff. Private Greedy (Cardinality) $\mathcal{G}^{\mathcal{O}}$

Input: Submodular function $f_D : 2^V \to \mathbb{R}$, dataset D, cardinality constraint k, privacy parameters ε_0, δ_0

Output: Size k subset of V

1. Initialize $S_0 = \emptyset$
2. For $i = 1, \ldots, k$:
   - Define $q_i : (V \setminus S_{i-1}) \times X^n \to \mathbb{R}$ via $q_i(v, \tilde{D}) = f_{\tilde{D}}(S_{i-1} \cup \{v\}) - f_{\tilde{D}}(S_{i-1})$
   - Compute $v_i \leftarrow_{\text{R}} \mathcal{O}(q_i, D; \varepsilon_0, \delta_0)$
   - Update $S_i \leftarrow (S_{i-1} \cup \{v_i\})$
3. Return S_k

Theorem 3.3. *Suppose $f_D : 2^V \to \mathbb{R}$ is monotone and has sensitivity λ. Then instantiating Algorithm 2 with $\mathcal{O} = \text{EM}$ (the exponential mechanism) and parameter $\varepsilon_0 > 0$ provides $(\varepsilon = k\varepsilon_0, \delta = 0)$-differential privacy. It also provides (ε, δ)-differential privacy for every $\delta > 0$ with $\varepsilon = k\varepsilon_0^2/2 + \varepsilon_0 \cdot \sqrt{2k \ln(1/\delta)}$.*

Moreover, for every $D \in X^n$,

$$\mathbb{E}\left[f_D(S_k)\right] \geq \left(1 - \frac{1}{e}\right) \text{OPT} - \frac{2\lambda k \ln |V|}{\varepsilon_0}$$

where $S_k \leftarrow_{\text{R}} \mathcal{G}^{\text{EM}}(D)$.

See Appendix B.2.1 for a full proof.

3.5.1 Matroid and p-Extendible System Constraints

We now show how to extend Algorithm 2 to privately maximize monotone submodular functions subject to more general constraints. To start, we review the definition of a p-extendible system. Consider a ground set V and a non-empty downward-closed family of subsets $\mathcal{I} \subseteq 2^V$ (i.e. if $T \in \mathcal{I}$, then $S \in \mathcal{I}$ for every $S \subseteq T$). Such an $\mathcal{I}$ is called a family of *independent sets*. The pair $(V, \mathcal{I})$ is said to be a *p-extendible system* (Mestre, 2006) if for all $S \subset T \in \mathcal{I}$, and $v \in V$ such that $S \cup \{v\} \in \mathcal{I}$, there exists a set $Z \subseteq (T \setminus S)$ such that $|Z| \leq p$ and $(T \setminus Z) \cup \{v\} \in \mathcal{I}$. Let $r(\mathcal{I})$ denote the size of the largest independent set in $\mathcal{I}$.

The definition of a *matroid* coincides with that of a 1-extendible system (with *rank* $r(\mathcal{I})$). For $p \geq 2$, the notion of a p-extendible system strictly generalizes that of an intersection of p matroids. A slight modification of Algorithm 2 gives a unified algorithm for privately maximizing a monotone submodular function subject to matroid and p-extendible system constraints, presented as Algorithm 3.

We obtain analogues of the algorithms and results presented for the cardinality constraints. For submodular functions with p-extentible system constraints, our algorithm will be similar to the previous algorithms where, in each iteration, we greedily add an element using some privacy-preserving selection mechanism. However, instead of running for a set number of iterations k, we will run until our set S_i is maximal (i.e. we cannot add another element without breaking the p-extendible system constraint) and we use k to denote the size of the set when the algorithm terminates.

Algorithm 3 Differentially Private Greedy (p-system) $\mathcal{G}^{\mathcal{O}}$

Input: Submodular function $f_D : 2^V \to \mathbb{R}$, dataset D, p-extendible family $(V, \mathcal{I})$, privacy parameters ε_0, δ_0
Output: Maximal independent subset of V

1. Initialize $S = \emptyset$
2. While $S \in \mathcal{I}$ is not maximal:
   - Define $q : (V \setminus S) \times X^n \to \mathbb{R}$ via $q(v, \tilde{D}) = f_{\tilde{D}}(S \cup \{v\}) - f_{\tilde{D}}(S)$
   - Compute $v_i \leftarrow_{\mathrm{R}} \mathcal{O}(q, D; \varepsilon_0, \delta_0)$
   - Update $S \leftarrow (S \cup \{v_i\})$
3. Return S

Theorem 3.4. *Suppose $f_D : 2^V \to \mathbb{R}$ has sensitivity λ. Then instantiating Algorithm 3 with $\mathcal{O} = \mathrm{EM}$ and parameter $\varepsilon_0 > 0$ provides $(\varepsilon = r(\mathcal{I})\varepsilon_0, \delta = 0)$-differential privacy. It also provides*

(ε, δ)-differential privacy for every $\delta > 0$ with $\varepsilon = r(\mathcal{I})\varepsilon^2/2 + \varepsilon \cdot \sqrt{2r(\mathcal{I})\ln(1/\delta)}$.

Moreover, for every $D \in X^n$,

$$\mathbb{E}\left[f_D(S)\right] \geq \frac{1}{p+1} \cdot \mathrm{OPT} - \frac{p}{p+1}\left(\frac{2\lambda r(\mathcal{I}) \ln |V|}{\varepsilon_0}\right)$$

where $S \leftarrow_R \mathcal{G}^{\mathrm{EM}}(D)$.

Please see Appendix B.2.2 for a full proof.

3.6 Non-Monotone Submodular Maximization with Differential Privacy

We now consider the problem of privately maximizing an arbitrary, possibly non-monotone, submodular function under a cardinality constraint. In general, the greedy algorithm presented in Section 3.5 fails to give any constant-factor approximation. Instead, our algorithm in this section will be based on the "stochastic greedy" algorithm first studied by Mirzasoleiman et al. (2015). In each round, the stochastic greedy algorithm first subsamples a random $\frac{1}{k} \cdot \ln(1/\alpha)$ fraction of the ground set for some $\alpha > 0$, and then greedily selects the item from this subsample that maximizes marginal gain. Mirzasoleiman et al. (2015) showed that for a monotone objective function f, this algorithm provides a $(1 - 1/e - \alpha)$-approximation to the optimal solution. Their original motivation was to improve the running time of the greedy algorithm: from $O(|V| \cdot k)$ evaluations of the objective function to linear $O(|V| \cdot \ln(1/\alpha))$.

Unfortunately, the stochastic greedy algorithm does not provide any approximation guarantee for non-monotone submodular functions. Buchbinder et al. (2014) instead proposed a "random greedy" algorithm that, in each iteration, randomly selects one of the k elements with the highest marginal gain. Buchbinder et al. (2014) showed that the random greedy algorithm achieves a $1/e$ approximation to the optimal solution (in expectation), using $k|V|$ function evaluations. However, it is not clear how to adapt this algorithm to accommodate differential privacy, since its analysis has a brittle dependence on the sampling procedure.

We make two main contributions to the analysis of the stochastic greedy and random greedy algorithms. First, we show that running the stochastic greedy algorithm on an exact $\frac{1}{k}$ fraction of the ground set per iteration still gives a (0.468)-approximation for monotone objectives, and moreover, gives a $\frac{1}{e}(1 - 1/e)$-approximation even for non-monotone objectives. Note that this algorithm evaluates the objective function on only $|V|$ elements, and still provides a constant factor approximation

guarantee. This makes our "subsample-greedy" algorithm the fastest algorithm for maximizing a general submodular function subject to a cardinality constraint (albeit with slightly worse approximation guarantees). Second, we show that the guarantees of this algorithm are robust to using a randomized greedy selection procedure (e.g. the exponential or large margin mechanism), and hence it can be adapted to ensure differential privacy.

We present the subsample-greedy algorithm as Algorithm 4 below. Assume that V is augmented by enough "dummy elements" to ensure that $|V|/k$ is an integer; each dummy element u is defined so that $f_D(S \cup \{u\}) = f_D(S)$ for every set S. We also explicitly account for an additional set U of k dummy elements, and ensure that at least one appears in every subsample.

Algorithm 4 Diff. Private "Subsample-Greedy" $\mathcal{SG}^{\mathcal{O}}$

Input: Submodular function $f_D : 2^V \to \mathbb{R}$, dataset D, cardinality constraint k, privacy parameters ε_0, δ_0

Output: Size k subset of V

1. Initialize $S_0 = \emptyset$, dummy elements $U = \{u^1, \ldots, u^k\}$
2. For $i = 1, \ldots, k$:
   - Sample $V_i \subset V$ a uniformly random subset of size $|V|/k$ and u_i a random dummy element
   - Define $q_i : (V_i \cup \{u_i\}) \times X^n \to \mathbb{R}$ via $q_i(v, \tilde{D}) = f_{\tilde{D}}(S_{i-1} \cup \{v\}) - f_{\tilde{D}}(S_{i-1})$
   - Compute $v_i \leftarrow_{\text{R}} \mathcal{O}(q_i, D; \varepsilon_0, \delta_0)$
   - Update $S_i \leftarrow (S_{i-1} \cup \{v_i\})$
3. Return S_k with all dummy elements removed

Theorem 3.5. *Suppose $f_D : 2^V \to \mathbb{R}$ has sensitivity λ. Then instantiating Algorithm 4 with $\mathcal{O} = \text{EM}$ provides (ε, δ)-differential privacy, and for every $D \in X^n$,*

$$\mathbb{E}\left[f_D(S)\right] \geq \frac{1}{e}\left(1 - \frac{1}{e}\right)\text{OPT} - \frac{2\lambda k \ln |V|}{\varepsilon}$$

where $S \leftarrow_R \mathcal{SG}^{\text{EM}}(D)$. Moreover, if f_D is monotone, then

$$\begin{aligned}\mathbb{E}\left[f_D(S)\right] &\geq \left(1 - e^{-(1-1/e)}\right)\text{OPT} - \frac{2\lambda k \ln |V|}{\varepsilon} \\ &\approx 0.468\,\text{OPT} - \frac{2\lambda k \ln |V|}{\varepsilon}.\end{aligned}$$

Please see Appendix B.2.3 for a full proof of Theorem 3.5.

The guarantees of Theorem 3.5 are of interest even without privacy. Letting MAX denote the selection procedure which simply outputs the true maximizer (equivalently, which runs the exponential mechanism with $\varepsilon_0 = +\infty$), we obtain the following non-private algorithm for maximizing a submodular function f_D:

Corollary 3.6. *Let $f_D : 2^V \to \mathbb{R}$ be any submodular function. Instantiating Algorithm 4 with $\mathcal{O} = \mathrm{MAX}$ gives*

$$\mathbb{E}\left[f_D(S)\right] \geq \frac{1}{e}\left(1 - \frac{1}{e}\right)\mathrm{OPT}$$

where $S \leftarrow_R \mathcal{SG}^{\mathrm{MAX}}(D)$. Moreover, if f_D is monotone, then

$$\mathbb{E}\left[f_D(S)\right] \geq \left(1 - e^{-(1-1/e)}\right)\mathrm{OPT} \approx 0.468\,\mathrm{OPT}\,.$$

3.7 Applications of Submodular Maximization with Differential Privacy

In this section we describe two concrete applications of our mechanisms.

3.7.1 Location Privacy

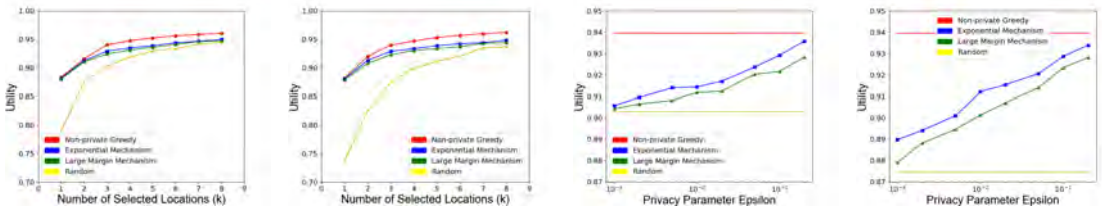
We analyze a dataset of 10,000 Uber pickups in Manhattan in April 2014 (UberDataset). Each individual entry in the dataset consists of the longitude and latitude coordinates of the pickup location. We want to use this dataset to select k public locations as waiting spots for idle Uber drivers, while also guaranteeing differential privacy for the passengers whose locations appear in this dataset.[1] We consider two different public sets of locations L:

- $L_{Popular}$ is a set of 33 popular locations in Manhattan.
- L_{Grid} is a set of 33 locations spread evenly across Manhattan in a grid-like manner.

We define a utility function $M(i,j)$ to be the normalized Manhattan distance between a pickup location i and the waiting location j. That is, if pickup location i is located at coordinates (i_1, i_2) and the waiting location j is located at coordinates (j_1, j_2), then $M(i,j) = \dfrac{|i_1 - j_1| + |i_2 - j_2|}{m}$, where $m = 0.266$ is simply the Manhattan distance between the two furthest spread apart points in Manhattan. This normalization ensures that $0 \leq M(i,j) \leq 1$, for all i, j. In order to make sure we have a maximization problem, we define the following objective function: $f_D(S) = n - \sum_{i \in D} \min_{j \in S} M(i,j)$, where $n = |D| = 10000$.

Observation 3.7. *The function f_D is λ-decomposable for $\lambda = 1$ (and hence has sensitivity 1).*

1. Under the assumption that each pickup corresponds to a unique individual.



(a) $L_{Popular}$ (b) L_{Grid} (c) $L_{Popular}$ (d) L_{Grid}

(e) $L_{Popular}$: Non-private greedy (f) $L_{Popular}$: EM-based greedy (g) L_{Grid}: Non-private greedy (h) L_{Grid}: EM-based greedy

Figure 3: (a) and (b) show utility for various cardinalities (k). (c) and (d) fix $k = 3$ and show utility for various privacy parameters (ϵ). Utility is normalized to be between 0 and 1. (e) - (h) shows a representative top 3 set under various settings.

This form of objective function is known to be monotone submodular and so we can use the greedy algorithms studied in this section. We use $\epsilon = 0.1$ and $\delta = 2^{-20}$. For our settings of parameters, "basic composition" outperforms "advanced composition," so the privacy budget of $\epsilon = 0.1$ is split equally across the k iterations, meaning the mechanism at each iteration uses $\epsilon_0 = \frac{\epsilon}{k}$. Our figures plot the average utility across 100 simulations.

From Figures 3(a) and (b) we see that the results for both $L_{Popular}$ and L_{Grid} are relatively similar and unsurprising. The non-private greedy algorithm achieves the highest utility, but both the exponential mechanism (EM)-based greedy and large margin mechanism (LMM)-based greedy algorithms exhibit comparable utility while preserving a high level of privacy. Interestingly, we also see that the utilities of the EM-based and LMM-based algorithms are almost identical for both $L_{Popular}$ and L_{Grid}. This indicates that our mechanisms are actually selecting good locations, rather than just getting lucky because there are a lot of good locations to choose from.

Figures 3(c) and (d) show how the utility of the EM-based and LMM-based algorithms vary with the privacy parameter ϵ. We can also think of this as varying the dataset size for a fixed ϵ. We fix $k = 3$ and take the average of 100 simulations for each value of ϵ. We see that even for very small

ϵ, our algorithms outperform fully random selection. As ϵ increases, so does the utility. It is not shown in this figure, but varying δ has very little effect.

From Figures 3(e) - (h), we see that the both the non-private and private algorithms select public locations that are relatively close to each other. For example, for the $L_{Popular}$ set of locations, the Empire State Building is close to the New York Public Library, the Soho Grand Hotel is close to NYU, and the Grand Army Plaza is close to the UN Headquarters. As a result, the private mechanisms manage to achieve comparable utility, while also masking the users' exact locations.

The theory suggests that, at least asymptotically, the large margin mechanism-based algorithm should outperform the exponential mechanism-based algorithm. However, in our experiments, we find that the large margin mechanism is generally only able to find a margin in the first iteration of the greedy algorithm. This is because the threshold for finding a margin depends only on ϵ, δ, and n and thus it stays the same across all k iterations. On the other hand, the marginal gain at each iteration drops very quickly, so the mechanism fails to find a margin and thus samples from all remaining locations. However, since the large margin mechanism spends half of its privacy budget to try to find a margin, the sampling step gives slightly worse guarantees than does the plain exponential mechanism, thus giving us the slightly weaker results we see in the figures.

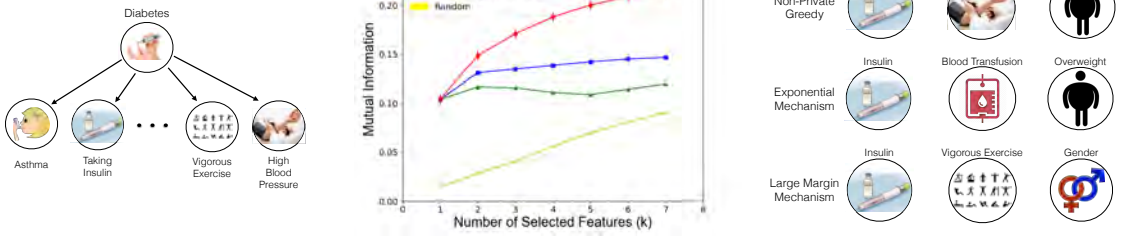
3.7.2 Feature Selection Privacy

We analyze a dataset created from a combination of National Health Examination Surveys ranging from 2007 to 2014 (NHANESDataset). There are $n = 23,876$ individuals in the dataset with information on whether or not they have diabetes, along with $m = 23$ other potentially related binary health features. Our goal is to privately select k of these features that provide as much information about the diabetes class variable as possible.

More specifically, our goal is to maximize the mutual information between Y and X_S, where Y is a binary random variable indicating whether or not an individual has diabetes and X_S is a random variable that represents a set S of k binary health features. Mutual information takes the form:

$$I(Y;X) = \sum_{y\in\mathcal{Y}}\sum_{x\in\mathcal{X}} p(x,y)\log_2\left(\frac{p(x,y)}{p(x)p(y)}\right).$$

Under the Naive Bayes assumption, we suppose the joint distribution on $(Y, X_1, \ldots, X_k)$ takes the



(a) Graphical model of Naive Bayes

(b) Expected mutual information

(c) Representative top 3 features

Figure 4: Privately selecting health features, from national health examination surveys, that correlate most with diabetes.

form $p(y, x_1, \ldots, x_k) = p(y) \prod_{i=1}^{k} p(x_i \mid y)$. Therefore, we can easily specify the entire probability distribution by finding each $p(x_i \mid y)$. We estimate each $p(x_i \mid y)$ by counting frequencies in the dataset.

Our goal is to choose a size k subset S of the features in order to maximize $f_D(S) = I(Y; X_S)$. Mutual information (under the Naive Bayes assumption) for feature selection is known to be monotone submodular in S (Krause and Guestrin, 2005), and thus we can apply the greedy algorithms described in this section.

Claim 3.8. *In iteration i of the greedy algorithm, the sensitivity of $f_D(S)$ is $\frac{(2i+1)\log_2(n)}{n}$.*

We run 1,000 simulations with $\epsilon = 1.0$ and $\delta = 2^{-20}$. As we can see from Figure 4(b), our private mechanisms maintain a comparable utility relative to the non-private algorithm. We also observe an interesting phenomenon where the expected utility obtained by our mechanism is not necessarily monotonically increasing with the number of features selected. This is an artifact of the fact that if we are selecting k features, then composition requires us to divide ϵ so that each iteration uses privacy budget $\frac{\epsilon}{k}$. This is problematic for this particular application because there happens to be one feature (insulin administration) that has much higher value than the rest. Therefore, the reduced probability of picking this single best feature (as a result of the lower privacy parameter $\frac{\epsilon}{k}$) is not compensated for by selecting more features.

From Figure 4(c), we see that both the private and non-private mechanisms generally select insulin administration as the top feature. However, while all three of the top features selected by the

non-private algorithm are clearly related to diabetes, the non-private mechanisms tend to select one feature (in our case, gender or having received a blood transfusion) that may not be quite as relevant.

3.8 Conclusion

We have presented a general framework for maximizing submodular functions while guaranteeing differential privacy. Our results demonstrate that simple and flexible greedy algorithms can preserve privacy while achieving competitive guarantees for a variety of submodular maximization problems: for all functions under cardinality constraints, as well as for monotone functions under matroid and p-extendible system constraints. Via our motivation to identify algorithms that could be made differentially private, we discovered a non-monotone submodular maximization algorithm that achieves guarantees that are novel even without concern for privacy. Finally, our experiments show thatour algorithms are indeed competitive with their non-private counterparts.

4. Scalable Submodularity

As mentioned earlier, one of the primary reasons that submodularity has been of interest to the broader machine learning community is that it allows for provably efficient optimization. Indeed, as we have seen in previous sections, there are many greedy-based linear-time algorithms. However, as the world continues to produce more and more data, even linear-time algorithms are becoming too slow. In this section, we will push the boundaries of scalable submodularity and show how we can leverage the mathematical notion of diminishing returns to break past the linear-time barrier.

This section will focus on two primary directions. First, we will look at the idea of **two-stage submodularity**, where the goal is to use some given training functions to reduce the ground set so that optimizing new functions (drawn from the same distribution) over the reduced set provides almost as much value as optimizing them over the entire ground set. In particular, we will explore streaming and distributed extentesions to existing algorithms for two-stage submodular maximization.

The second section will focus more closely on streaming submodularity. In a nutshell, we present a novel streaming algorithm that achieves optimal bounds both for the approximation guarantee as well as the space complexity. Furthermore, we also present a streaming algorithm with low adaptive complexity (both for the single and multi-stream settings) that can take advantage of parallel computation and thus scale to even larger amounts of data.

4.1 Data Summarization At Scale: A Two-Stage Submodular Approach

4.1.1 Introduction To Two-Stage Submodular Maximization

In the context of machine learning, it is not uncommon to have to repeatedly optimize a set of functions that are fundamentally related to each other. We have already seen that submodular functions can be approximately optimized in linear time. However, modern datasets are growing so large that even linear time solutions can be computationally expensive. Ideally, we want to find a sublinear summary of the given dataset so that optimizing these related functions over this reduced subset is nearly as effective, but not nearly as expensive, as optimizing them over the full dataset.

As a concrete example, suppose Uber is trying to give their drivers suggested waiting locations across New York City based on historical rider pick-ups. Even if they discretize the potential waiting locations to just include points at which pick-ups have occurred in the past, there are still hundreds of thousands, if not millions, of locations to consider. If they wish to update these ideal waiting locations every day (or at any routine interval), it would be invaluable to be able to drastically reduce the number of locations that need to be evaluated, and still achieve nearly optimal results.

In this scenario, each day would have a different function that quantifies the value of a set of locations for that particular day. For example, in the winter months, spots near ice skating rinks would be highly valuable, while in the summer months, waterfront venues might be more prominent. On the other hand, major tourist destinations like Times Square will probably be busy year-round.

In other words, although the most popular pick-up locations undoubtedly vary over time, there is also some underlying distribution of the user behavior that remains relatively constant and ties the various days together. This means that even though the functions for future days are technically unknown, if we can select a good reduced subset of candidate locations based on the functions derived from historical data, then this same reduced subset should perform well on future functions that we cannot explicitly see yet.

In more mathematical terms, consider some unknown distribution of functions $\mathbb{D}$ and a ground set Ω of n elements to pick from. We want to select a subset S of ℓ elements (with $\ell \ll n$) such that optimizing functions (drawn from this distribution $\mathbb{D}$) over the reduced subset S is comparable to optimizing them over the entire ground set Ω.

Algorithm	Approx.	Time Complexity	Setup
LocalSearch (Balkanski et al., 2016)	$1/2(1-1/e)$	$O(km\ell n^2 \log n)$	Centralized
Replacement-Greedy (Stan et al., 2017)	$1/2(1-1/e^2)$	$O(km\ell n)$	Centralized
Replacement-Streaming (ours)	$1/7$	$O(kmn \log \ell)$	Streaming
Replacement-Distributed (R) (ours)	$1/4(1-1/e^2)$	$O(km\ell n/\mathsf{M} + \mathsf{M}km\ell^2)$	Distributed
Distributed-Fast (R) (ours)	0.107	$O(kmn \log \ell/\mathsf{M} + \mathsf{M}km\ell^2 \log \ell)$	Distributed

Table 2: Comparison of algorithms for two-stage monotone submodular maximization. Bounds that hold in expectation are marked (R). For distributed algorithms, we report the time complexity of each single machine, where M represent the number of machines.

This problem was first introduced by Balkanski et al. (2016) as **two-stage submodular maximization**. This name comes from the idea that the overall framework can be viewed as two separate stages. First, we want to use the given functions to select a representative subset S, that is ideally sublinear in size of the entire ground set Ω. In the second stage, for any functions drawn from this same distribution, we can optimize over S, which will be much faster than optimizing over Ω.

Our Contributions. In today's era of massive data, an algorithm is rarely practical if it is not scalable. We build on existing work to provide solutions for two-stage submodular maximization in both the streaming and distributed settings. Table 2 summarizes the theoretical results and compares them with the previous state of the art. All proofs are given in Appendix C.

4.1.2 Related Work

Data summarization is one of the most natural applications that falls under the umbrella of submodularity. As such, there are many existing works applying submodular theory to a variety of important summarization settings. For example, Mirzasoleiman et al. (2013) used an exemplar-based clustering approach to select representative images from the *Tiny Images* dataset (Torralba et al., 2008). Kirchhoff and Bilmes (2014) and Feldman et al. (2018) also worked on submodular image summarization, while Lin and Bilmes (2011) and Wei et al. (2013) focused on document summarization.

There have also been many successful efforts in scalable submodular optimization. For our distributed implementation we will primarily build on the framework developed by Barbosa et al. (2015). Other similar algorithms include works by Mirzasoleiman et al. (2013) and Mirrokni and Zadimoghaddam (2015), as well as Kumar et al. (2013). Balkanski and Singer (2018), Liu and Vondrák (2018), and Kazemi et al. (2020) also wrote papers broadly related to distributed submodular optimization.

In terms of the streaming setting, there are two existing works we will focus on: Badanidiyuru et al. (2014) and Buchbinder et al. (2015). The key difference between the two is that Badanidiyuru et al. (2014) relies on thresholding and will terminate as soon as k elements are selected from the stream, while Buchbinder et al. (2015) will continue through the end of the stream, swapping elements in and out. Haba et al. (2020) extend the streaming framework for submodular maximization beyond monotone functions and cardinality constraints.

Repeated optimization of related submodular functions has been a well-studied problem with works on structured prediction (Lin and Bilmes, 2012), submodular bandits (Yue and Guestrin, 2011; Chen et al., 2017), and online submodular optimization (Jegelka and Bilmes, 2011). However, unlike our work, these approaches are not concerned with data summarization as a key pre-processing step.

The problem of two-stage submodular maximization was first introduced by Balkanski et al. (2016). They present two algorithms with strong approximation guarantees, but both runtimes are prohibitively expensive. Recently, Stan et al. (2017) presented a new algorithm known as Replacement-Greedy that improved the approximation guarantee from $\frac{1}{2}(1-\frac{1}{e})$ to $\frac{1}{2}(1-\frac{1}{e^2})$ and the run time from $O(km\ell n^2 \log(n))$ to $O(km\ell n)$. They also show that, under mild conditions over the functions, maximizing over the sublinear summary can be arbitrarily close to maximizing over the entire ground set. In a nutshell, their method indirectly constructs the summary S by greedily building up solutions T_i for each given function f_i simultaneously over ℓ rounds.

Although Balkanski et al. (2016) and Stan et al. (2017) presented centralized algorithms with constant factor approximation guarantees, there is a dire need for scalable solutions in order for the algorithm to be practically useful. In particular, the primary purpose of two-stage submodular maximization is to tackle problems where the dataset is too large to be repeatedly optimized by simple greedy-based approaches. As a result, in many cases, the datasets can be so large that existing algorithms cannot even be run once. The greedy approach requires that the entire data must fit into main memory, which may not be possible, thus requiring a streaming-based solution. Furthermore, even if we have enough memory, the problem may simply be so large that it requires a distributed approach in order to run in any reasonable amount of time.

Set	Cardinality	Description
F	m	Set of functions $(f_1, \ldots, f_m)$ drawn from an unknown distribution $\mathbb{D}$ of monotone submodular functions.
Ω	n	Given ground set of all elements. Generally so large that even greedy is too expensive.
$S^{m,\ell}$	ℓ	Optimal solution to Problem 2, i.e. $S^{m,\ell} = \arg\max_{S \subseteq \Omega, \lvert S \rvert \leq \ell} \frac{1}{m} \sum_{i=1}^{m} \max_{\lvert T \rvert \leq k, T \subseteq S} f_i(T)$.
$S_i^{m,\ell}$	k	Optimal solution to each function f_i from set $S^{m,\ell}$, i.e. $S_i^{m,\ell} = \arg\max_{S \subseteq S^{m,\ell}, \lvert S \rvert \leq k} f_i(S)$.
OPT	1	The value of the optimal solution to Problem 2, i.e. $\text{OPT} = \frac{1}{m} \sum_{i=1}^{m} f_i(S_i^{m,\ell})$.
S	ℓ	Reduced subset of elements we want to select. Ideally sublinear in n, but still representative.
T_i	k	Solution we select for each function f_i (chosen from S), i.e., $T_i \subset S$.

Table 3: Summary of important terminology

4.1.3 Problem Definition

Consider some unknown distribution $\mathbb{D}$ of monotone submodular functions and a ground set Ω of n elements to choose from. We want to select a set S of at most ℓ items that maximizes the following function:

$$G(S) = \mathbb{E}_{f \sim \mathbb{D}}[\max_{T \subseteq S, \lvert T \rvert \leq k} f(T)]. \tag{1}$$

That is, the set S we choose should be optimal in expectation over all functions in this distribution $\mathbb{D}$. However, in general, the distribution $\mathbb{D}$ is unknown and we only have access to a small set of functions $F = (f_1, \ldots, f_m)$ drawn from $\mathbb{D}$. Therefore, the best approximation we have is to optimize the following related function:

$$G_m(S) = \frac{1}{m} \sum_{i=1}^{m} \max_{T_i^* \subseteq S, \lvert T_i^* \rvert \leq k} f_i(T_i^*). \tag{2}$$

Stan et al. (2017, Theorem 1) shows that with enough sample functions, $G_m(S)$ becomes an arbitrarily good approximation to $G(S)$.

To be clear, each $T_i^* \subset S$ is the corresponding size k optimal solution for f_i. However, in general we cannot find the true optimal T_i^*, so throughout this section we will use T_i to denote the approximately-optimal size k solution we select for each f_i. Table 3 summarizes the important terminology and can be used as a reference, if needed.

It is very important to note that although each function f_i is monotone submodular, $G(S)$ is **not** submodular (Balkanski et al., 2016), and thus using the regular greedy algorithm to directly build up S will give no theoretical guarantees. We also note that although $G(S)$ is an instance of an XOS function (Feige, 2009), existing methods that use the XOS property would require an exponential number of evaluations in this scenario (Stan et al., 2017).

4.1.4 Streaming Algorithm for Two-Stage Submodular Maximization

In many applications, the data naturally arrives in a streaming fashion. This may be because the data is too large to fit in memory, or simply because the data is arriving faster than we can store it. Therefore, in the streaming setting we are shown one element at a time and we must immediately decide whether or not to keep this element. There is a limited number of elements we can hold at any one time and once an element is rejected it cannot be brought back.

There are two general approaches for submodular maximization (under the cardinality constraint k) in the streaming setting: (i) Badanidiyuru et al. (2014) introduced a thresholding-based framework where each element from the stream is added only if its marginal value is at least $\frac{1}{2k}$ of the optimum value. The optimum is usually not known a priori, but they showed that with only a logarithmic increase in memory requirement, it is possible to efficiently guess the optimum value. (ii) Buchbinder et al. (2015) introduced streaming submodular maximization with preemption. At each step, they keep a solution A of size k with value $f(A)$. Each incoming element is added if and only if it can be exchanged with a current element of A for a net gain of at least $\frac{f(A)}{k}$. In this section, we combine these two approaches in a novel and non trivial way in order to design a streaming algorithm (called Replacement-Streaming) for the two-stage submodular maximization problem.

The goal of Replacement-Streaming is to pick a set S of at most ℓ elements from the data stream, where we keep sets $T_i \subseteq S, 1 \leq i \leq m$ as the solutions for functions f_i. We continue to process elements until one of the two following conditions holds: (i) ℓ elements are chosen, or (ii) the data stream ends. This algorithm starts from empty sets S and $\{T_i\}$. For every incoming element u^t, we use the subroutine Exchange to determine whether we should keep that element or not. To formally describe Exchange, we first need to define a few notations.

We define the marginal gain of adding an element x to a set A as follows: $f_i(x|A) = f_i(x+A) - f_i(A)$. For an element x and set A, $\textsc{Rep}_i(x, A)$ is an element of A such that removing it from A and replacing it with x results in the largest gain for function f_i, i.e.,

Algorithm 5 EXCHANGE

Input: $u, S, \{T_i\}, \tau$ and α ▷ ∇_i terms use α
if $|S| < \ell$ **then**
  if $\frac{1}{m}\sum_{i=1}^{m} \nabla_i(u, T_i) \geq \tau$ **then**
    $S \leftarrow S + u$
    for $1 \leq i \leq m$ **do**
      if $\nabla_i(u, T_i) > 0$ **then**
        if $|T_i| < k$ **then**
          $T_i \leftarrow T_i + u$
        else
          $T_i \leftarrow T_i + u - \text{REP}(u, T_i)$

$$\text{REP}_i(x, A) = \arg\max_{y \in A} f_i(A + x - y) - f_i(A). \tag{3}$$

The value of this largest gain is represented by

$$\Delta_i(x, A) = f_i(A + x - \text{REP}_i(x, A)) - f_i(A). \tag{4}$$

We define the gain of an element x with respect to a set A as follows:

$$\nabla_i(x, A) = \begin{cases} \mathbb{1}_{\{f_i(x|A) \geq (\alpha/k) \cdot f_i(A)\}} f_i(x|A) & \text{if } |A| < k, \\ \mathbb{1}_{\{\Delta_i(x,A) \geq (\alpha/k) \cdot f_i(A)\}} \Delta_i(x, A) & \text{o.w.}, \end{cases}$$

where $\mathbb{1}$ is the indicator function. That is, $\nabla_i(x, A)$ tells us how much we can increase the value of $f_i(A)$ by either adding x to A (if $|A| < k$) or optimally swapping it in (if $|A| = k$). However, if this potential increase is less than $\frac{\alpha}{k} \cdot f_i(A)$, then $\nabla_i(x, A) = 0$. In other words, if the gain of an element does not pass a threshold of $\frac{\alpha}{k} \cdot f_i(A)$, we consider its contribution to be 0.

An incoming element is picked if the average of the ∇_i terms is larger than or equal to a threshold τ. Indeed, for u^t, the EXCHANGE routine computes the average gain $\frac{1}{m}\sum_{i=1}^{m} \nabla_i(u^t, T_i)$. If this average gain is at least τ, then u^t is added to S; u^t is also added to all sets T_i with $\nabla_i(u^t, T_i) > 0$. Algorithm 5 explains EXCHANGE in detail.

Now we define the optimum solution to Problem 2 by

$$S^{m,\ell} = \arg\max_{S \subseteq \Omega, |S| \leq \ell} \frac{1}{m} \sum_{i=1}^{m} \max_{|T| \leq k, T \subseteq S} f_i(T),$$

where the optimum solution to each function is defined by

Algorithm 6 REPLACEMENT-STREAMING-KNOW-OPT

Input: OPT, α and β
Output: Sets S and $\{T_i\}_{1\leq i\leq m}$, where $T_i \subset S$
$S \leftarrow \emptyset$ and
$T_i \leftarrow \emptyset$ for all $1 \leq i \leq m$
for every arriving element u^t **do**
    EXCHANGE$(u^t, S, \{T_i\}, \frac{\text{OPT}}{\beta\ell}, \alpha)$
Return: S and $\{T_i\}_{1\leq i\leq m}$

$$S_i^{m,\ell} = \underset{S\subseteq S^{m,\ell}, |S|\leq k}{\arg\max} f_i(S).$$

We define $\text{OPT} = \frac{1}{m}\sum_{i=1}^{m} f_i(S_i^{m,\ell})$.

In Section 4.1.5, we assume that the value of OPT is known a priori. This allows us to design REPLACEMENT-STREAMING-KNOW-OPT, which has a constant factor approximation guarantee. Furthermore, in Section 4.1.6, we show how we can efficiently guess the value of OPT by a moderate increase in the memory requirement. This enables us to finally explain REPLACEMENT-STREAMING.

4.1.5 KNOWING OPT

If OPT is somehow known a priori, we can use REPLACEMENT-STREAMING-KNOW-OPT. As shown in Section 6, we begin with empty sets S and $\{T_i\}$. For each incoming element u^t, it uses EXCHANGE to update sets S and $\{T_i\}$. The threshold parameter τ in EXCHANGE is set to $\frac{\text{OPT}}{\beta\ell}$ for a constant value of β. This threshold guarantees that if an element is added to S, then the average of functions f_i over T_i's is increased by a value of at least $\frac{\text{OPT}}{\beta\ell}$. Therefore, if we end up with ℓ elements in S, we guarantee that $\frac{1}{m}\sum_{i=1}^{m} f_i(T_i) \geq \frac{\text{OPT}}{\beta}$. On the other hand, if $|S| < \ell$, we are still able to prove that our algorithm has picked good enough elements such that $\frac{1}{m}\sum_{i=1}^{m} f_i(T_i) \geq \frac{\alpha\cdot(\beta-1)\cdot\text{OPT}}{\beta\cdot((\alpha+1)^2+\alpha)}$. The pseudocode of REPLACEMENT-STREAMING-KNOW-OPT is provided in Algorithm 6.

Theorem 4.1. *The approximation factor of* REPLACEMENT-STREAMING-KNOW-OPT *is at least* $\min\{\frac{\alpha(\beta-1)}{\beta\cdot((\alpha+1)^2+\alpha)}, \frac{1}{\beta}\}$. *Hence, for* $\alpha = 1$ *and* $\beta = 6$ *the competitive ratio is at least* $1/6$.

4.1.6 Guessing OPT in the Streaming Setting

In this section, we discuss ideas on how to efficiently guess the value of OPT, which is generally not known a priori. First consider Lemma 4.2, which provides bounds on OPT.

Lemma 4.2. *Assume* $\delta = \frac{1}{m} \max_{u \in \Omega} \sum_{i=1}^{m} f_i(u)$. *Then we have* $\delta \leq \text{OPT} \leq \ell \cdot \delta$.

Now consider the following set:

$$\Gamma = \{(1+\epsilon)^l \mid l \in \mathbb{Z}, \frac{\delta}{1+\epsilon} \leq (1+\epsilon)^l \leq \ell \cdot \delta\}$$

We define $\tau_l = (1+\epsilon)^l$. From Lemma 4.2, we know that one of the $\tau_l \in \Gamma$ is a good estimate of OPT. More formally, there exists a $\tau_l \in \Gamma$ such that $\frac{\text{OPT}}{1+\epsilon} \leq \tau_l \leq \text{OPT}$. For this reason, we should run parallel instances of Algorithm 6, one for each $\tau_l \in \Gamma$. The number of such thresholds is $O(\frac{\log \ell}{\epsilon})$. The final answer is the best solution obtained among all the instances.

Note that we do not know the value of δ in advance. So we would need to make one pass over the data to learn δ, which is not possible in the streaming setting. The question is, can we get a good enough estimate of δ within a single pass over the data? Let's define $\delta^t = \frac{1}{m} \max_{u^{t'}, t' \leq t} \sum_{i=1}^{m} f_i(u^{t'})$ as our current guess for the maximum value of δ. Unfortunately, getting δ^t as an estimate of δ does not resolve the problem. This is due to the fact that a newly instantiated threshold τ could potentially have already seen elements with additive value of $\tau/(\beta\ell)$. For this reason, we instantiate thresholds for an increased range of $\delta^t/(1+\epsilon) \leq \tau_l \leq \ell \cdot \beta \cdot \delta^t$. To show that this new range would solve the problem, first consider the next lemma.

Lemma 4.3. *For the maximum gain of an incoming element* u^t, *we have* $\frac{1}{m} \sum_{i=1}^{m} \nabla_i(u^t, T_i^{t-1}) \leq \delta^t$.

We need to show that for a newly instantiated threshold τ at time $t+1$, the gain of all elements which arrived before time $t+1$ is less than τ; therefore this new instance of the algorithm would not have picked them if it was instantiated from the beginning. To prove this, note that since τ is a new threshold at time $t+1$, we have $\tau > \frac{\ell \cdot \beta \cdot \delta^t}{\beta \cdot \ell} = \delta^t$. From Lemma 4.3 we conclude that the marginal gain of all the $u^{t'}, t' \leq t$ is less than τ and Exchange would not have picked them. The Replacement-Streaming algorithm is shown pictorially in Figure 5 and the pseudocode is given in Algorithm 7.

Algorithm 7 REPLACEMENT-STREAMING

$\Gamma^0 = \{(1+\epsilon)^l | l \in \mathbb{Z}\}$
For each $\tau \in \Gamma^0$ set $S_\tau \leftarrow \emptyset$ and $T_{\tau,i} \leftarrow \emptyset$ for all $1 \leq i \leq m$ ▷ Maintain the sets lazily
$\delta^0 \leftarrow 0$
for every arriving element u^t **do**
    $\delta^t = \max\{\delta^{t-1}, \frac{1}{m}\sum_{i=1}^{m} f_i(u^t)\}$
    $\Gamma^t = \{(1+\epsilon)^l \mid l \in \mathbb{Z}, \frac{\delta^t}{(1+\epsilon)\cdot\beta\cdot\ell} \leq (1+\epsilon)^l \leq \delta^t\}$
    Delete all S_τ and $T_{\tau,i}$ such that $\tau \notin \Gamma^t$
    for all $\tau \in \Gamma^t$ **do**
        EXCHANGE$(u^t, S_\tau, \{T_{\tau,i}\}_{1\leq i\leq m}, \tau, \alpha)$
Return: $\arg\max_{\tau\in\Gamma^n}\{\frac{1}{m}\sum_{i=1}^{m} f_i(T_{\tau,i})\}$

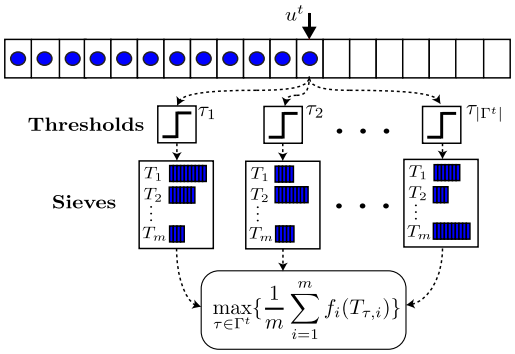


Figure 5: Illustration of REPLACEMENT-STREAMING. Stream of data arrives at any arbitrary order. At each step t, the set of thresholds Γ^t is updated based on a new estimation of δ^t. Note that at each time the number of such thresholds is bounded. For each $\tau \in \Gamma^t$ there is a running instance of the streaming algorithm.

Theorem 4.4. *Algorithm 7 satisfies the following properties:*

- *It outputs sets S and $\{T_i\} \subset S$ for $1 \leq i \leq m$, such that $|S| \leq \ell, |T_i| \leq k$ and $\frac{1}{m}\sum_{i=1}^{m} f_i(T_i) \geq \min\{\frac{\alpha(\beta-1)}{\beta((\alpha+1)^2+\alpha)}, \frac{1}{\beta(1+\epsilon)}\} \cdot \mathrm{OPT}$.*
- *For $\alpha = 1$ and $\beta = \frac{6+\epsilon}{1+\epsilon}$ the approximation factor is at least $\frac{1}{6+\epsilon}$. For $\epsilon = 1.0$ the approximation factor is $1/7$.*
- *It makes one pass over the dataset and stores at most $O(\frac{\ell \log \ell}{\epsilon})$ elements. The update time per each element is $O(\frac{km \log \ell}{\epsilon})$.*

4.1.7 DISTRIBUTED ALGORITHM FOR TWO-STAGE SUBMODULAR MAXIMIZATION

In recent years, there have been several successful approaches to the problem of distributed submodular maximization (Kumar et al., 2013; Mirzasoleiman et al., 2013; Mirrokni and Zadimoghaddam, 2015; Barbosa et al., 2015). Specifically, Barbosa et al. (2015) proved that the following simple

Algorithm 8 REPLACEMENT-DISTRIBUTED

for $e \in \Omega$ **do**
    Assign e to a machine chosen uniformly at random
Run REPLACEMENT-GREEDY on each machine l to obtain S^l and $\{T_i^l\}$ for $1 \leq i \leq m$
$S, \{T_i\} \leftarrow \arg\max_{S^l, \{T_i^l\}} \frac{1}{m} \sum_{i=1}^{m} f_i(T_i^l)$
$S', \{T_i'\} \leftarrow$ REPLACEMENT-GREEDY$(\bigcup_l S^l)$
Return: $\arg\max\{\frac{1}{m} \sum_{i=1}^{m} f_i(T_i), \frac{1}{m} \sum_{i=1}^{m} f_i(T_i')\}$

procedure results in a distributed algorithm with a constant factor approximation guarantee: (i) randomly split the data amongst M machines, (ii) run the classical greedy on each machine and pass outputs to a central machine, (iii) run another instance of the greedy algorithm over the union of all the collected outputs from all M machines, and (iv) output the maximizing set amongst all the collected solutions. Although our objective function $G(S)$ is **not** submodular, we use a similar framework and still manage to prove that our algorithms achieve constant factor approximations to the optimal solution.

In REPLACEMENT-DISTRIBUTED (Algorithm 8), a central machine first randomly partitions data among M machines. Next, each machine runs REPLACEMENT-GREEDY (Stan et al., 2017) on its assigned data. The outputs $S^l, \{T_i^l\}$ of all the machines are sent to the central machine, which runs another instance of REPLACEMENT-GREEDY over the union of all the received answers. Finally, the highest value set amongst all collected solutions is returned as the final answer. See Section C.5 for a detailed explanation of REPLACEMENT-GREEDY.

Theorem 4.5. *The* REPLACEMENT-DISTRIBUTED *algorithm outputs sets* $S^*, \{T_i^*\} \subset S$*, with* $|S^*| \leq \ell, |T_i^*| \leq k$*, such that*

$$\mathbb{E}[\frac{1}{m} \sum_{i=1}^{m} f_i(T_i^*)] \geq \frac{\alpha}{2} \cdot \mathrm{OPT},$$

where $\alpha = \frac{1}{2}(1 - \frac{1}{e^2})$*. The time complexity of algorithm is* $O(km\ell n/\mathsf{M} + \mathsf{M}km\ell^2)$*.*

Unfortunately, for very large datasets, the time complexity of REPLACEMENT-GREEDY could be still prohibitive. For this reason, we can use a modified version of REPLACEMENT-STREAMING (called REPLACEMENT-PSEUDO-STREAMING) to design an even more scalable distributed algorithm. This algorithm receives all elements in a centralized way, but it uses a predefined order to generate a (pseudo) stream before processing the data. This consistent ordering is used to ensure that the output of REPLACEMENT-PSEUDO-STREAMING is independent of the random or-

dering of the elements. The only other difference between REPLACEMENT-PSEUDO-STREAMING and REPLACEMENT-STREAMING is that it outputs all sets $S_\tau, \{T_{\tau,i}\}$ for all $\tau \in \Gamma^n$ (instead of just the maximum). We use this modified algorithm as one of the main building blocks for DISTRIBUTED-FAST (outlined in Section C.4).

Theorem 4.6. *The* DISTRIBUTED-FAST *algorithm outputs sets* $S^*, \{T_i^*\} \subset S$*, with* $|S^*| \leq \ell, |T_i^*| \leq k$*, such that*

$$\mathbb{E}[\frac{1}{m}\sum_{i=1}^{m} f_i(T_i^*)] \geq \frac{\alpha \cdot \gamma}{\alpha + \gamma} \cdot \mathrm{OPT},$$

where $\alpha = \frac{1}{2}(1 - \frac{1}{e^2})$ *and* $\gamma = \frac{1}{6+\epsilon}$*. The time complexity of algorithm is* $O(^{kmn \log \ell}/_{\mathsf{M}} + \mathsf{M}km\ell^2 \log \ell)$.

From Theorems 4.5 and 4.6, we conclude that the optimum number of machines M for REPLACEMENT-DISTRIBUTED and DISTRIBUTED-FAST is $O(\sqrt{n/\ell})$ and $O(\sqrt{n}/\ell)$, respectively. Therefore, DISTRIBUTED-FAST is a factor of $O(\sqrt{n}/\log \ell)$ and $O(\sqrt{\ell}/\log \ell)$ faster than REPLACEMENT-GREEDY and REPLACEMENT-DISTRIBUTED, respectively.

4.1.8 STREAMING IMAGE SUMMARIZATION

In this experiment, we will use a subset of the *VOC2012* dataset (Everingham et al.). This dataset has images containing objects from 20 different classes, ranging from birds to boats. For the purposes of this application, we will use $n = 756$ different images and we will consider all $m = 20$ classes that are available. Our goal is to choose a small subset S of images that provides a good summary of the entire ground set Ω. In general, it can be difficult to even define what a good summary of a set of images should look like. Fortunately, each image in this dataset comes with a human-labelled annotation that lists the number of objects from each class that appear in that image.

Using the exemplar-based clustering approach (Mirzasoleiman et al., 2013), for each image we generate an m-dimensional vector x such that x_i represents the number of objects from class i that appear in the image. Figure 6 shows an example.

We define Ω_i to be the set of all images that contain objects from class i, and correspondingly $S_i = \Omega_i \cap S$ (i.e. the images we have selected that contain objects from class i).

We want to optimize the following monotone submodular functions: $f_i(S) = L_i(\{e_0\}) - L_i(S \cup \{e_0\})$, where $L_i(S) = \frac{1}{|\Omega_i|}\sum_{x \in \Omega_i} \min_{y \in S_i} d(x, y)$. We use $d(x, y)$ to denote the "distance" between two images x and y. More accurately, we measure the distance between two images as the ℓ_2 norm

0 - Aeroplane	10 - Dining Table
1 - Bicycle	11 - Dog
2 - Bird	12 - Horse
3 - Boat	13 - Motorbike
4- Bottle	14 - Person
5 - Bus	15 - Potted Plant
6 - Car	16 - Sheep
7 - Cat	17 - Sofa
8 - Chair	18 - Train
9 - Cow	19 - TV Monitor

(a)

(b)

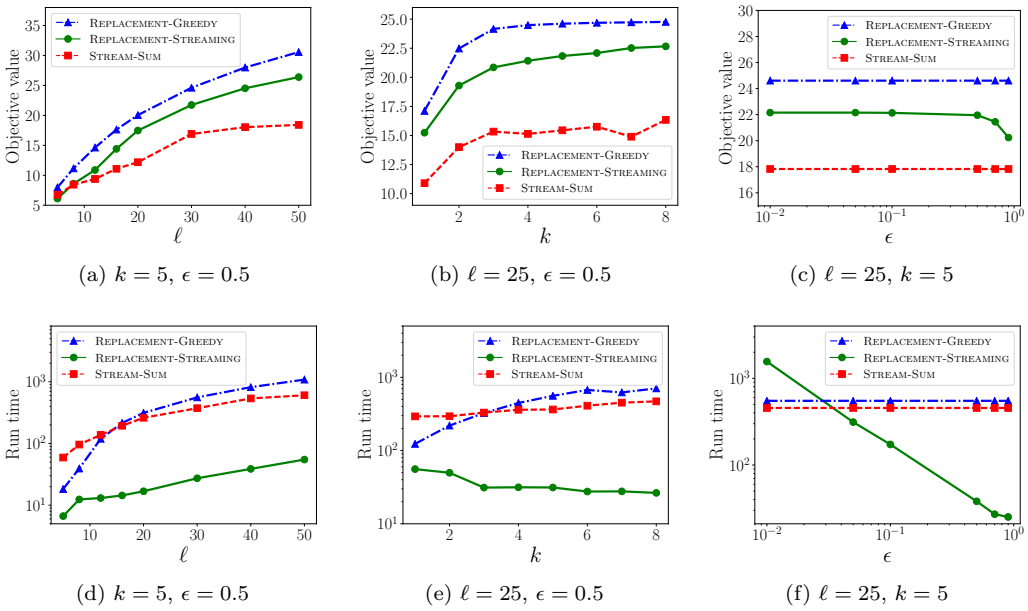
Figure 6: (a) shows the twenty classes that appear in the *VOC2012* dataset. The number adjacent to each class represents the index of that class in the characteristic vector associated with each image. For example, the image shown in (b) contains one boat, one bird, and one person. Therefore, the characteristic vector for this image is [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]. This also means that the image in (b) appears in the sets Ω_2, Ω_4, and Ω_{14}.

between their characteristic vectors. We also use e_0 to denote some auxiliary element, which in our case is the all-zero vector.

Since image data is generally quite storage-intensive, streaming algorithms can be particularly desirable. With this in mind, we will compare our streaming algorithm Replacement-Streaming against the non-streaming baseline of Replacement-Greedy. We also compare against a heuristic streaming baseline that we call Stream-Sum. This baseline first greedily optimizes the submodular function $F(S) = \sum_{i=1}^{m} f_i(S)$ using the streaming algorithm developed by Buchbinder et al. (2015). Having selected ℓ elements from the stream, it then constructs each T_i by greedily selecting k of these elements for each f_i.

To evaluate the various algorithms, we consider two primary metrics: the objective value, which we define as $\sum_{i=1}^{m} f_i(T_i)$, and the wall-clock running time. We note that the trials were run using Python 2.7 on a quad-core Linux machine with 3.3 GHz Intel Core i5 processors and 8 GB of RAM. Figure 7 shows our results.

The graphs are organized so that each column shows the effects of varying a particular parameter, with the objective value being shown in the top row and the running time in the bottom row. The primary observation across all the graphs is that our streaming algorithm Replacement-Streaming not only achieves an objective value that is similar to that of the non-streaming baseline Replacement-Greedy, but it also speeds up the running time by a full order of magnitude. We



(a) $k = 5$, $\epsilon = 0.5$ (b) $\ell = 25$, $\epsilon = 0.5$ (c) $\ell = 25$, $k = 5$

(d) $k = 5$, $\epsilon = 0.5$ (e) $\ell = 25$, $\epsilon = 0.5$ (f) $\ell = 25$, $k = 5$

Figure 7: The top row of graphs shows the objective values achieved by the various algorithms, while the bottom row shows the run times. In (a) and (d) we vary l, the maximum size of the subset S. In (b) and (e), we vary k, the maximum size of the set T_i assigned to each function f_i. Lastly, in (c) and (f), we vary ϵ, the parameter that controls the number of guesses we make for OPT.

also see that Replacement-Streaming outperforms the streaming baseline Stream-Sum in both objective value and running time.

Another noteworthy observation from Figure 7(c) is that ϵ can be increased all the way up to $\epsilon = 0.5$ before we start to see loss in the objective value. Recall that ϵ is the parameter that trades off the accuracy of Replacement-Streaming with the running time by changing the granularity of our guesses for OPT. As seen Figure 7(f), increasing ϵ up to 0.5 also covers the majority of running time speed-up, with diminishing returns kicking in as we get close to $\epsilon = 1$.

Also in the context of running time, we see in Figure 7(e) that Replacement-Streaming actually speeds up as k increases. This seems counter-intuitive at first glance, but one possible reason is that the majority of the time cost for these replacement-based algorithms comes from the swapping that must be done when the T_i's fill up. Therefore, the longer each T_i is not completely full, the faster the overall algorithm will run.

Figure 8 shows some sample images selected by Replacement-Greedy (top) and Replacement-Streaming (bottom). Although the two summaries contain only one image that is exactly the same, we see that the different images still have a similar theme. For example, both images in the second column contain bikes and people; while in the third column, both images contain sheep.

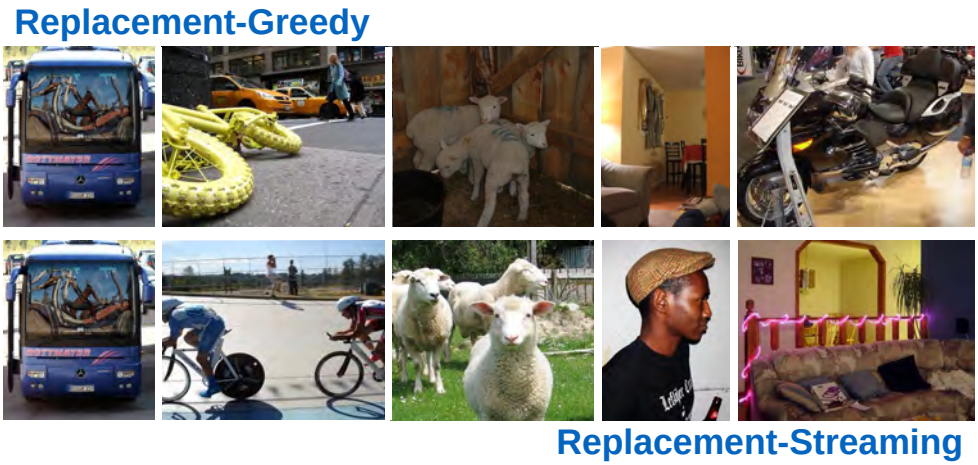


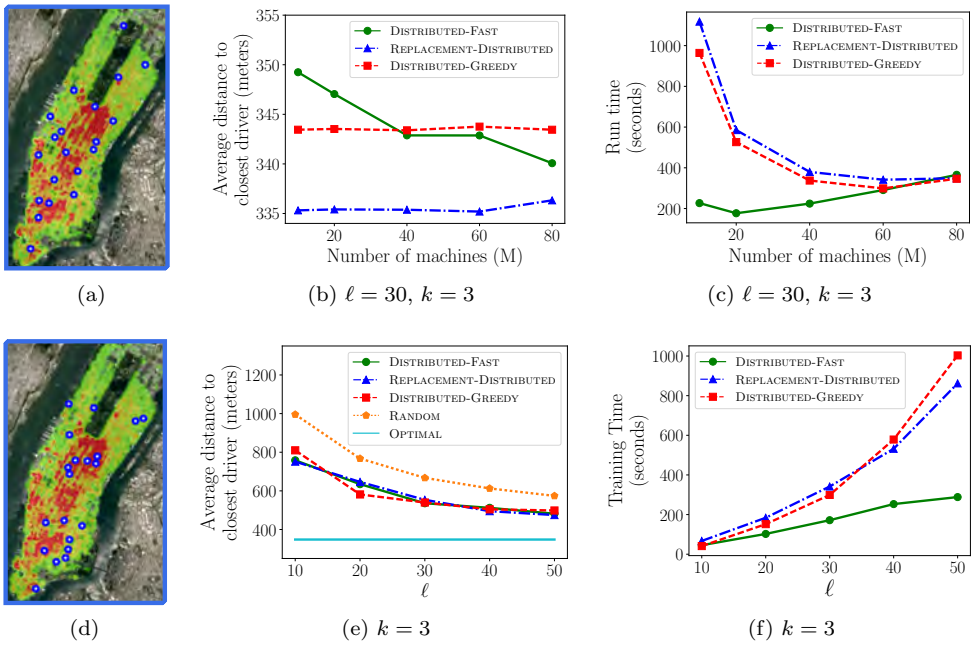
Figure 8: Representative images selected in the different settings.

4.1.9 Distributed Ride-Share Optimization

In this application we want to use past Uber data to select optimal waiting locations for idle drivers. Towards this end, we analyze a dataset of 100,000 Uber pick-ups in Manhattan from September 2014 (UberDataset), where each entry in the dataset is given as a (latitude, longitude) coordinate pair. We model this problem as a classical facility location problem, which is known to be monotone submodular.

Given a set of potential waiting locations for drivers, we want to pick a subset of these locations so that the distance from each customer to his closest driver is minimized. In particular, given a customer location $a = (x_a, y_a)$, and a waiting driver location $b = (x_b, y_b)$, we define a "convenience score" $c(a, b)$ as follows: $c(a, b) = 2 - \frac{2}{1+e^{-200d(a,b)}}$, where $d(a, b) = |x_a - x_b| + |y_a - y_b|$ is the Manhattan distance between the two points.

Next, we need to introduce some functions we want to maximize. For this experiment, we can think about different functions corresponding to different (possibly overlapping) regions around Manhattan. The overlap means that there will still be some inherent connection between the functions, but they are still relatively distinct from each other. More specifically, we construct regions $R_1, \ldots, R_m$ by randomly picking m points across Manhattan. Then, for each point p_i, we want to define the corresponding region R_i by all the pick-ups that have occurred within one kilometer of p_i. However, to keep the problem computationally tractable, we instead randomly select only ten pick-up locations within that same radius. Figure 9(a) shows the center points of the $m = 20$ randomly selected regions, overlaid on top of a heat map of all the customer pick-up locations.



(a) (b) $\ell = 30$, $k = 3$ (c) $\ell = 30$, $k = 3$

(d) (e) $k = 3$ (f) $k = 3$

Figure 9: (a) shows a heatmap of all pick-up locations, as well as the centers of the twenty random regions that define each function f_i. (b) and (c) show the effects of changing the number of machines we use to distribute the computation. (d) shows the centers of the twenty new regions (chosen from the same distribution) used for the evaluation in (e). (f) shows the training time for each summary used in (e).

Given any set of driver waiting locations T_i, we define $f_i(T_i)$ as follows: $f_i(T_i) = \sum_{a \in R_i} \max_{b \in T_i} c(a, b)$. For this application, we will use every customer pick-up location as a potential waiting location for a driver, meaning we have 100,000 elements in our ground set Ω. This large number of elements, combined with the fact that each single function evaluation is computationally intensive, means running the regular Replacement-Greedy will be prohibitively expensive. Hence, we will use this setup to evaluate the two distributed algorithms we presented in Section 4.1.7. We will also compare our algorithms against a heuristic baseline that we call Distributed-Greedy. This baseline will first select ℓ elements using the greedy distributed framework introduced by Mirzasoleiman et al. (2013), and then greedily optimize each f_i over these ℓ elements.

Each algorithm produces two outputs: a small subset S of potential waiting locations (with size $\ell = 30$), as well as a solution T_i (of size $k = 3$) for each function f_i. In other words, each algorithm will reduce the number of potential waiting locations from 100,000 to 30, and then choose 3 different waiting locations for drivers in each region.

In Figure 9(b), we graph the average distance from each customer to his closest driver, which we will refer to as the cost. One interesting observation is that while the cost of Distributed-

FAST decreases with the number of machines, the costs of the other two algorithms stay relatively constant, with REPLACEMENT-DISTRIBUTED marginally outperforming DISTRIBUTED-GREEDY. In Figure 9(c), we graph the run time of each algorithm. We see that the algorithms achieve their optimal speeds at different values of M, verifying the theory at the end of Section 4.1.7. Overall, we see that while all three algorithms have very comparable costs, DISTRIBUTED-FAST is significantly faster than the others.

While in the previous application we only looked at the objective value for the given functions $f_1, \ldots, f_m$, in this experiment we also evaluate the utility of our summary on new functions drawn from the same distribution. That is, using the regions shown in Figure 9(a), each algorithm will select a subset S of potential waiting locations. Using only these reduced subsets, we then greedily select k waiting locations for each of the twenty new regions shown in 9(d).

In Figure 9(e), we see that the summaries from all three algorithms achieve a similar cost, which is significantly better than RANDOM. In this scenario, RANDOM is defined as the cost achieved when optimizing over a random size ℓ subset and OPTIMAL is defined as the cost that is achieved when optimizing the functions over the entire ground set rather than a reduced subset. In Figure 9(f), we confirm that DISTRIBUTED-FAST is indeed the fastest algorithm for constructing each summary. Note that 9(f) is demonstrating how long each algorithm takes to construct a size ℓ summary, not how long it is taking to optimize over this summary.

4.1.10 CONCLUSION

This section focused on the two-stage submodular maximization framework and provided the first streaming and distributed solutions to this problem. In addition to proving constant factor theoretical guarantees, we demonstrated the effectiveness of our algorithms on real world applications in image summarization and ride-share optimization.

4.2 Submodular Streaming in All Its Glory: Tight Approximation, Minimum Memory and Low Adaptive Complexity

4.2.1 INTRODUCTION

In this section, we consider the following optimization problem: given a non-negative monotone submodular function f, find the set S^* of size at most k that maximizes the function f:

$$S^* = \underset{S \subseteq V, |S| \leq k}{\arg\max}\ f(S). \tag{5}$$

We define $\mathrm{OPT} = f(S^*)$. When the data is relatively small and it does not change over time, the greedy algorithm and other fast centralized algorithms provide near-optimal solutions.

However, in many real-world applications, we are dealing with massive streams of images, videos, texts, sensor logs, tweets, and high-dimensional genomics data which are produced from different data sources. These data streams have an unprecedented volume and are produced so rapidly that they cannot be stored in memory, which means we cannot apply classical submodular maximization algorithms. In this section, our goal is to design efficient algorithms for streaming submodular maximization in order to simultaneously provide the best approximation factor, memory complexity, running time, and communication cost.

For problem 5, Norouzi-Fard et al. (2018) proved that any streaming algorithm[2] with a memory $o(n/k)$ cannot provide an approximation guarantee better than $1/2$. Sieve-Streaming is the first streaming algorithm with a constant approximation factor (Badanidiyuru et al., 2014). This algorithm guarantees an approximation factor of $1/2 - \epsilon$ and memory complexity of $O(k \log(k)/\epsilon)$. While the approximation guarantee of their Sieve-Streaming is optimal, the memory complexity is a factor of $\log(k)$ away from the desired lower bound $\Theta(k)$. In contrast, Buchbinder et al. (2015) designed a streaming algorithm with a $1/4$-approximation factor and optimal memory $\Theta(k)$. The first contribution in this section is to answer the following question: Is there a streaming algorithm with an approximation factor arbitrarily close to $1/2$ whose memory complexity is $O(k)$?

Our new algorithm, Sieve-Streaming++, closes the gap between the optimal approximation factor and memory complexity, but it still has some drawbacks. In fact, in many applications of submodular maximization, the function evaluations (or equivalently Oracle queries)[3] are computationally expensive and can take a long time to process.

In this context, the fundamental concept of adaptivity quantifies the number of sequential rounds required to maximize a submodular function, where in each round, we can make polynomially many independent Oracle queries in parallel. More formally, given an Oracle f, an algorithm is ℓ-adaptive if every query $\mathcal{Q}$ to the Oracle f at a round $1 \leq i \leq \ell$ is independent of the answers $f(\mathcal{Q})$ to all other

2. They assume the submodular function is evaluated only on feasible sets of cardinality at most k. In this section, we make the same natural assumption regarding the feasible queries.
3. The Oracle for a submodular function f receives a set S and returns its value $f(S)$.

queries $\mathcal{Q}'$ at rounds j, $i \leq j \leq \ell$ (Balkanski and Singer, 2018). The adaptivity of an algorithm has important practical consequences as low adaptive complexity results in substantial speedups in parallel computing time.

All the existing streaming algorithms require at least one Oracle query for each incoming element. This results in an adaptive complexity of $\Omega(n)$ where n is the total number of elements in the stream. Furthermore, in many real-world applications, data streams arrive at such a fast pace that it is not possible to perform multiple Oracle queries in real time. This could result in missing potentially important elements or causing a huge delay.

Our idea to tackle the problem of adaptivity is to introduce a hybrid model where we allow a machine to buffer a certain amount of data, which allows us to perform many queries in parallel. We design a sampling algorithm that, in only a few adaptive rounds, picks items with good marginal gain and discards the rest. The main benefit of this method is that we can quickly empty the buffer and continue the optimization process. In this way, we obtain an algorithm with optimal approximation, query footprint, and near-optimal adaptivity.

Next, we focus on an additional challenge posed by real-world data where often multiple streams co-exist at the same time. In fact, while submodular maximization over only one stream of data is challenging, in practice there are many massive data streams generated simultaneously from a variety of sources. For example, these multi-source streams are generated by tweets from news agencies, videos and images from sporting events, or automated security systems and sensor logs. These data streams have an enormous volume and are produced so rapidly that they cannot be even transferred to a central machine. Therefore, in the multi-source streaming setting, other than approximation factor, memory and adaptivity, it is essential to keep communication cost low. To solve this problem, we show that a carefully-designed generalization of our proposed algorithm for single-source streams also has an optimal communication cost.

4.2.2 Related Work

Badanidiyuru et al. (2014) were the first to consider a one-pass streaming algorithm for maximizing a monotone submodular function under a cardinality constraint. Buchbinder et al. (2015) improved the memory complexity of (Badanidiyuru et al., 2014) to $\Theta(k)$ by designing a $1/4$ approximation algorithm. Norouzi-Fard et al. (2018) introduced an algorithm for random order streams that beats the $1/2$ bound. Recently, Agrawal et al. (2019) substantially improved the result for random order

streams to an almost tight $1 - 1/e - \varepsilon - O(k^{-1})$ approximation factor. Norouzi-Fard et al. (2018) also studied the multi-pass streaming submodular maximization problem.

Chakrabarti and Kale (2015) studied streaming submodular maximization problem subject to the intersection of p matroid constraints. These results were further extended to more general constraints such as p-matchoids (Chekuri et al., 2015; Feldman et al., 2018). Also, there have been some very recent works to generalize these results to non-monotone submodular functions (Chakrabarti and Kale, 2015; Chekuri et al., 2015; Chan et al., 2017; Mirzasoleiman et al., 2018; Feldman et al., 2018). Elenberg et al. (2017) provide a streaming algorithm with a constant factor approximation for a generalized notion of submodular objective functions, called weak submodularity. In addition, a few other works study the streaming submodular maximization over sliding windows (Chen et al., 2016; Epasto et al., 2017).

To scale to very large datasets, several solutions to the problem of submodular maximization have been proposed in recent years (Mirzasoleiman et al., 2015, 2016a; Feldman et al., 2017; Badanidiyuru and Vondrák, 2014; Mitrovic et al., 2017a). Mirzasoleiman et al. (2015) proposed the first linear-time algorithm for maximizing a monotone submodular function subject to a cardinality constraint that achieves a $(1 - 1/e - \epsilon)$-approximation. Buchbinder et al. (2017) extended these results to non-monotone submodular functions.

Another line of work investigates the problem of scalable submodular maximization in the MapReduce setting where the data is split amongst several machines (Kumar et al., 2013; Mirzasoleiman et al., 2016b; Barbosa et al., 2015; Mirrokni and Zadimoghaddam, 2015; Mirzasoleiman et al., 2016c; Barbosa et al., 2016; Liu and Vondrák, 2018). Each machine runs a centralized algorithm on its data and passes the result to a central machine. Then, the central machine outputs the final answer. Since each machine runs a variant of the greedy algorithm, the adaptivity of all these approaches is linear in k, i.e., it is $\Omega(n)$ in the worst-case.

Practical concerns of scalability have motivated studying the adaptivity of submodular maximization algorithms. Balkanski and Singer (2018) showed that no algorithm can obtain a constant factor approximation in $o(\log n)$ adaptive rounds for monotone submodular maximization subject to a cardinality constraint. They introduced the first constant factor approximation algorithm for submodular maximization with logarithmic adaptive rounds. Their algorithm, in $O(\log n)$ adaptive rounds, finds a solution with an approximation arbitrarily close to $1/3$. These bounds were recently improved by $(1 - 1/e - \epsilon)$-approximation algorithm with $O(\log(n)/\mathrm{poly}(\epsilon))$ adaptivity (Fahrbach et al.,

2019; Balkanski et al., 2019a; Ene and Nguyen, 2019). More recently and independently, Balkanski et al. (2019b) and Chekuri and Quanrud (2019) studied the additivity of submodular maximization under a matroid constraint. In addition, Balkanski et al. (2018) proposed an algorithm for maximizing a non-monotone submodular function with cardinality constraint k whose approximation factor is arbitrarily close to $1/(2e)$ in $O(\log^2 n)$ adaptive rounds. Fahrbach et al. (2018) improved the adaptive complexity of this problem to $O(\log(n))$. Chen et al. (2018) considered the unconstrained submodular maximization problem and proposed the first algorithm that achieves the optimal approximation guarantee in a constant number of adaptive rounds.

Contributions The main contributions in this section are:

- We introduce Sieve-Streaming++ which is the first streaming algorithm with optimal approximation factor and memory complexity. Note that our optimality result for the approximation factor is under the natural assumption that the Oracle is allowed to make queries only over the feasible sets of cardinality at most k.
- We design an algorithm for a hybrid model of submodular maximization, where it enjoys a near-optimal adaptive complexity and it still guarantees both optimal approximation factor and memory complexity. We also prove that our algorithm has a very low communication cost in a multi-source streaming setting.
- We use multi-source streams of data from Twitter and YouTube to compare our algorithms against state-of-the-art streaming approaches.
- We significantly improve the memory complexity for several important problems in the submodular maximization literature by applying the main idea of Sieve-Streaming++ (see Appendix D.1).

4.2.3 Streaming Submodular Maximization

In this section, we propose an algorithm called Sieve-Streaming++ that has the optimal $1/2$-approximation factor and memory complexity $O(k)$. Our algorithm is designed based on the Sieve-Streaming algorithm (Badanidiyuru et al., 2014).

The general idea behind Sieve-Streaming is that choosing elements with marginal gain at least $\tau^* = \frac{\text{OPT}}{2k}$ from a data stream returns a set S with an objective value of at least $f(S) \geq \frac{\text{OPT}}{2}$. The main problem with this primary idea is that the value of OPT is not known. Badanidiyuru et al.

(2014) pointed out that, from the submodularity of f, we can trivially deduce $\Delta_0 \leq \text{OPT} \leq k\Delta_0$ where Δ_0 is the largest value in the set $\{f(\{e\}) \mid e \in V\}$. It is also possible to find an accurate guess for OPT by dividing the range $[\Delta_0, k\Delta_0]$ into small intervals of $[\tau_i, \tau_{i+1})$. For this reason, it suffices to try $\log k$ different thresholds τ to obtain a close enough estimate of OPT. Furthermore, in a streaming setting, where we do not know the maximum value of singletons a priori, Badanidiyuru et al. (2014) showed it suffices to only consider the range $\Delta \leq \text{OPT} \leq 2k\Delta$, where Δ is the maximum value of singleton elements observed so far. The memory complexity of Sieve-Streaming is $O(k \log k/\epsilon)$ because there are $O(\log k/\epsilon)$ different thresholds and, for each one, we could keep at most k elements.

4.2.4 The Sieve-Streaming++ Algorithm

In the rest of this section, we show that with a novel modification to Sieve-Streaming it is possible to significantly reduce the memory complexity of the streaming algorithm.

Our main observation is that in the process of guessing OPT, the previous algorithm uses Δ as a lower bound for OPT; but as new elements are added to sets S_τ, it is possible to get better and better estimates of a lower bound on OPT. More specifically, we have $\text{OPT} \geq \texttt{LB} \triangleq \max_\tau f(S_\tau)$ and as a result, there is no need to keep thresholds smaller than $\frac{\texttt{LB}}{2k}$. Also, for a threshold τ we can conclude that there is at most $\frac{\texttt{LB}}{\tau}$ elements in set S_τ. These two important observations allow us to get a geometrically decreasing upper bound on the number of items stored for each guess τ, which gives the asymptotically optimal memory complexity of $O(k)$. The details of our algorithm (Sieve-Streaming++) are described in Algorithm 9. Note that we represent the marginal gain of a set A to the set B with $f(A \mid B) = f(A \cup B) - f(B)$. Theorem 4.7 guarantees the performance of Sieve-Streaming++. See Appendix D.2 for the proof. Table 4 compares the state-of-the-art streaming algorithms based on approximation ratio, memory complexity and queries per element.

Theorem 4.7. *For a non-negative monotone submodular function f subject to a cardinality constraint k,* Sieve-Streaming++ *returns a solution S such that (i) $f(S) \geq (1/2-\epsilon)\cdot\max_{A\subseteq V,|A|\leq k} f(A)$, (ii) memory complexity is $O(k/\epsilon)$, and (iii) number of queries is $O(\log(k)/\epsilon)$ per each element.*

Algorithm 9 SIEVE-STREAMING++

Input: Submodular function f, data stream V, cardinality constraint k and error term ϵ

$\tau_{\min} \leftarrow 0$, $\Delta \leftarrow 0$ and $\texttt{LB} \leftarrow 0$
while there is an incoming item e from V **do**
  $\Delta \leftarrow \max\{\Delta, f(\{e\})\}$
  $\tau_{\min} = \frac{\max(\texttt{LB},\Delta)}{2k}$
  Discard all sets S_τ with $\tau < \tau_{\min}$
  for $\tau \in \{(1+\epsilon)^i | {}^{\tau_{\min}}/_{(1+\epsilon)} \leq (1+\epsilon)^i \leq \Delta\}$ **do**
    if τ is a new threshold **then** $S_\tau \leftarrow \emptyset$
    if $|S_\tau| < k$ and $f(\{e\} \mid S_\tau) \geq \tau$ **then**
      $S_\tau \leftarrow S_\tau \cup \{e\}$ and $\texttt{LB} \leftarrow \max\{\texttt{LB}, f(S_\tau)\}$
return $\arg\max_{S_\tau} f(S_\tau)$

Algorithm	Approx. Ratio	Memory	Queries per Element
PREEMPTION-STREAMING	$1/4$	$O(k)$	$O(k)$
SIEVE-STREAMING	$1/2 - \epsilon$	$O({}^{k \log(k)}/_{\epsilon})$	$O({}^{\log(k)}/_{\epsilon})$
SIEVE-STREAMING++	$1/2 - \epsilon$	$O({}^{k}/_{\epsilon})$	$O({}^{\log(k)}/_{\epsilon})$

Table 4: Streaming algorithms for non-negative and monotone submodular maximization subject to a cardinality constraint k. PREEMPTION-STREAMING is due to Buchbinder et al. (2015), SIEVE-STREAMING is due to Badanidiyuru et al. (2014), and SIEVE-STREAMING++ is our contribution.

4.2.5 The Batch-Sieve-Streaming++ Algorithm

The SIEVE-STREAMING++ algorithm, for each incoming element of the stream, requires at least one query to the Oracle which increases its adaptive complexity to $\Omega(n)$. Since the adaptivity of an algorithm has a significant impact on its ability to be executed in parallel, there is a dire need to implement streaming algorithms with low adaptivity. To address this concern, our proposal is to first buffer a fraction of the data stream and then, through a parallel threshold filtering procedure, reduce the adaptive complexity, thus substantially lower the running time. Our results show that a small buffer memory can significantly parallelize streaming submodular maximization.

One natural idea for parallelization is to iteratively perform the following two steps: (i) for a threshold τ, in one adaptive round, compute the marginal gain of elements to set S_τ and discard those with a gain less than τ, and (ii) pick one of the remaining items with a good marginal gain and add it to S_τ. This process is repeated at most k times. We refer to this algorithm as SAMPLE-ONE-STREAMING and we will use it as a baseline in Section 4.2.9.

Although this method gives a $1/2 - \epsilon$ approximation factor, the adaptive complexity of this algorithm is $\Omega(k)$ which is still prohibitive in the worst case. For this reason, we introduce a hybrid

algorithm called Batch-Sieve-Streaming++. This algorithm enjoys two important properties: (i) the number of adaptive rounds is near-optimal, and (ii) it has an optimal memory complexity (by adopting an idea similar to Sieve-Streaming++). Next, we explain Batch-Sieve-Streaming++ (Algorithm 10) in detail.

First, we assume that the machine has a buffer $\mathcal{B}$ that can store at most $\texttt{B}$ elements. For a data stream V, whenever `Threshold` fraction of the buffer is full, the optimization process begins. The purpose of `Threshold` is to empty the buffer before it gets completely full and to avoid losing arriving elements. Similar to the other sieve streaming methods, Batch-Sieve-Streaming++ requires us to guess the value of $\tau^* = \frac{\text{OPT}}{2k}$. For each guess τ, Batch-Sieve-Streaming++ uses Threshold-Sampling (Algorithm 11) as a subroutine. Threshold-Sampling iteratively picks random batches of elements T. If their average marginal gain to the set of picked elements S_τ is at least $(1-\epsilon)\tau$ it adds that batch to S_τ. Otherwise, all elements with marginal gain less than τ to the set S_τ are filtered out. Threshold-Sampling repeats this process until the buffer is empty or $|S_\tau| = k$.

Note that in Algorithm 10, we define the function f_S as $f_S(A) = f(A \mid S)$, which calculates the marginal gain of adding a set A to S. It is straightforward to show that if f is a non-negative and monotone submodular function, then f_S is also non-negative and monotone submodular.

The adaptive complexity of Batch-Sieve-Streaming++ is the number of times its buffer gets full (which is at most $N/\texttt{B}$) multiplied by the adaptive complexity of Threshold-Sampling. The reason for the low adaptive complexity of Threshold-Sampling is quite subtle. In Line 3 of Algorithm 11, with a non-negligible probability, a constant fraction of items is discarded from the buffer. Thus, the while loop continues for at most $O(\log \texttt{B})$ steps. Since we increase the batch size by a constant factor of $(1+\epsilon)$ each time, the for loops within each while loop will run at most $O(\log(k)/\epsilon)$ times. Therefore, the total adaptive complexity of Batch-Sieve-Streaming++ is $O(\frac{N \log(\texttt{B}) \log(k)}{\texttt{B}\epsilon})$ Note that when $|S| < 1/\epsilon$, multiplying the size by $(1+\epsilon)$ would increase it less than one, so we increase the batch size one by one for the first loop in Lines 4–10 of Algorithm 11. Theorem 4.8 guarantees the performance of Batch-Sieve-Streaming++. See Appendix D.3 for the proof.

Theorem 4.8. *For a non-negative monotone submodular function f subject to a cardinality constraint k, define N to be the total number of elements in the stream, $\texttt{B}$ to be the buffer size and $\epsilon < 1/3$ to be a constant. For* Batch-Sieve-Streaming++ *we have: (i) the approximation factor is $1/2 - 3\epsilon/2$, (ii) the memory complexity is $O(\texttt{B} + k/\epsilon)$, and (iii) the expected adaptive complexity is $O(\frac{N \log(\texttt{B}) \log(k)}{\texttt{B}\epsilon})$.*

Algorithm 10 BATCH-SIEVE-STREAMING++

Input: Stream of data V, submodular set function f, cardinality constraint k, buffer $\mathcal{B}$ with a memory `B`, `Threshold`, and error term ϵ.

$\Delta \leftarrow 0, \tau_{\min} \leftarrow 0, \texttt{LB} \leftarrow 0$ and $\mathcal{B} \leftarrow \emptyset$
while there is an incoming element e from V **do**
  Add e to $\mathcal{B}$
  if the buffer $\mathcal{B}$ is `Threshold` percent full **then**
    $\Delta \leftarrow \max\{\Delta, \max_{e \in \mathcal{B}} f(e)\}$, $\tau_{\min} = \frac{\max(\texttt{LB}, \Delta)}{2k(1+\epsilon)}$ and discard all sets S_τ with $\tau < \tau_{\min}$
    for $\tau \in \{(1+\epsilon)^i | \tau_{\min} \leq (1+\epsilon)^i \leq \Delta\}$ **do**
      If τ is a new threshold then assign a new set S_τ to it, i.e., $S_\tau \leftarrow \emptyset$
      if $|S_\tau| < k$ **then**
        $T \leftarrow$ THRESHOLD-SAMPLING$(f_{S_\tau}, \mathcal{B}, k - |S_\tau|, \tau, \epsilon)$
        $S_\tau \leftarrow S_\tau \cup T$
    $\texttt{LB} = \max_{S_\tau} f(S_\tau)$ and $\mathcal{B} \leftarrow \emptyset$
return $\arg\max_{S_\tau} f(S_\tau)$

Algorithm 11 THRESHOLD-SAMPLING

Input: Submodular set function f, set of buffered items $\mathcal{B}$, cardinality constraint k, threshold τ, and error term ϵ

$S \leftarrow \emptyset$
while $|\mathcal{B}| > 0$ and $|S| < k$ **do**
  update $\mathcal{B} \leftarrow \{x \in \mathcal{B} : f(\{x\} \mid S) \geq \tau\}$ and filter out the rest
  for $i = 1$ to $\lceil \frac{1}{\epsilon} \rceil$ **do**
    Sample x uniformly at random from $\mathcal{B} \setminus S$
    if $f(\{x\}|S) \leq (1-\epsilon)\tau$ **then**
      break and go to Line 2
    else
      $S \leftarrow S \cup \{x\}$
      if $|S| = k$ **then return** S
  for $i = \lfloor \log_{1+\epsilon}(1/\epsilon) \rfloor$ to $\lceil \log_{1+\epsilon} k \rceil - 1$ **do**
    $t \leftarrow \min\{\lfloor (1+\epsilon)^{i+1} - (1+\epsilon)^i \rfloor, |\mathcal{B} \setminus S|, k - |S|\}$
    Sample a random set T of size t from $\mathcal{B} \setminus S$
    if $|S \cup T| = k$ **then return** $S \cup T$
    if $\frac{f(T \mid S)}{|T|} \leq (1-\epsilon)\tau$ **then**
      $S \leftarrow S \cup T$ and **break**
    else
      $S \leftarrow S \cup T$
return S

Remark It is important to note that THRESHOLD-SAMPLING is inspired by recent progress for maximizing submodular functions with low adaptivity (Fahrbach et al., 2019; Balkanski et al., 2019a; Ene and Nguyen, 2019) but it uses a few new ideas to adapt the result to our setting. Indeed, if we had used the sampling routines from these previous works, it was even possible to slightly improve the adaptivity of the hybrid model. The main issue with these methods is that their adaptivity heavily depends on evaluating many random subsets of the ground set in each round. As it is discussed in

the next section, we are interested in algorithms that are efficient in the multi-source setting. In that scenario, the data is distributed among several machines, so existing sampling methods dramatically increases the communication cost of our hybrid algorithm.

4.2.6 Multi-Source Data Streams

In general, the important aspects to consider for a single source streaming algorithm are approximation factor, memory complexity, and adaptivity. In the multi-source setting, the communication cost of an algorithm also plays an important role. While the main ideas of Sieve-Streaming++ give us an optimal approximation factor and memory complexity, there is always a trade-off between adaptive complexity and communication cost in any threshold sampling procedure.

As we discussed before, existing submodular maximization algorithms with low adaptivity need to evaluate the utility of random subsets several times to guarantee the marginal gain of sampled items. Consequently, this incurs high communication cost. In this section, we explain how Batch-Sieve-Streaming++ can be generalized to the multi-source scenario with both low adaptivity and low communication cost.

We assume elements arrive from m different data streams and for each stream the elements are placed in a separate machine with a buffer $\mathcal{B}_i$. When the buffer memory of at least one of these m machines is `Threshold`% full, the process of batch insertion and filtering begins. The only necessary change to Batch-Sieve-Streaming++ is to use a parallelized version of Threshold-Sampling with inputs from $\{\mathcal{B}_i\}$. In this generalization, Lines 5 and 13 of Algorithm 11 are executed in a distributed way where the goal is to perform the random sampling procedure from the buffer memory of all machines. Indeed, in order to pick a batch of t random items, the central coordinator asks each machine to send a pre-decided number of items. Note that the set of picked elements S_τ for each threshold τ is shared among all machines. And therefore the filtering step at Line 3 of Algorithm 11 can be done independently for each stream in only one adaptive round. Our algorithm is shown pictorially in Figure 10.

Theorem 4.9 guarantees the communication cost of Batch-Sieve-Streaming++ in the multi-source setting. See Appendix D.4 for the proof. Notably, the communication cost of our algorithm is independent of the buffer size `B` and the total number of elements N.

Theorem 4.9. *For a non-negative and monotone submodular function f in a multi-source streaming*

setting subject to a cardinality constraint k, define Δ_0 as the largest singleton value when for the first time a buffer gets full, and $= \frac{\text{OPT}}{\Delta_0}$. The total communication cost of Batch-Sieve-Streaming++ *is $O(\frac{k \log}{\epsilon^2})$.*

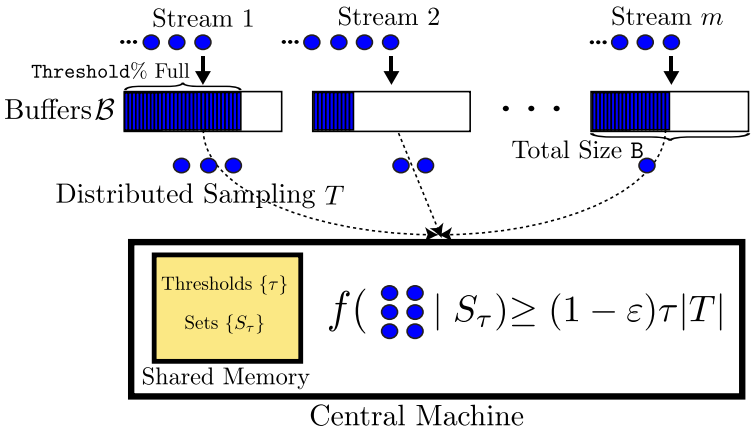


Figure 10: The schematic representation of our proposed hybrid algorithm: there are m simultaneous streams where data from each stream is buffered separately. When a buffer is `Threshold`% full, a central machine starts the sampling process. The thresholds $\{\tau\}$ and sets $\{S_\tau\}$ are stored in a shared memory. First, for each threshold τ, all elements with marginal gain less than τ are discarded from the buffers. Then the central machine randomly samples t items T (with geometrically increasing values of t) from the buffers of all streams and adds them to set S_τ if their average marginal gain is at least $(1-\epsilon)\tau$. The sampling procedure stops when the average value of randomly picked items is not good enough. These iterative steps are performed until k items are picked or the buffer memories of all machines are emptied.

4.2.7 Experiments Introduction

In these experiments, we have three main goals:

1. For the single-source streaming scenario, we want to demonstrate the memory efficiency of Sieve-Streaming++ relative to Sieve-Streaming.

2. For the multi-source setting, we want to showcase how Batch-Sieve-Streaming++ requires the fewest adaptive rounds amongst algorithms with optimal communication costs.

3. Lastly, we want to illustrate how a simple variation of Batch-Sieve-Streaming++ can trade off communication cost for adaptivity, thus allowing the user to find the best balance for their particular problem.

These experiments will be run on a Twitter stream summarization task and a YouTube Video summarization task, as described next.

Twitter Stream Summarization In this application, we want to produce real-time summaries for Twitter feeds. As of January 2019, six of the top fifty Twitter accounts (also known as "handles") are dedicated primarily to news reporting. Each of these handles has over thirty million followers, and there are many other news handles with tens of millions of followers as well. Naturally, such accounts commonly share the same stories. Whether we want to provide a periodic synopsis of major events or simply to reduce the clutter in a user's feed, it would be very valuable if we could produce a succinct summary that still relays all the important information.

To collect the data, we scraped recent tweets from 30 different popular news accounts, giving us a total of 42,104 unique tweets. In the multi-source experiments, we assume that each machine is scraping one page of tweets, so we have 30 different streams to consider.

We want to define a submodular function that covers the important stories of the day without redundancy. To this end, we extract the keywords from each tweet and weight them proportionally to the number of retweets the post received. In order to encourage diversity in a selected set of tweets, we take the square root of the value assigned to each keyword. More formally, consider a function f defined over a ground set V of tweets. Each tweet $e \in V$ consists of a positive value val_e denoting its number of retweets and a set of ℓ_e keywords $W_e = \{w_{e,1}, \cdots, w_{e,\ell_e}\}$ from a general set of keywords $\mathcal{W}$. The score of a word $w \in W_e$ for a tweet e is defined by $\text{score}(w, e) = \text{val}_e$. If $w \notin W_e$, we define $\text{score}(w, e) = 0$. For a set $S \subseteq V$ of tweets, the function f is defined as follows:

$$f(S) = \sum_{w \in \mathcal{W}} \sqrt{\sum_{e \in S} \text{score}(w, e)}.$$

Figure 11 gives an example and Appendix D.5 gives a proof of submodularity for this function.

YouTube Video Summarization In this second task, we want to select representative frames from multiple simultaneous and related video feeds. In particular, we consider YouTube videos of New Year's Eve celebrations from ten different cities around the world. Although the cities are not all in the same time zone, in our multi-source experiments we assume that we have one machine processing each video simultaneously.

Using the first 30 seconds of each video, we train an autoencoder that compresses each frame into a 4-dimensional representative vector. Given a ground set V of such vectors, we define a matrix M such that $M_{ij} = e^{-\text{dist}(v_i, v_j)}$, where $\text{dist}(v_i, v_j)$ is the euclidean distance between vectors $v_i, v_j \in V$. Intuitively, M_{ij} encodes the similarity between the frames represented by v_i and v_j.

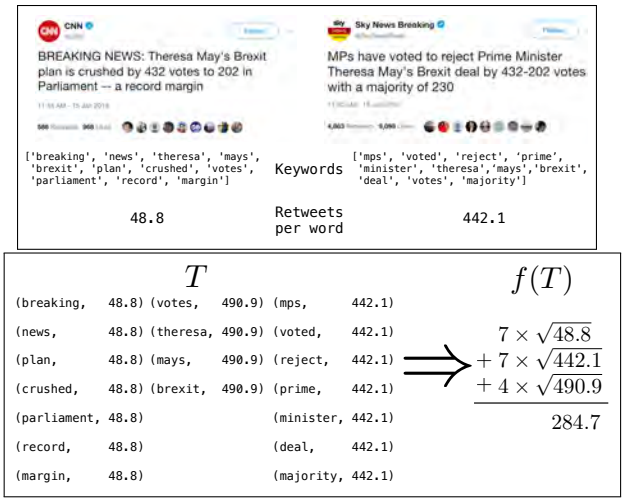


Figure 11: At the top, we show two tweets on the same subject from different accounts. We first extract the list of keywords, as well as the number of retweets per word. We combine these into a single list T of (keyword, score) pairs and then pass this list through our submodular function f.

Figure 12: Eight representative frames chosen by Batch-Sieve-Streaming++ from ten different simultaneous feeds of New Year's Eve fireworks from around the world.

The utility of a set $S \subseteq V$ is defined as a non-negative monotone submodular objective $f(S) = \log\det(I + \alpha M_S)$, where I is the identity matrix, $\alpha > 0$ and M_S is the principal sub-matrix of M indexed by S (Herbrich et al., 2003). Informally, this function is meant to measure the diversity of the vectors in S. Figure 12 shows the representative images selected by our Batch-Sieve-Streaming++ algorithm for $k = 8$.

4.2.8 Single-Source Experiments

In this section, we want to emphasize the power of Sieve-Streaming++ in the single-source streaming scenario. As discussed earlier, the two existing standard approaches for monotone k-cardinality submodular streaming are Sieve-Streaming and Preemption-Streaming.

As mentioned in Section 4.2.3, Sieve-Streaming++ theoretically has the best properties of both of these existing baselines, with optimal memory complexity and the optimal approximation guarantee. Figures 13a through 13d show the performance of these three algorithms on the YouTube task and confirm that this holds in practice as well.

For the purposes of this test, we simply combined the different video feeds into one single stream. We see that the memory required by Sieve-Streaming++ is much smaller than the memory required by Sieve-Streaming, but it still achieves the exact same utility. Furthermore, the memory requirement of Sieve-Streaming++ is within a constant factor of Preemption-Streaming, while its utility is much better. The Twitter experiment gives similar results so those graphs are deferred to Appendix D.6.

4.2.9 Multi-Source Experiments

Once we move into the multi-source setting, the communication cost of algorithms becomes a key concern also. In this section, we compare the performance of algorithms in terms of utility and adaptivity where their communication cost is optimal.

Our first baseline is a trivial extension of Sieve-Streaming++. The multi-source extension for this algorithm essentially functions by locally computing the marginal gain of each incoming element, and only communicating it to the central machine if the marginal gain is above the desired threshold. However, as mentioned at the beginning of Section 4.2.4, this algorithm requires $\Omega(n)$ adaptive rounds. Our second baseline is Sample-One-Streaming, which was described in Section 4.2.5.

Figures 13e and 13f show the effect of the buffer size `B` on the performance of these algorithms for the Twitter task. The main observation is that Batch-Sieve-Streaming++ can achieve roughly the same utility as the two baselines with many fewer adaptive rounds. Note that the number of adaptive rounds is shown in log scale.

Figures 13g and 13h show how these numbers vary with ϵ. Again, the utilities of the three baselines

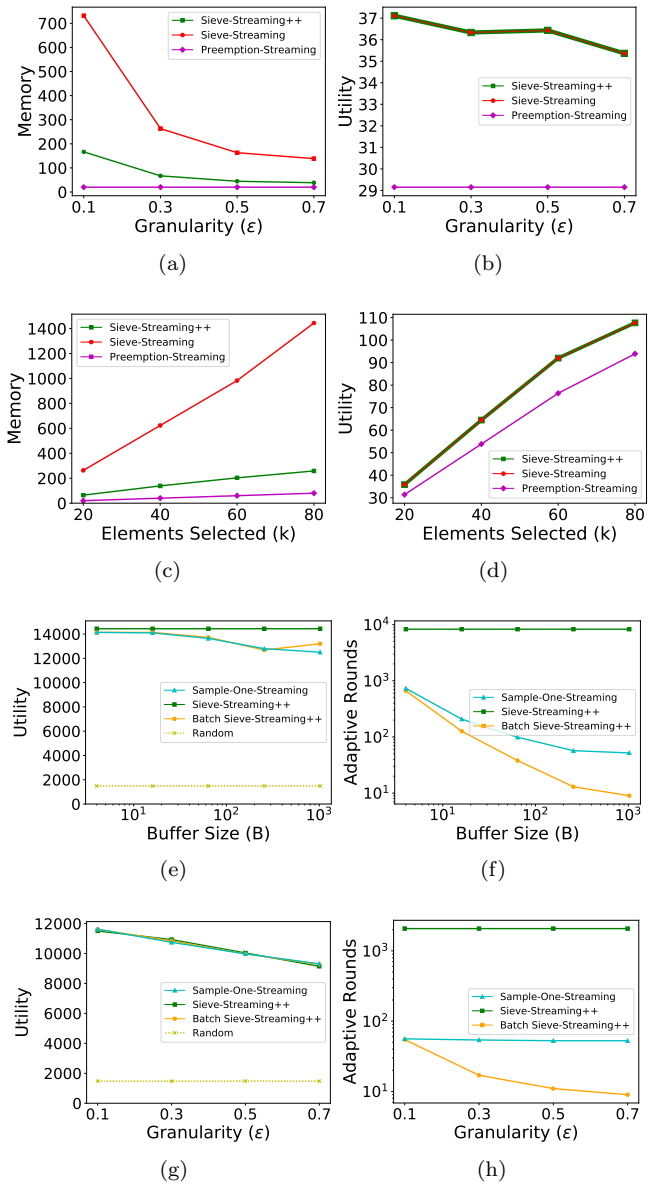


Figure 13: Graphs (a) to (d) show how the memory and utility of various single-source streaming algorithms vary with the cardinality k and granularity ϵ. Note that the utility of SIEVE-STREAMING++ and SIEVE-STREAMING exactly overlap. In (a) and (b) we use k = 20, while in (c) and (d) we use $\epsilon = 0.3$. Graphs (e) to (h) show how the utility and adaptivity of various multi-source streaming algorithms vary with the buffer size B and the granularity ϵ. Unless they are being varied on the x-axis, we set $\epsilon = 0.7$, B = 100, and $k = 50$.

are similar. We also see that increasing ϵ results in a large drop in the number of adaptive rounds for BATCH-SIEVE-STREAMING++, but not for SAMPLE-ONE-STREAMING. Appendix D.6 gives some additional graphs, as well as the results for the YouTube dataset.

4.2.10 Trade-off Between Communication and Adaptivity

In the multi-source setting, there is a natural exchange between communication cost and adaptivity. Intuitively, the idea is that if we sample items more aggressively (which translates into higher communication cost), a set S of k items is generally picked faster, thus it reduces the adaptivity. In the real world, the preference for one or the other can depend on a wide variety of factors ranging from resource constraints to the requirements of the particular problem.

In Threshold-Sampling, we ensure the optimal communication performance by sampling $t_i = \lceil (1+\epsilon)^{i+1} - (1+\epsilon)^i \rceil$ items in each step of the for loop. Instead, to reduce the adaptivity by a factor of $\log(k)$, we could sample all the required k items in a single step. Thus, in one adaptive round we mimic the two for loops of Threshold-Sampling. Doing this in each call to Algorithm 11 would reduce the expected adaptive complexity of Threshold-Sampling to the optimal $\log(\mathtt{B})$, but dramatically increase the communication cost to $O(k \log \mathtt{B})$.

In order to trade off between communication and adaptivity, we can instead sample $t_i^R = \lceil (1+\epsilon)^{i+R} - (1+\epsilon)^i \rceil$ elements to perform R consecutive adaptive rounds in only one round. However, to maintain the same chance of a successful sampling, we still need to check the marginal gain. Finally, we pick a batch of the largest size t_i^j such that the average marginal gain of the first t_i^{j-1} items is above the desired threshold. Then we just add just this subset to S_τ, meaning we have wasted $\lceil (1+\epsilon)^{i+R} - (1+\epsilon)^{i+j} \rceil$ communication.

Scatter plots of Figure 14 shows how the number of adaptive rounds varies with the communication cost. Each individual dot represents a single run of the algorithm on a different subset of the data. The different colors cluster the dots into groups based on the value of R that we used in that run. Note that the parameter R controls the communication cost.

The plot on the left comes from the Twitter experiment, while the plot on the right comes from the YouTube experiment. Although the shapes of the clusters are different in the two experiments, we see that increasing R increases the communication cost, but also decreases the number of adaptive rounds, as expected.

4.2.11 Conclusion

In this section, we studied the problem of maximizing a non-negative submodular function over a multi-source stream of data subject to a cardinality constraint k. We first proposed Sieve-Streaming++ with the optimum approximation factor and memory complexity for a single stream

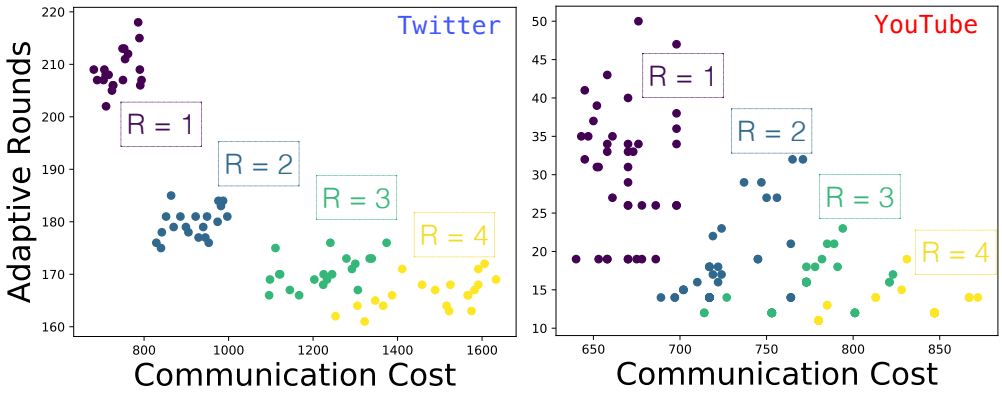


Figure 14: Scatter plots showing how we can lower the number of adaptive rounds by increasing communication. Each dot is the result of a single run of the algorithm and the colored clusters represent a particular setting for R.

of data. Build upon this idea, we designed an algorithm for multi-source streaming setting with a $1/2$ approximation factor, $O(k)$ memory complexity, a very low communication cost, and near-optimal adaptivity. We evaluated the performance of our algorithms on two real-world data sets of multi-source tweet streams and video streams. Furthermore, by using the main idea of Sieve-Streaming++, we significantly improved the memory complexity of several important submodular maximization problems.

5. Sequence Submodularity

As a motivating example, consider the problem of recommending movies to a user. A recommendation system could determine that the user might be interested in The Lord of the Rings franchise. However, if the model does not consider the order of the movies it recommends, the user may watch The Return of the King first and The Fellowship of the Ring last, which is likely to lead make the user totally unsatisfied with an otherwise excellent recommendation.

Although there has been a large volume of work devoted to the study of submodular functions in recent years, the vast majority of this work has been focused on algorithms that output sets, not sequences. However, in many settings, the order in which we output items can be just as important as the items themselves. In Section 5.1, we more formally introduce the concept of sequence submodularity and present our contributions to this new subfield of submodularity.

Building on the example of recommending movies, many successful recommender systems not only consider sequences, but also gather feedback from the user and then use this feedback to improve

further recommendations. Not only do sequences already pose a dauntingly large search space, but if we want to add such a notion of adaptivity, we must explicitly take into account past observations, as well as the uncertainty of future outcomes. In Section 5.2, we introduce the first framework for adaptive sequence submodularity.

5.1 Submodularity on Hypergraphs: From Sets to Sequences

5.1.1 Introduction to Sequence Submodularity

As mentioned earlier, the vast majority of existing results in the literature for submodularity are limited to the scenario where we wish to output sets, not sequences. Alaei and Malekian (2010) and Zhang et al. (2016) consider functions they call string- or sequence-submodular, but it is in a different context. Li and Milenkovic (2017) look at a combination of submodularity and hypergraphs, but it is specifically within the context of hypergraph clustering.

Tschiatschek et al. (2017) were the first to define sequence submodularity in the context that we follow in this dissertation. In particular, we use directed graphs where the items are encoded as vertices and the edges between these vertices encode the additional value of selecting items in a particular order. The only known theoretical result for this setting is limited to the case where the underlying graph is a directed acyclic graph (Tschiatschek et al., 2017). Considering sequences instead of sets causes an exponential increase in the size of the search space, but it allows for much more expressive models.

Recall that the goal is to select a **sequence** of items that will maximize some given objective function. To generalize the problem description, we will refer to items as vertices from now on.

Let $V = \{v_1, v_2, \ldots, v_n\}$ be the set of n vertices (items) we can pick from. A set of edges E encodes the fact that there is additional value in picking certain vertices in a certain order. More specifically, an edge $e_{ij} = (v_i, v_j)$ encodes the fact that there is additional utility in selecting v_j after v_i has already been chosen. Self-loops (i.e., edges that begin and end at the same vertex) encode the fact that there is some individual utility in selecting a vertex.

In general, our input consists of a directed graph $G = (V, E)$, a non-negative monotone submodular set function $h\colon 2^E \to \mathbb{R}_{\geq 0}$, and a parameter k. The objective is to output a non-repeating sequence σ of k unique nodes that maximizes the objective function:

$$f(\sigma) = h\big(E(\sigma)\big)$$

where

$$E(\sigma) = \big\{(\sigma_i, \sigma_j) \mid (\sigma_i, \sigma_j) \in E, i \leq j\big\}$$

We say that $E(\sigma)$ is the set of edges induced by the sequence σ. It is important to note that the function h is a submodular set function over the edges, not over the vertices. Furthermore, the objective function f is neither a set function, nor is it necessarily submodular on the vertices.

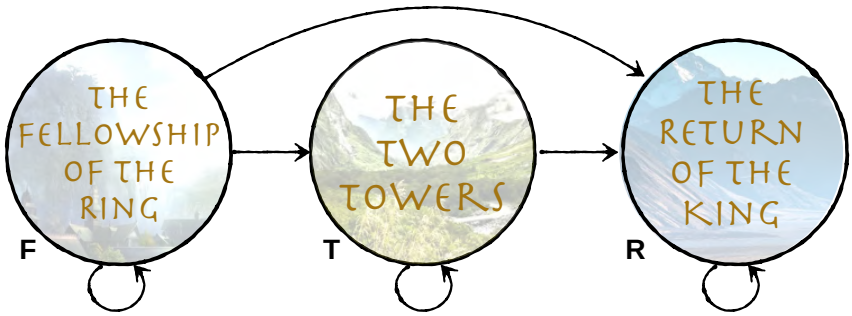


Figure 15: Graph for The Lord of the Rings franchise. The self-loops encode the fact that each movie has some individual value. The edges encode the fact that there is additional utility in watching the movies in the correct order. Notice that the utility of watching The Return of the King after having already seen both The Fellowship of the Ring and The Two Towers is higher than the utility of watching The Return of the King after having seen just one of the two.

For example, consider the graph in Figure 15, and let $h\big(E(\sigma)\big) = |E(\sigma)|$. That is, the value of a sequence is simply the number of edges induced by that sequence. Consider the sequence $\sigma_A = (F)$ where the user has watched only The Fellowship of the Ring, the sequence $\sigma_B = (T)$ where the user watched only The Two Towers, and the sequence $\sigma_C = (F, T)$ where the user watched The Fellowship of the Ring and then The Two Towers:

$$f(\sigma_A) = f(F) = h\big((F, F)\big) = 1$$
$$f(\sigma_B) = f(T) = h\big((T, T)\big) = 1$$
$$f(\sigma_C) = f(F, T) = h\big((F, F), (F, T), (T, T)\big) = 3$$

This example shows that although the marginal gain of the edges is non-increasing in the context of a growing set of edges (i.e., the function h is submodular on the edges), it is clear that the function f is *not* submodular on the vertices. In particular, the marginal gain of The Two Towers is larger

once the user has already viewed The Fellowship of the Ring.

Furthermore, just to fully clarify the concept of edges being induced by a sequence, consider the sequence $\sigma_D = (T, F)$ where the user watched The Two Towers and then The Fellowship of the Ring.

$f(\sigma_D) = f(T, F) = h\big((T, T), (F, F)\big) = 2$

Notice that although sequences σ_C and σ_D contain the same movies, the order of σ_D means that the edge (F, T) is not induced, and thus, the value of the sequence is lower.

Throughout this section, we use the notation $\Delta = \min\{d_{\text{in}}, d_{\text{out}}\}$, where $d_{\text{in}} = \max_{v \in V} d_{\text{in}}(v)$ and $d_{\text{out}} = \max_{v \in V} d_{\text{out}}(v)$. The previous work on our problem, due to Tschiatschek et al. (2017), presented an algorithm (OMegA) enjoying a $(1-e^{-\frac{1}{2\Delta}})$-approximation guarantee when the underlying graph G is a directed acyclic graph (except for self-loops).

Our Contributions We present two new algorithms: Sequence-Greedy and Hyper Sequence-Greedy, which also provably achieve constant factor approximations (when Δ is constant), but the guarantees hold for general graphs and hypergraphs, respectively. Although the example given in Figure 15 is indeed a directed acyclic graph, many real-world problems require a general graph or hypergraph.

We showcase the utility of our algorithms on real world applications in movie recommendation, online link prediction, and the design of course sequences for massive open online courses (MOOCs). Furthermore, we show that even when the underlying graph is a directed acyclic graph, our general graph algorithm performs comparably well. Our experiments also demonstrate the power of being able to utilize hypergraphs and hyperedges.

5.1.2 Theoretical Results for General Graphs

In this section, we present our first algorithm, Sequence-Greedy. Sequence-Greedy is essentially the same as the classical greedy algorithm, but instead of choosing the most valuable vertex at each step, it chooses the most valuable valid edge.

More specifically, we start off with an empty sequence σ. At each step, we define $\mathcal{E}$ to be the set of all edges whose end point is not already in σ. We then greedily select the edge $e_{ij} \in \mathcal{E}$ with

maximum marginal gain $h\big(e_{ij} \mid E(\sigma)\big)$, where

$$h\big(e_{ij} \mid E(\sigma)\big) = h\big(E(\sigma) \cup e_{ij}\big) - h\big(E(\sigma)\big) \ .$$

Recall that $e_{ij} = (v_i, v_j)$. That is, v_i is the start point of e_{ij} and v_j is the endpoint. If e_{ij} is a self-loop, then $j = i$ and we append the single vertex v_j to σ. Similarly, if $j \neq i$, but v_i is already in σ, then we still only append v_j. Finally, if e_{ij} has two distinct vertices and neither of them is already in the sequence, we append v_i and then v_j to σ. This description is summarized in pseudo-code in Algorithm 12.

Algorithm 12 Sequence-Greedy (Forward)

Input: Directed graph $G = (V, E)$
Monotone submodular function $h : 2^E \to \mathbb{R}$
Cardinality parameter k
Let $\sigma \leftarrow ()$
while $|\sigma| \leq k - 2$ **do**
    $\mathcal{E} = \{e_{ij} \in E \mid v_j \notin \sigma\}$ ▷ $e_{ij} = (v_i, v_j)$
    if $\mathcal{E} = \varnothing$ **then**
        break
    $e_{ij} = \arg\max_{e \in \mathcal{E}} h(e \mid E(\sigma))$
    if $v_j = v_i$ **or** $v_i \in \sigma$ **then**
        $\sigma = \sigma \oplus v_j$ ▷ $\oplus$ means concatenate
    else
        $\sigma = \sigma \oplus v_i \oplus v_j$
Return: σ

Theorem 5.1. *The approximation ratio of Algorithm 12 is at least* $\frac{1-e^{-(1-\frac{1}{k})}}{2d_{in}+1}$.

The proof for Theorem 5.1 is given in Appendix E.1. Notice that the approximation guarantee of Algorithm 12 depends on the maximum in-degree d_{in}. Intuitively, this is because Algorithm 12 builds σ by appending vertices to the end of the sequence. This means that each vertex we add to σ decreases the size of $\mathcal{E}$ by at most d_{in}.

However, one can easily modify Algorithm 12 to build σ backwards by *prepending* vertices to the start of the sequence at each step. More specifically, we redefine $\mathcal{E}$ to be the set of all edges whose *start point* is not already in σ. Again we greedily select the edge $e_{ij} \in \mathcal{E}$ that maximizes $h\big(e_{ij} \mid E(\sigma)\big)$. Now, if e_{ij} is a self-loop or v_j is already in σ, we prepend the single vertex v_i to the start of σ. Otherwise, if e_{ij} has two distinct vertices and neither of them is already in the sequence, we prepend v_j to σ first, and then prepend v_i (thus, maintaining the order). This description is summarized in

pseudo-code in Algorithm 13 with the main differences noted as comments.

Algorithm 13 Sequence-Greedy (Backward)

Input: Directed graph $G = (V, E)$
Monotone submodular function $h : 2^E \to \mathbb{R}$
Cardinality parameter k
Let $\sigma \leftarrow ()$
while $|\sigma| \leq k - 2$ **do**
    $\mathcal{E} = \{e_{ij} \in E \mid v_i \notin \sigma\}$. ▷ Note that $\mathcal{E}$ is defined differently than in Algorithm 12
    if $\mathcal{E} = \varnothing$ **then**
        break
    $e_{ij} = \arg\max_{e \in \mathcal{E}} h(e \mid E(\sigma))$
    if $v_i = v_j$ **or** $v_j \in \sigma$ **then**
        $\sigma = v_i \oplus \sigma$
    else
        $\sigma = v_i \oplus v_j \oplus \sigma$ ▷ vertices are appended to the beginning of σ
Return: σ

Algorithm 13 gives the same approximation ratio as Algorithm 12, but with a dependence on d_{out} instead of d_{in}. Thus, if we run both the forwards and backwards version of Sequence-Greedy and take the maximum, we get an approximation ratio that depends on $\Delta = \min\{d_{\text{in}}, d_{\text{out}}\}$. Furthermore, notice that the approximation ratio improves as k increases. Therefore, we can summarize the approximation ratio of Sequence-Greedy as follows.

Theorem 5.2. *As $k \to \infty$, the approximation ratio of Sequence-Greedy approaches $\frac{1-\frac{1}{e}}{2\Delta+1}$.*

This is comparable to the $(1 - e^{-\frac{1}{2\Delta}})$-approximation guarantee that is achieved by the existing algorithm OMegA, except that our guarantee is valid on general graphs, not just directed acyclic graphs.

In addition to this provable approximation ratio, Sequence-Greedy has the strong advantage of being computationally efficient. Both finding $\mathcal{E}$ and identifying the most valuable edge in $\mathcal{E}$ can be done in $O(m)$ time, where $m = |E|$. Thus, Sequence-Greedy runs in $O(km)$ time. This is faster than OMegA, which runs in $O(m\Delta k^2 \log k)$.

5.1.3 Extension to Hypergraphs

Extending our results to hypergraphs allows us to encode increasingly sophisticated models. For example, looking back on Figure 15, we see that the value of watching all three movies is just the

sum of the pairwise additional values. However, hyperedges allow us to encode the fact that there is even further utility in watching the entire franchise in order.

From this point on, we replace the directed graph G with a directed hypergraph $H = (V, E)$. Each edge $e \in E$ of this directed hypergraph is a non-empty non-repeating sequence of vertices from V. Let $V(e)$ be the set of vertices found in the hyperedge e. We assume that the intersection of a sequence and a set maintains the order of the sequence, which allows us to redefine $E(\sigma)$ as

$$E(\sigma) = \{e \in E \mid \sigma \cap V(e) = e\} \ .$$

Informally, $E(\sigma)$ contains an edge $e \in E$ if and only if all the vertices of e appear in σ in the proper order.

We also need to explain how the concept of in-degrees and out-degrees extends to hypergraphs. Self-loops contribute 1 to both the in-degree and the out-degree of that vertex. For all other edges $e \in E$ such that $v \in V(e)$, they will contribute 1 to $d_{\text{in}}(v)$ if v is not the first vertex of e, and 1 to $d_{\text{out}}(v)$ if v is not the last vertex of e. Finally, we define r as the maximum size of any edge in E. More formally, $r = \max_{e \in E} |e|$.

Aside from the above redefinition of $E(\sigma)$, there is no need to make other changes in the definition of the objective function f. Specifically, it is still defined as $f(\sigma) = h(E(\sigma))$, where $h \colon 2^E \to \mathbb{R}_{\geq 0}$ is a non-negative monotone submodular function.

Our algorithm for hypergraphs, Hyper Sequence-Greedy, is an extension of the original Sequence-Greedy. Again, we start off with an empty sequence σ. This time, at each step we define $\mathcal{E}$ to be the set of all hyperedges $e \in E$ such that $\sigma \cap V(e)$ is a prefix of e. The idea is that we can only select a hyperedge e if the vertices of e included in our sequence σ form a prefix of e, and they appear in σ in the right order. We then select the hyperedge $e^* \in \mathcal{E}$ that has the maximum marginal gain, and append the vertices of e^* (that are not already in our sequence) to σ without changing their order. This description is summarized in pseudo-code in Algorithm 14.

Theorem 5.3. *The approximation ratio of Algorithm 14 is at least* $\frac{1-e^{-(1-\frac{r}{k})}}{rd_{\text{in}}+1}$.

The proof for Theorem 5.3 is given in Appendix E.2. As with Sequence-Greedy, we can also run Hyper Sequence-Greedy backwards and take the maximum of the two results. In the backwards

Algorithm 14 Hyper Sequence-Greedy (Forward)

Input: Directed hypergraph $H = (V, E)$
Monotone submodular function $h : 2^E \to \mathbb{R}$
Cardinality parameter k
Let $\sigma \leftarrow ()$
while $|\sigma| \leq k - r$ **do**
    Let $\mathcal{E} = \{e \in E \mid \sigma \cap V(e) \text{ is a prefix of } e\}$
    if $\mathcal{E} = \varnothing$ **then**
        break
    $e^* = \arg\max_{e \in \mathcal{E}} h(e \mid E(\sigma))$
    for every $v \in e^*$ in order **do**
        if $v \notin \sigma$ **then**
            $\sigma = \sigma \oplus v$
Return: σ

version, we *prepend* the vertices to the start of the sequence and we can only select a hyperedge e if $V(e) \cap \sigma$ is a suffix of e. Once more, this improves the approximation ratio in the sense that the dependence on d_{in} is replaced with a dependence on $\Delta = \min\{d_{\text{in}}, d_{\text{out}}\}$. Additionally notice that, as before, our approximation ratio improves as k increases. Thus, we can summarize the performance guarantee of Hyper Sequence-Greedy as follows.

Theorem 5.4. *As $k \to \infty$, the approximation ratio of Hyper Sequence-Greedy approaches $\frac{1-\frac{1}{e}}{r\Delta+1}$.*

Remarks: One can observe that this hypergraph setting is a generalization of the previous directed graph setting. Specifically, Sequence-Greedy and the associated theory is a special case of Hyper Sequence-Greedy for $r = 2$. Furthermore, if $r = 1$ (i.e., our graph has only self-loops) then Hyper Sequence-Greedy is the same as the classical greedy algorithm.

We also note that while Algorithm 14 may select fewer than k vertices, the theoretical guarantees still hold. Furthermore, since we assume that h is monotone, we can safely select k vertices in practice every time. One simple heuristic for extending σ to k vertices is to only consider hyperedges with at most $k - |\sigma|$ vertices.

5.1.4 Movie Recommendation Application

In this application, we use the *Movielens 1M* dataset (Harper and Konstan, 2015) to recommend movies to users based on the films they have reviewed in the past. This dataset contains 1,000,209 anonymous, time-stamped ratings made by 6,040 users for 3,706 different movies. As in Tschiatschek et al. (2017), we do not want to predict a user's rating for a given movie, instead we want to predict

which movies the user will review next.

One issue with this dataset is that the distribution of the number of ratings per user (shown in Figure 16a) has a very long tail, with the most prolific reviewer having reviewed 2,314 movies. In order for our data to be representative of the general population, we remove all users who have rated fewer than 20 movies or more than 50 movies. We also remove all movies with fewer than 1,000 reviews. This leaves us with 67,757 ratings made by 2,047 users for 207 different movies.

We first group and sort all the reviews by user and time-stamp, so that each user i has an associated sequence σ^i of movies they have rated, where σ^i_j refers to the j^{th} movie that user i has reviewed. We use a 90/10 training/testing split of the data and 10-fold cross validation.

For each user i in the test set (D_{test}), we use their first 8 movies as a given starting sequence $S_i = \{\sigma^i_1 \dots \sigma^i_8\}$. We want to use S_i to select k movies that we think user i will review in the future. Therefore, for each user i, we build a hypergraph $H_i = (V, E_i)$, where $V = \{v_1, \dots, v_n\}$ is the set of all movies, and E_i is a set of hyperedges. Each hyperedge e_s has value p_s, where s is a movie sequence of length at most 3. Intuitively, p_s is the conditional probability of reviewing the last movie in s given that the rest of the movies in s have already been reviewed in the proper order.

Since we use empirical frequencies in the training data to calculate these conditional probabilities, we may run into the issue of overfitting to rare sequences. To avoid this, we add a parameter d to the denominator of our calculation of each edge value. This will increase the relative value for sequences that appear more often. In this experiment, we use $d = 20$.

More formally, define N_s to be the number of users in the training set (D_{train}) that have reviewed all the movies in the sequence s in the proper order. Also define s_l to be last element in s, and s' to be s with s_l removed. Now we can define the value of each edge e_s as follows:

$$p_s = \begin{cases} \dfrac{N_s}{N_{s'} + d} & s' \subseteq S_i \ , \\ p_{s'} \dfrac{N_s}{N_{s'} + d} & \text{otherwise} \ . \end{cases} \tag{6}$$

As mentioned above, the idea is that p_s represents the conditional probability of reviewing s_l given that all the movies in s' have already been reviewed in the proper order. If user i has not reviewed all the movies in s', then we scale down the value of that edge by $p_{s'}$ (i.e., the conditional probability of reviewing all the movies in s').

Note that if $s' = \varnothing$, then we define $N_{s'} = |D_{train}|$, thus ensuring that this definition also applies for self-loops. A small subgraph of a fully trained hypergraph is shown in Figure 16b.

We use a probabilistic coverage utility function as our non-negative monotone submodular function h. Mathematically,

$$h(E) = \sum_{v \in nodes(E)} \Big[1 - \prod_{s \in E | s_l = v} (1 - p_s)\Big]$$

We compare the performance of our algorithms, Sequence-Greedy and Hyper Sequence-Greedy, to the existing submodular sequence baseline (OMegA), as well as a naive baseline (Frequency), which just outputs the most popular movies that the user has not yet reviewed.

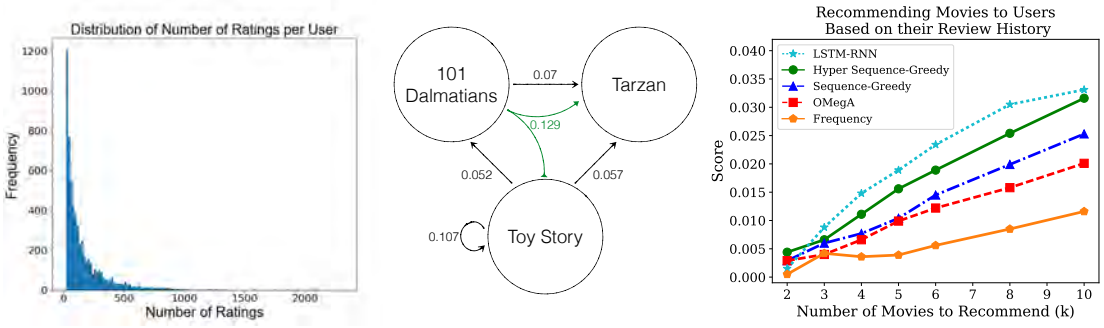
We also compare to a simple long short-term memory (LSTM) recurrent neural network (RNN). In addition to tuning parameters, we experimented with various frameworks such as training on uniform vs. variable-sized sequences. In the end, we obtained the best results when we trained the neural network on the first k movies of each σ^i, where the target is to predict the next k movies that the user i will review. In terms of the architecture, we use one layer of 512 LSTM nodes (with a dropout of 0.5) followed by a dense layer with a softmax activation that returns a 207×1 vector P, where entry P_i is the probability that movie i will be reviewed. For each k, we simply return the k highest values in P. As before, we used a 90/10 training/testing split with 10-fold cross validation.

With enough data, neural networks will likely significantly outperform our algorithms. However, with this comparison, we would like to show that in situations where data is relatively scarce, our algorithms are competitive with existing neural network frameworks.

To measure the accuracy of a prediction, we use a modified version of the Kendall tau distance (Kendall, 1938). First, for any sequence σ, we define $T(\sigma)$ to be the set of all ordered pairs in σ. For example, if $\sigma = \{1, 3, 2\}$, then $T(\sigma) = \big[(1, 3), (1, 2), (3, 2)\big]$.

Let P_i be our predicted sequence for the next k movies that user i will review, and let Q_i be the next k movies that user i actually reviewed. Then, we define the accuracy of the prediction P_i as follows.

$$\tau(P_i, Q_i) = \frac{|T(P_i) \cap T(Q_i)|}{|T(Q_i)|}$$



(a) (b) (c)

Figure 16: (a) Shows the long-tailed distribution of the number of ratings per user in the *Movielens 1M* dataset. (b) Shows a small subgraph of the overall hypergraph H that we train. For clarity, we only show edges with value $p_s > 0.05$, as defined in equation (6). We also highlight the size 3 hyperedge in green. (c) Shows the performance of our algorithms against existing baselines for various cardinalities k.

In other words, $\tau(P_i, Q_i)$ is the fraction of ordered pairs of the true answer Q_i that appear in our prediction P_i. Our experimental results in terms of this accuracy measure are summarized in Figure 16c.

These results showcase the power of using hypergraphs, as Hyper Sequence-Greedy consistently outperforms Sequence-Greedy. We also notice that Hyper Sequence-Greedy outperforms the score of the existing baseline OMegA by roughly 50%.

5.1.5 Online Link Prediction Application

In this application, we consider users who are searching through Wikipedia for some target article. Given a sequence of articles they have previously visited, we want to predict which link they will follow next. We use the Wikispeedia dataset (West et al., 2009), which consists of 51,138 completed search paths on a condensed version of Wikipedia that contains 4,604 articles and 119,882 links between them.

The setup for this problem is similar to that of section 5.1.4, so we will only go over the main differences. Again we will use a 90/10 training/testing split of the data with 10-fold cross validation.

For each training set D_{train}, we build the underlying hypergraph $H = (V, E)$. This time, V is the set of all articles, and E is a set of hyperedges e_s where p_s is the conditional probability of moving to article s_l given that the user had just visited s' in succession.

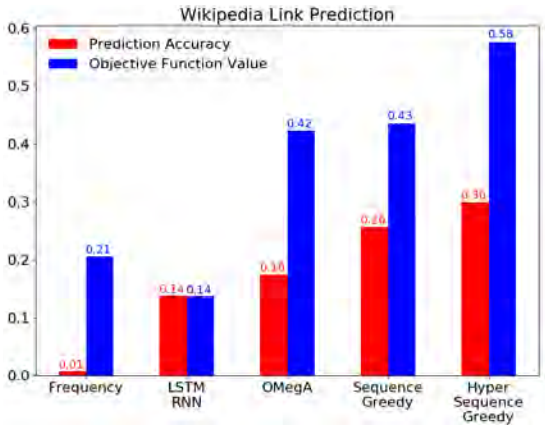


Figure 17: Given a sequence of articles a Wikipedia user has visited, we want to predict the next link they will click. This bar chart shows the prediction accuracy, as well as the objective function value, of various algorithms.

For each testing set D_{test}, we will use the last article in each completed path as the target, and the previous 3 articles as the given sequence. This means we will be able to use hyperedges of up to size 4. We employ the same probabilistic coverage function h and the same baseline comparisons as in Section 5.1.4. For this application, our neural network was most effective when we used a single layer of 32 LSTM nodes (with a dropout of 0.2). Our results are shown in Figure 17.

In this case, Hyper Sequence-Greedy exhibits the best performance. We see that the simple neural network implementation is outperformed by Hyper Sequence-Greedy as well as by some of the baselines. This is likely a result of the data in this experiment being more sparse. Although in this application we technically have more data than in the previous one, here we attempt to choose between 4,604 articles, rather than just 207 movies.

We also show the results that the various algorithms achieve when evaluated on our objective function $f(\sigma) = h\big(E(\sigma)\big)$. Asides from the LSTM-RNN, which doesn't consider the objective function at all, we see that the objective function values are relatively in line with the prediction accuracy. This demonstrates that the probabilistic coverage function was a good choice for the objective function.

5.1.6 Course Sequence Design Application

In this final application we want to use historical enrollment data in Massive Open Online Courses (MOOCs) to generate a sequence of courses that we think would be of interest to users. We use a publicly available dataset (Ho et al., 2014) that covers the first year of open online courses offered

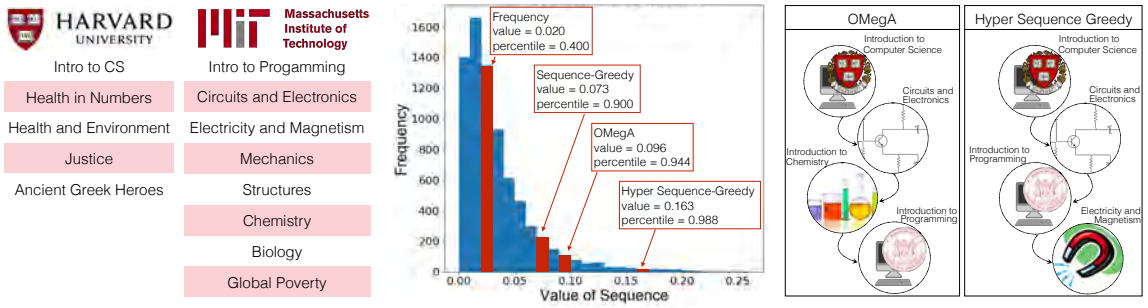
by edX. The dataset consists of 641,139 registrations from 476,532 unique users across 13 different online courses offered by Harvard and MIT. Amongst a plethora of other statistics, the data contains information on when each user first and last accessed each course, how many course chapters they accessed, and the grade they achieved if they were ultimately certified (i.e., fully completed) in the course.

One natural way to think about the value of a sequence of courses is in terms of prerequisites. That is, in what order should we offer courses to students in order to help them learn as much as possible. This model comes with a natural measure of success as well, which is the grade each student gets in each course. Unfortunately, out of the 476,532 unique users in this dataset only 180 were certified (and thus, received grades) in 3 or more courses. Furthermore, this dataset only contains 13 different courses (shown in Figure 18a), none of which are logical prerequisites for each other.

Instead, we can think about a sequence of courses being valuable if they will all be interesting to a user who registers for them. Similarly to the prerequisites model where the order of courses affects the user's grade, the order in which a user registers for courses should also affect their interest. In this dataset, we can measure interest by the percentage of the course that the user accessed. In particular, we say that if a user was interested in a course i if she accessed at least one-third of all the chapters for course i.

As always, we need to build the underlying hypergraph $H = (V, E)$ for each training set. In this case, V is the set of all courses and E is a set of hyperedges of form e_s, where s is a sequence of at most 3 courses and p_s is the probability that a user will be interested in s_l given that she previously showed interest in s' in the proper order. Recall that s_l is the last course in s, and s' is the sequence obtained from s after deleting s_l. As in section 5.1.4, we also use a parameter d to avoid overfitting to rare sequences. In this case we use $d = 100$. However, unlike Section 5.1.4, we are not making recommendations based on a user's history. Instead each algorithm will use the underlying hypergraph to build a single sequence σ. Since we are not starting with any given sequence, we can finally run Sequence-Greedy and Hyper Sequence-Greedy both forwards and backwards, and take the maximum of the two results.

Different users will naturally have different interests, so it is unreasonable to expect that any single sequence σ will work for all users. However, if σ is a "good" sequence, we could expect that users who start all the courses in σ in the correct order ultimately end up showing interest in those courses. Intuitively, the idea is that σ should capture a sequence of courses with some common theme and



(a) (b) (c)

Figure 18: (a) shows the 13 different courses that were available to students in this dataset. (b) is a histogram of the value of every 4 course sequence that appears in the dataset. The values of the courses selected by the various algorithms are overlayed on top of the corresponding bar in the histogram. (c) shows representative course sequences selected by OMegA and Hyper Sequence-Greedy.

present them in the best possible order. Therefore, if a user begins all the courses in σ they likely have some interest in this common theme. Hence, if σ is a good sequence, it will present these courses in a good order and properly pique the interest of these users.

Mathematically, we define S_σ to be the set of users who started all the courses in σ in the proper order, and c_{ij} to be the percentage of course j that user i completed. Therefore, the value of σ for a given test set D_{test} is defined as:

$$D_{test}(\sigma) = \frac{\sum_{i \in S_\sigma} \sum_{j \in \sigma} c_{ij}}{\big(|S_\sigma| + d\big)|\sigma|}$$

Using a 75/25 training/testing split of the data and 4-fold cross validation, we compare the effectiveness of Hyper Sequence-Greedy, Sequence Greedy, OMegA, and Frequency for the task of selecting a sequence of 4 courses. Note that due to the inherent randomness in the training/testing split, there is some variance in the results. To be conservative, the results shown in Figure 18b are actually on the lower end of the performance we see from our algorithms. Figure 18c shows some representative sequences.

We see that Hyper Sequence-Greedy outperforms the other algorithms, as expected. From the histogram, we also see that Hyper Sequence-Greedy tends to select one of the best possible sequences, with Sequence-Greedy and OMegA both performing in the 90^{th} percentile. Somewhat surprisingly, OMegA (which has to use a random topological order in the absence of a directed acyclic graph) outperforms Sequence-Greedy. However, this may be explained by the fact that $k = 4$ is relatively

small. Unfortunately, only 1,153 users even started more than 4 courses, meaning that we cannot effectively test sequences of larger length with this dataset.

5.1.7 Conclusion

In this section, we presented work that extended results on submodular sequences from directed acyclic graphs to general graphs and hypergraphs. Our theoretical results showed that both our algorithms, Sequence-Greedy and Hyper Sequence-Greedy, approach a constant factor approximation to the optimal solution (for constant Δ). Furthermore, we demonstrated the utility of our algorithms, in particular the power of using hyperedges, on real world applications in movie recommendation, online link prediction, and the design of course sequences for MOOCs.

5.2 Adaptive Sequence Submodularity

5.2.1 Introduction to Adaptive Sequence Submodularity

The machine learning community has long recognized the importance of both sequential and adaptive decision making. The study of sequences has led to novel neural architectures such as LSTMs (Hochreiter and Schmidhuber, 1997), which have been used in a variety of applications ranging from machine translation (Sutskever et al., 2014) to image captioning (Vinyals et al., 2015). Similarly, the study of adaptivity has led to the establishment of some of the most popular subfields of machine learning including active learning (Settles, 2012) and reinforcement learning (Sutton and Barto, 2018).

In this section, we build on the work from Section 5.1 and we consider the optimization of problems where both sequences and adaptivity are integral part of the process. More specifically, we focus on problems that can be modeled as selecting a sequence of items, where each of these items takes on some (initially unknown) state. The idea is that the value of any sequence depends not only on the items selected and the order of these items but also on the states of these items.

As in the previous section, we will consider recommender systems as a running example. To start, the order in which we recommend items can be just as important as the items themselves. For instance, if we believe that a user will enjoy the Lord of the Rings franchise, it is vital that we recommend the movies in the proper order. If we suggest that the user watches the final installment first,

she may end up completely unsatisfied with an otherwise excellent recommendation. Furthermore, whether it is explicit feedback (such as rating a movie on Netflix) or implicit feedback (such as clicking/not clicking on an advertisement), most recommender systems are constantly interacting with and adapting to each user. This feedback allows us to learn about the states of items we have already selected, as well as make inferences about the states of items we have not selected yet.

Unfortunately, the expressive modeling power of sequences and adaptivity comes at a cost. Not only does optimizing over sequences instead of sets exponentially increase the size of the search space, but adaptivity also necessitates a probabilistic approach that further complicates the problem. Without further assumptions, even approximate optimization is infeasible. As a result, we address this challenge from the perspective of *submodularity*, an intuitive diminishing returns condition that appears in a broad scope of different areas, but still provides enough structure to make the problem tractable.

In terms of sequences, as discussed earlier, the majority of research on submodularity has focused on sets rather than sequences. Section 5.1.1 gives further background. In terms of adaptivity, adaptive *set* submodularity has been studied extensively (Golovin and Krause, 2011; Chen and Krause, 2013; Gotovos et al., 2015; Fujii and Sakaue, 2019; Esfandiari et al., 2020; Agarwal et al., 2019), these approaches fail to capture order dependencies. Other relevant work includes a paper by Chen et al. (2015).

Our Contributions The main contributions of this section are organized as follows:

- In Section 5.2.2, we introduce our framework of *adaptive sequence submodularity*, which brings tractability to problems that include both sequences and adaptivity.
- In Section 5.2.3, we present our algorithm for adaptive sequence submodular maximization. We present theoretical guarantees for our approach and we elaborate on the necessity of our novel proof techniques. We also show that these techniques simultaneously improve the state-of-the-art bounds for the problem of sequence submodularity by a factor of $\frac{e}{e-1}$. Furthermore, we argue that any approximation guarantee must depend on the structure of the underlying graph unless the exponential time hypothesis is false.
- In Sections 5.2.4 and 5.2.5, we use datasets from Amazon and Wikipedia to compare our algorithm against existing sequence submodular baselines, as well as state-of-the-art deep learning-based approaches.

5.2.2 Adaptive Sequence Submodularity Framework

As discussed above, sequences and adaptivity are an integral part of many real-world problems. This means that many real-world problems can be modeled as selecting a sequence σ of items from a ground set V, where each of these items takes on some (initially unknown) state $o \in O$. A particular mapping of items to states is known as a **realization** ϕ, and we assume there is some unknown distribution $p(\phi)$ that governs these states.

For example in movie recommendation, the set of all movies is our ground set V and our goal is to select a sequence of movies that a particular user will enjoy. If we recommend a movie $v_i \in V$ and the user likes it, we place v_i in state 1 (i.e. $o_i = 1$). If not, we put it into state 0. Naturally, the value of a movie should be higher if the user liked it, and lower if she did not.

Formally, we want to select a sequence σ that maximizes $f(\sigma, \phi)$, where $f(\sigma, \phi)$ is the value of sequence σ under realization ϕ. However, ϕ is initially unknown to us and the state of each item in the sequence is revealed to us only after we select it. In fact, even if we knew ϕ perfectly, the set of all sequences poses an intractably large search space. From an optimization perspective, this problem is hopeless without further structural assumptions.

Our first step towards taming this problem is to follow the work of Tschiatschek et al. (2017) and assume that the value of a sequence can be defined using a graph. Concretely, we have a directed graph $G = (V, E)$, where each item in our ground set is represented as a vertex $v \in V$, and the edges encode the additional value intrinsic to picking certain items in certain orders. Mathematically, selecting a sequence of items σ will induce a set of edges $E(\sigma)$:

$$E(\sigma) = \big\{(\sigma_i, \sigma_j) \mid (\sigma_i, \sigma_j) \in E, i \leq j\big\}.$$

For example, consider the graph in Figure 19a and consider the sequence $\sigma_A = [F, T]$ where the user watched The Fellowship of the Ring, and then The Two Towers, as well as the sequence $\sigma_B = [T, F]$ where the user watched the same two movies but in the opposite order.

$$E(\sigma_A) = E\big([F, T]\big) = \big\{(F, F), (T, T), (F, T)\big\}$$
$$E(\sigma_B) = E\big([T, F]\big) = \big\{(T, T), (F, F)\big\}$$

Using the self-loops, this graph encodes the fact that there is certainly some intrinsic value to watching these movies regardless of the order. On the other hand, the edge (F, T) encodes the fact that watching The Fellowship of the Ring before The Two Towers will bring additional value to the

Figure 19: (a) shows an underlying graph for a movie recommendation problem. The vertices are movies and edges denote the additional value of watching certain movies in certain orders. (b) extends this to the adaptive case, where both the vertices and the edges take on a state. The user has reported that she liked the Fellowship of the Ring (so it is placed in state 1), but she did not like The Two Towers (so it is placed in state 0). The state of the last movie is still unknown. In this example, the state of an edge is equal to the state of its starting vertex.

viewer, and this edge is only induced if the movies appear in the correct order in the sequence.

With this graph based set-up, however, we run into issues when it comes to adaptivity. In particular, the states of items naturally translate to states for the vertices, but it is not clear how to extend adaptivity to the *edges*. We tackle this challenge by assigning a state $q \in Q$ to each edge strictly as a function of the states of its endpoints. That is, similarly to how a sequence σ induces a set of edges $E(\sigma)$, a realization ϕ for the states of the vertices induces a realization ϕ^E for the states of the edges. We want to emphasize that our framework works for *any* deterministic mapping from vertex states to edge states. One simple option that we will use throughout this section as a running example is to define the state of an edge to always be equal to the state of its start vertex.

As we will discuss later, the analysis for this approach will necessitate some novel proof techniques, but the resulting framework is very flexible and it allows us to fully redefine the adaptive sequence problem in terms of the underlying graph:

$$f(\sigma, \phi) = h\big(E(\sigma), \phi^E\big) \text{ where } \sigma \text{ induces } E(\sigma) \text{ and } \phi \text{ induces } \phi^E.$$

The last necessary ingredient to bring tractability to this problem is submodularity. In particular, we will assume that $h\big(E(\sigma), \phi^E\big)$ is *weakly adaptive set submodular*. This is a relaxed version of standard adaptive set submodularity that can model an even larger variety of problems, and it is a natural fit for the applications we consider in this section.

In order to formally define weakly-adaptive submodularity, we need a bit more terminology. To start, we define a **partial realization** ψ to be a mapping for only some subset of items (i.e., the states of the remaining items are unknown). For notational convenience, we define the domain of

ψ, denoted $dom(\psi)$, to be the list of items v for which the state of v is known. We say that ψ is a **subrealization** of ψ', denoted $\psi \subseteq \psi'$, if $dom(\psi) \subseteq dom(\psi')$ and they are equal everywhere in the domain of ψ. Intuitively, if $\psi \subseteq \psi'$, then ψ' has all the same information as ψ, and potentially more.

Given a partial realization ψ, we define the marginal gain of a set A as

$$\Delta(A \mid \psi) = \mathbb{E}\Big[h\big(dom(\psi) \cup A, \phi\big) - h\big(dom(\psi), \phi\big) \mid \psi\Big],$$

where the expectation is taken over all the full realizations ϕ such that $\psi \subseteq \phi$. In other words, we condition on the states given by the partial realization ψ, and then we take the expectation across all possibilities for the remaining states.

Definition 5.1. *A function $h : 2^E \times Q^E \to \mathbb{R}_{\geq 0}$ is **weakly adaptive set submodular** with parameter γ if for all sets $A \subseteq E$ and for all $\psi \subseteq \psi'$ we have:*

$$\Delta(A \mid \psi') \leq \frac{1}{\gamma} \cdot \sum_{e \in A} \Delta(e \mid \psi).$$

This notion is a natural generalization of weak submodular functions Das and Kempe (2011); Horel and Singer (2016) to adaptivity. The primary difference is that we condition on subrealizations instead of just sets because we need to account for the states of items. Note that in our context, h is a function on the edges, so we will condition on subrealizations of the edges ψ^E. However, these concepts apply more generally to functions on any set and state spaces, so we use ψ in the formal definitions.

Definition 5.2. *A function $h : 2^E \times Q^E \to \mathbb{R}_{\geq 0}$ is **adaptive monotone** if $\Delta(e \mid \psi) \geq 0$ for all partial realizations ψ. That is, the conditional expected marginal benefit of any element is non-negative.*

Figure 19b is designed to help clarify these concepts. It includes the same graph as Figure 19a, but now we can receive feedback from the user. If we recommend a movie and the user likes it, we put the corresponding vertex in state 1 (green in the image). Otherwise, we put the vertex in state 0 (red in the image). Vertices whose states are still unknown are denoted by a dotted black line.

Next, in our example, we need to define a state for each edge in terms of the states of its endpoints. In this case, we will define the state of each edge to be equal to the state of its start point. In Figure 19b, the user liked The Fellowship of the Ring, which puts edges (F, F), (F, T), and (F, R) in state 1 (green). She did not like The Two Towers, so edges (T, T) and (T, R) are in state 0 (red), and we do not know the state for The Return of the King, so the state of (R, R) is also unknown. We call this partial realization ψ_1 for the vertices, and the induced partial realization for the edges ψ_1^E.

Suppose our function h counts all induced edges that are in state 1. Furthermore, let us simply assume that any unknown vertex is equally likely to be in state 0 or state 1. This means that the self-loop (R, R) is also equally likely to be in either state 0 or state 1. Therefore, $\Delta\big((R, R) \mid \psi_1^E\big) = \frac{1}{2} \times 0 + \frac{1}{2} \times 1 = \frac{1}{2}$.

On the other hand, consider the edge (F, R). Under ψ_1, we know F is in state 1, which means (F, R) is also in state 1, and thus, $\Delta\big((F, R) \mid \psi_1^E\big) = 1$. However, if we consider a subrealization $\psi_2 \subseteq \psi_1$ where we do not know the state of F, then it is equally likely to be in either state and $\Delta\big((F, R) \mid \psi_2^E\big) = \frac{1}{2} \times 0 + \frac{1}{2} \times 1 = \frac{1}{2}$. Therefore, for this simple function we know that $\gamma \leq 0.5$.

5.2.3 Adaptive Sequence-Greedy Policy and Theoretical Results

In this section, we introduce our Adaptive Sequence-Greedy policy and present its theoretical guarantees. We first formally define **weakly adaptive sequence submodularity**.

Definition 5.3. *A function $f(\sigma, \phi)$ defined over a graph $G(V, E)$ is **weakly adaptive sequence submodular** if $f(\sigma, \phi) = h\big(E(\sigma), \phi^E\big)$ where a sequence σ of vertices in V induces a set of edges $E(\sigma)$, realization ϕ induces ϕ^E, and the function h is weakly adaptive set submodular. Note that if h is adaptive monotone, then f is also adaptive monotone.*

Formally, a policy π is an algorithm that builds a sequence of k vertices by seeing which states have been observed at each step, then deciding which vertex should be chosen and observed next. If $\sigma_{\pi,\phi}$ is the sequence returned by policy π under realization ϕ, then we write the expected value of π as:

$$f_{\text{avg}}(\pi) = \mathbb{E}\big[f(\sigma_{\pi,\phi}, \phi)\big] = \mathbb{E}\Big[h\big(E(\sigma_{\pi,\phi}), \phi^E\big)\Big]$$

where again the expectation is taken over all possible realizations ϕ. Our goal is to find a policy π that maximizes $f_{\text{avg}}(\pi)$, as defined above.

Our Adaptive Sequence Greedy policy π (Algorithm 15) starts with an empty sequence σ. Throughout the policy, we define ψ_σ to be the partial realization for the vertices in σ. In turn this gives us the partial realization ψ_σ^E for the induced edges.

At each step, we define the valid set of edges $\mathcal{E}$ to be the edges whose endpoint is not already in σ. The main idea of our policy is that, at each step, we select the valid edge $e \in \mathcal{E}$ with the highest expected value $\Delta(e \mid \psi_\sigma^E)$. For each such edge, the endpoints that are not already in the sequence σ are concatenated ($\oplus$ means concatenate) to the end of σ, and their states are observed (updating ψ_σ).

Algorithm 15 Adaptive Sequence Greedy Policy π

Input: Directed graph $G = (V, E)$, weakly adaptive sequence submodular $f(\sigma, \phi) = h\big(E(\sigma), \phi^E\big)$, and cardinality constraint k
Let $\sigma \leftarrow ()$
while $|\sigma| \leq k - 2$ **do**
    $\mathcal{E} = \{e_{ij} \in E \mid v_j \notin \sigma\}$
    if $\mathcal{E} \neq \emptyset$ **then**
        $e_{ij} = \arg\max_{e \in \mathcal{E}} \Delta(e \mid \psi_\sigma^E)$
        if $v_i = v_j$ **or** $v_i \in \sigma$ **then**
            $\sigma = \sigma \oplus v_j$ and observe state of v_j
        else
            $\sigma = \sigma \oplus v_i \oplus v_j$ and observe states of v_i, v_j
    else
        break
Return σ

Theorem 5.5. *For adaptive monotone and weakly adaptive sequence submodular function f, the Adaptive Sequence Greedy policy π represented by Algorithm 15 achieves*

$$f_{avg}(\pi) \geq \frac{\gamma}{2d_{\text{in}} + \gamma} \cdot f_{avg}(\pi^*),$$

where γ is the weakly adaptive submodularity parameter, π^* is the policy with the highest expected value and d_{in} is the largest in-degree of the input graph G.

As discussed by Mitrovic et al. (2018a), using a hypergraph H instead of a normal graph G allows us to encode more intricate relationships between the items. For example, in Figure 19a, the edges only encode pairwise relationships. However, there may be relationships between larger groups of items that we want to encode explicitly. For instance, if included, the value of a hyperedge (F, T, R) in Figure 19a would explicitly encode the value of watching The Fellowship of the Ring, followed by watching The Two Towers, and then concluding with The Return of the King.

We can also extend our policy to general hypergraphs (see Algorithm 20 in Appendix F.2.3). Theorem 5.6 guarantees the performance of our proposed policy for hypergraphs.

Theorem 5.6. *For adaptive monotone and weakly adaptive sequence submodular function f, the policy π' represented by Algorithm 20 achieves*

$$f_{avg}(\pi') \geq \frac{\gamma}{rd_{\text{in}} + \gamma} \cdot f_{avg}(\pi^*),$$

where γ is the weakly adaptive submodularity parameter, π^* is the policy with the highest expected value and r is the size of the largest hyperedge in the input hypergraph.

In our proofs, we have to handle the sequential nature of picking items and the revelation of states in a combined setting. Unfortunately, the existing proof methods for sequence submodular maximization are not linear enough to allow for the use of the linearity of expectation that captures the stochasticity of the states. For this reason, we develop a novel analysis technique to guarantee the performance of our algorithms. Our proof replaces several lemmas from Mitrovic et al. (2018a) with tighter, more linear analyses. Surprisingly, these new techniques also improve the theoretical guarantees of the non-adaptive Sequence-Greedy and Hyper Sequence-Greedy (Mitrovic et al., 2018a) by a factor of $\frac{e}{e-1}$.

Proofs for both theorems are given in Appendix F.2.

General Unifying Framework One more theoretical point we want to highlight is that weakly adaptive sequence submodularity provides a general unifying framework for a variety of common submodular settings including, adaptive submodularity, weak submodularity, sequence submodularity, and classical set submodularity. If we have $\gamma = 1$ and the state of all vertices is deterministic, then we have sequence submodularity. Conversely, if the vertex states are unknown, but our graph only has self-loops, then we have weakly adaptive set submodularity (and correspondingly adaptive set submodularity if $\gamma = 1$). Lastly, if we have a graph with only self-loops, full knowledge of all states, and $\gamma = 1$, then we recover the original setting of classical set submodularity.

Tightness of Theoretical Results We acknowledge that the constant factor approximation we present depends on the maximum in-degree. While ideally the theoretical bound would be completely independent of the structure of the graph, we argue here that such a dependence is likely necessary.

Indeed, getting a dependence better than $O(n^{1/4})$ in the approximation factor (where n is the total number of items) would improve the state-of-the-art algorithm for the very well-studied densest k subgraph problem (DkS) (Kortsarz and Peleg, 1993; Bhaskara et al., 2010). Moreover, if we could get an approximation that is completely independent of the structure of the graph, then the exponential time hypothesis would be proven false[4]. In fact, even an almost polynomial approximation would break the exponential time hypothesis Manurangsi (2017). Next, we formally state this hardness relationship. The proof is given in Appendix F.2.4.

Theorem 5.7. *Assuming the exponential time hypothesis is correct, there is no algorithm that approximates the optimal solution for the (adaptive) sequence submodular maximization problem within a $n^{1/(\log\log n)^c}$ factor, where n is the total number of items and $c > 0$ is a universal constant independent of n.*

5.2.4 Amazon Product Recommendation

Using the Amazon Video Games review dataset (McAuley et al., 2015), we consider the task of recommending products to users. In particular, given the first g products that the user has purchased, we want to predict the next k products that she will buy. Full experimental details are given in Appendix F.3.1.

We start by using the training data to build a graph $G = (V, E)$, where V is the set of all products and E is the set of edges between these products. The weight of each edge, w_{ij}, is defined to be the conditional probability of purchasing product j given that the user has previously purchased product i. There are also self-loops with weight w_{ii} that represent the fraction of users that purchased product i.

We define the state of each edge (i, j) to be equal to the state of product i. The intuitive idea is that edge (i, j) encodes the value of purchasing product j after already having purchased product i. Therefore, if the user has definitely purchased i (i.e., product i is in state 1), then they should receive the full value of w_{ij}. On the other hand, if she has definitely not purchased i (i.e., product i is in state 0), then edge (i, j) provides no value. Lastly, if the state of i is unknown, then the expected gain of edge (i, j) is discounted by w_{ii}, the value of the self-loop on i, which can be viewed as a simple estimate for the probability of the user purchasing product i. See Figure 20a for a small example.

4. If the exponential time hypothesis is true it would imply that P $\neq$ NP, but it is a stronger statement.

We use a probabilistic coverage utility function as our monotone weakly-adaptive set submodular function h. Mathematically,

$$h(E_1) = \sum_{j \in V} \Big[1 - \prod_{(i,j) \in E_1} (1 - w_{ij})\Big],$$

where $E_1 \subseteq E$ is the subset of edges that are in state 1. Note that with this set-up, the value of γ can be difficult to calculate exactly. However, roughly speaking, it is inversely proportional to the value of the smallest weight self-loop w_{ii}.

We compare the performance of our Adaptive Sequence-Greedy policy against Sequence-Greedy from Mitrovic et al. (2018a), the existing sequence submodularity baseline that does not consider states. To give further context for our results, we compare against Frequency, a naive baseline that ignores sequences and adaptivity and simply outputs the k most popular products.

We also compare against a set of deep learning-based approaches (see Appendix F.4 for full details). In particular, we implement adaptive and non-adaptive versions of both a regular Feed Forward Neural Network and an LSTM. The adaptive version will update its inputs after every prediction to reflect whether or not the user liked the recommendation. Conversely, the non-adaptive version will simply make k predictions using just the original input.

We use two different measures to compare the various algorithms. The first is the **Accuracy Score**, which simply counts the number of recommended products that the user indeed ended up purchasing. While this is a sensible measure, it does not explicitly consider the order of the sequence. Therefore, we also consider the **Sequence Score**, which is a measure based on the Kendall-Tau distance (Kendall, 1938). In short, this measure counts the number of ordered pairs that appear in both the predicted sequence and the true sequence. Figure 20d gives an example comparing the two measures.

Figures 20b and 20c show the performance of the various algorithms using the accuracy score and sequence score, respectively. These results highlight the importance of adaptivity as the adaptive algorithms consistently outperform their non-adaptive counterparts under both scoring regimes. Notice that in both cases, as the number of recommendations increases, our proposed Adaptive Sequence-Greedy policy is outperformed only by the Adaptive Feed Forward Neural Network. Although LSTMs are generally considered better for sequence data than vanilla feed-forward networks, we think it is a lack of data that causes them to perform poorly in our experiments.

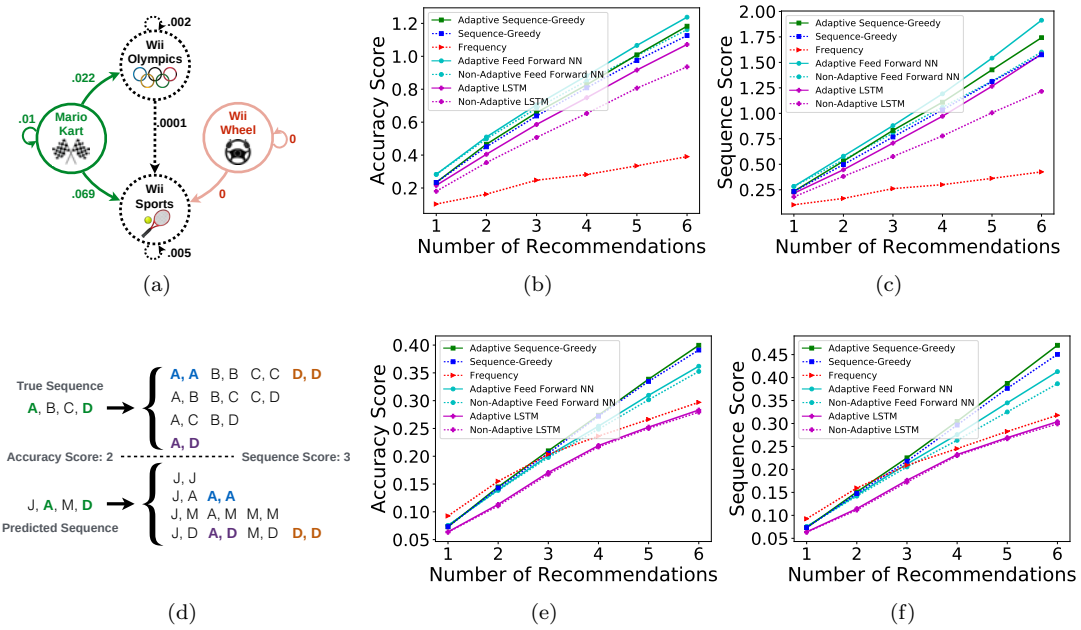


Figure 20: (a) shows a small subset of the underlying graph with states for a particular user. (b) and (c) show our results on the Amazon product recommendation task. In all these graphs, the number of given products g is 4. (d) gives an example illustrating the difference between the two performance measures. (e) and (f) show our results on the same task, but using only 1% of the available training to show that our algorithm outperforms deep learning-based approaches in data scarce environments.

Another observation, which fits the conventional wisdom, is that deep learning-based approaches can perform well when there is a lot of data. However, when the data is scarce, we see that the Sequence-Greedy based approaches outperform the deep learning-based approaches. Figures 20e and 20f simulate a data-scarce environment by using only 1% of the available data as training data. Note that the difference between the adaptive algorithms and their non-adaptive counterparts is less obvious in this setting because the adaptive algorithms use correct guesses to improve future recommendations, but the data scarcity makes it difficult to make a correct guess in the first place.

Aside from competitive accuracy and sequence scores, the Adaptive Sequence-Greedy algorithm provides several advantages over the neural network-based approaches. From a theoretical perspective, the Adaptive Sequence-Greedy algorithm has provable guarantees on its performance, while little is known about the theoretical performance of neural networks. Furthermore, the decisions made by the Adaptive Sequence-Greedy algorithm are easily interpretable and understandable (it is just picking the edge with the highest expected value), while neural networks are generally a black-box. On a similar note, Adaptive Sequence-Greedy may be preferable from an implementation perspective because it does not require any hyperparameter tuning. It is also more robust to changing inputs in

the sense that we can easily add another product and its associated edges to our graph, but adding another product to the neural network requires changing the entire input and output structure, and thus, generally necessitates retraining the entire network.

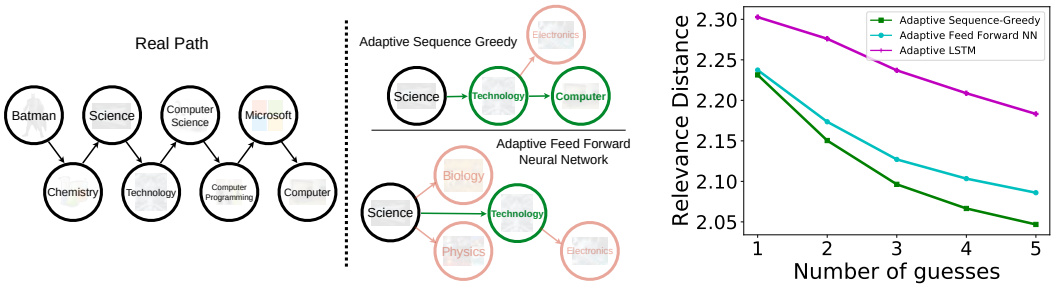
5.2.5 Wikipedia Link Prediction

Using the Wikispeedia dataset (West et al., 2009), we consider users who are surfing through Wikipedia towards some target article. Given a sequence of articles the user has previously visited, we want to guide her to the page she is trying to reach. Since different pages have different valid links, the order of pages we visit is critical to this task. Formally, given the first $g = 3$ pages each user visited, we want to predict which page she is trying to reach by making a series of suggestions for which link to follow.

In this case, we have $G = (V, E)$, where V is the set of all pages and E is the set of existing links between pages. Similarly to before, the weight w_{ij} of an edge $(i, j) \in E$ is the probability of moving to page j given that the user is currently at page i. In this case, there are no self-loops as we assume we can only move using links, and thus we cannot jump to random pages. We again define two states for the nodes: 1 if the user definitely visits this page and 0 if the user does *not* want to visit this page.

This application highlights the importance of adaptivity because the non-adaptive sequence submodularity framework cannot model this problem properly. This is because the Sequence-Greedy algorithm is free to choose any edge in the underlying graph, so there is no way to force the algorithm to pick a link that is connected to the user's current page. On the other hand, with Adaptive Sequence-Greedy, we can use the states to penalize invalid edges, and thus force the algorithm to select only links connected to the user's current page. Similarly, we only have the adaptive versions of the deep learning baselines because we need information about our current page in order to construct a valid path (Appendix F.4 gives a more detailed explanation).

Figure 21a shows an example of predicted paths, while Figure 21b shows our quantitative results. More detail about the relevance distance metric is given in Appendix F.3.2, but the idea is that it measures the relevance of the final output page to the true target page (a lower score indicates a higher relevance). The main observation is that the Adaptive Sequence Greedy algorithm actually outperforms the deep-learning based approaches. The main reason for this discrepancy is likely a lack of data as we have 619 pages to choose from and only 7,399 completed search paths.



(a) (b)

Figure 21: (a) The left side shows the real path a user followed from *Batman* to *Computer*. Given the first three pages, the right side shows the path predicted by Adaptive Sequence Greedy versus a deep learning-based approach. Green shows correct guesses that were followed, while red shows incorrect guesses that were not pursued further. (b) shows the overall performance of the various approaches.

5.2.6 Conclusion

In this section we introduced adaptive sequence submodularity, a general framework for bringing tractability to the broad class of optimization problems that consider both sequences and adaptivity. We presented Adaptive Sequence-Greedy—a general policy for optimizing weakly adaptive sequence submodular functions. We provide a provable theoretical guarantee for our algorithm, as well as a discussion about the tightness of our result. Our novel analysis also improves the theoretical guarantees of Sequence-Greedy and Hyper Sequence-Greedy (Mitrovic et al., 2018a) by a factor of $\frac{e}{e-1}$. Finally, we evaluated the performance of Adaptive Sequence-Greedy on an Amazon product recommendation task and a Wikipedia link prediction task. Not only does our Adaptive Sequence-Greedy policy exhibit competitive performance with the state-of-the-art, but it also provides several notable advantages, including interpretability, ease of implementation, and robustness against both data scarcity and input adjustments.

6. Conclusion

Although Nemhauser, Wolsey, and Fisher published their seminal paper on submodularity all the way back in 1978, there has been a strong resurgence in the interest, study, and application of submodularity in the past decade, particularly as it relates to machine learning. One of the primary reasons for this revival is that submodularity introduces enough structure to a problem so that we can develop algorithms with theoretical guarantees for both how quickly we will arrive at a solution (usually linear time, or faster), as well as how close that solution will be to the true optimal solution

(usually a constant-factor approximation). This stands in stark contrast to the majority of the current state-of-the-art in machine learning where heuristics with strong performance in practice, but no provable theoretical guarantees, still reign supreme.

While submodularity gives us enough structure to achieve these nice theoretical guarantees, these results would not be very interesting if they were never applicable in the real world. Indeed, another major reason for the popularity of submodularity in the machine learning community is that many machine learning problems naturally do have the diminishing returns property that submodularity requires. Section 1 gives a non-exhaustive list of machine learning problems where submodularity has successfully been applied. However, in spite of these successes, there are still many open problems in submodularity, particularly within the context of real-world machine learning applications. In this thesis, we presented novel results and solutions for three major challenges for modern machine learning applications of submodularity.

The first challenge relates to controversial topic of data privacy and the problem of somehow still using people's data while provably guaranteeing that their privacy will be protected. In Section 3, we propose a general and systematic study of differentially private submodular maximization. We present privacy-preserving algorithms for both monotone and non-monotone submodular maximization under cardinality, matroid, and p-system constraints, with guarantees that are competitive with optimal. Along the way, we analyze a new algorithm for non-monotone submodular maximization, which is the first (even non-privately) to achieve a constant approximation ratio while running in linear time. We additionally provide two concrete experiments to validate the efficacy of these algorithms. In the first experiment, we privately solve the facility location problem using a dataset of Uber pickup locations in Manhattan. In the second experiment, we perform private submodular maximization of a mutual information measure to select features relevant to classifying patients by diabetes status.

The second challenge relates to scalability and the exponential growth in data that we have seen in recent years. In general, more data is good, but in many cases, modern datasets have grown so large that even the provable linear-time algorithms enabled by submodularity can be too slow. In Section 4.1, we focus on a two-stage submodular framework where the goal is to use some given training functions to reduce the ground set so that optimizing new functions (drawn from the same distribution) over the reduced set provides almost as much value as optimizing them over the entire ground set. In particular, we develop the first streaming and distributed solutions to this problem. In addition to providing strong theoretical guarantees, we demonstrate both the utility

and efficiency of our algorithms on real-world tasks including image summarization and ride-share optimization.

In Section 4.2, we also focus on the challenge of scalability, but this time we look more closely at the streaming framework. We first propose Sieve-Streaming++, which requires just one pass over the data, keeps only $O(k)$ elements and achieves the tight $1/2$-approximation guarantee. The best previously known streaming algorithms either achieve a suboptimal $1/4$-approximation with $\Theta(k)$ memory or the optimal $1/2$-approximation with $O(k \log k)$ memory. Next, we show that by buffering a small fraction of the stream and applying a careful filtering procedure, one can heavily reduce the number of adaptive computational rounds, thus substantially lowering the computational complexity of Sieve-Streaming++. We then generalize our results to the more challenging multi-source streaming setting. We show how one can achieve the tight $1/2$-approximation guarantee with $O(k)$ shared memory while minimizing not only the required rounds of computations but also the total number of communicated bits. Finally, we demonstrate the efficiency of our algorithms on real-world data summarization tasks for multi-source streams of tweets and of YouTube videos.

The last major challenge we study in this thesis relates to sequences and the observation that for some machine learning problems (such as recommender systems), if we explicitly consider the order of both the input and the output (i.e. treat them as sequences rather than sets), we can get much better solutions. To extend the notion of submodularity to sequences, we use a directed graph on the items where the edges encode the additional value of selecting items in a particular order. Existing theory is limited to the case where this underlying graph is a directed acyclic graph. In Section 5.1, we introduce two new algorithms that provably give constant factor approximations for general graphs and hypergraphs having bounded in or out degrees. Furthermore, we show the utility of our new algorithms for real-world applications in movie recommendation, online link prediction, and the design of course sequences for MOOCs.

Building on the concept of sequence submodularity, we observed that in some machine learning applications, one needs to interactively select a sequence of items (e.g., recommending movies based on a user's feedback) or make sequential decisions in a certain order (e.g., guiding an agent through a series of states). Not only do sequences already pose a dauntingly large search space, but we must also take into account past observations, as well as the uncertainty of future outcomes. Without further structure, finding an optimal sequence is notoriously challenging, if not completely intractable. In Section 5.2, we introduce the framework of adaptive sequence submodularity and propose an adaptive greedy policy with strong theoretical guarantees. Additionally, to demonstrate the practical utility of

our results, we run experiments on Amazon product recommendation and Wikipedia link prediction tasks.

While the theoretical side of submodularity has seen a great deal of research and progress, the applied side is still catching up in comparison. There is a vast variety of important and interesting applications of submodularity in the machine learning world, well beyond the few examples studied in this thesis. There are also still many challenges when it comes to machine learning applications of submodularity, but we hope that the work and results presented in this thesis are a step towards eventually overcoming these challenges and ultimately promoting more wide-spread adoption of submodularity past purely research settings and into the real world.

References

Arpit Agarwal, Sepehr Assadi, and Sanjeev Khanna. Stochastic submodular cover with limited adaptivity. In *SODA*, 2019.

Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular Secretary Problem with Shortlists. In *Innovations in Theoretical Computer Science Conference, ITCS*, pages 1:1–1:19, 2019.

Saeed Alaei and Azarakhsh Malekian. Maximizing sequence-submodular functions and its application to online advertising. *arXiv preprint arXiv:1009.4153*, 2010.

Francis Bach. *Learning with submodular functions: A convex optimization perspective*. Foundations and Trends in Machine Learning, 2013.

Ashwin Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *KDD*, 2014.

Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *SODA*, 2014.

Eric Balkanski and Yaron Singer. The adaptive complexity of maximizing a submodular function. In *STOC*, 2018.

Eric Balkanski, Baharan Mirzasoleiman, Andreas Krause, and Yaron Singer. Learning sparse combinatorial representations via two-stage submodular maximization. In *ICML*, 2016.

Eric Balkanski, Adam Breuer, and Yaron Singer. Non-monotone Submodular Maximization in Exponentially Fewer Iterations. In *Advances in Neural Information Processing Systems*, pages 2359–2370, 2018.

Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An Exponential Speedup in Parallel Running Time for Submodular Maximization without Loss in Approximation. In *SODA*, 2019a.

Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An optimal approximation for submodular maximization under a matroid constraint in the adaptive complexity model. In *STOC*, 2019b.

Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *STOC*, 2006.

Rafael Barbosa, Alina Ene, Huy Nguyen, and Justin Ward. The power of randomization: Distributed submodular maximization on massive datasets. In *ICML*, 2015.

Rafael Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. A New Framework for Distributed Submodular Maximization. In *FOCS*, 2016.

Raef Bassily, Adam D. Smith, and Abhradeep Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *FOCS*, 2014.

Raef Bassily, Kobbi Nissim, Adam D. Smith, Thomas Steinke, Uri Stemmer, and Jonathan Ullman. Algorithmic stability for adaptive data analysis. In *STOC*, 2016.

Amos Beimel, Kobbi Nissim, and Uri Stemmer. Private learning and sanitization: Pure vs. approximate differential privacy. *Theory of Computing*, 12(1):1–61, 2016.

Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC*, 2010.

Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *SODA*, 2014.

Niv Buchbinder, Moran Feldman, and Roy Schwartz. Online submodular maximization with preemption. In *SODA*, 2015.

Niv Buchbinder, Moran Feldman, and Roy Schwartz. Comparing Apples and Oranges: Query Trade-off in Submodular Maximization. *Math. Oper. Res.*, 42(2):308–329, 2017.

Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, pages 635–658, 2016.

Mark Bun, Kobbi Nissim, Uri Stemmer, and Salil P. Vadhan. Differentially private release and learning of threshold functions. In *FOCS*, 2015.

Gruia Calinescu, Chandra Chekuri, Martin Pal, and Jan Vondrak. Maximizing a submodular set function subject to a matroid constraint. *SIAM Journal on Computing*, 2011.

Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Math. Program.*, 154(1-2):225–247, 2015.

T.-H. Hubert Chan, Zhiyi Huang, Shaofeng H.-C. Jiang, Ning Kang, and Zhihao Gavin Tang. Online Submodular Maximization with Free Disposal: Randomization Beats 1/4 for Partition Matroids. In *SODA*, 2017.

Kamalika Chaudhuri, Daniel J. Hsu, and Shuang Song. The large margin mechanism for differentially private maximization. In *NeurIPS*, 2014.

Chandra Chekuri and Kent Quanrud. Parallelizing greedy for submodular set function maximization in matroids and beyond. In *STOC*, 2019.

Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *International Colloquium on Automata, Languages, and Programming*, pages 318–330. Springer, 2015.

Jiecao Chen, Huy L. Nguyen, and Qin Zhang. Submodular Maximization over Sliding Windows. *CoRR*, abs/1611.00129, 2016.

Lin Chen, Andreas Krause, and Amin Karbasi. Interactive submodular bandit. In *NIPS*, 2017.

Lin Chen, Moran Feldman, and Amin Karbasi. Unconstrained submodular maximization with constant adaptive complexity. *CoRR*, abs/1811.06603, 2018.

Yuxin Chen and Andreas Krause. Near-optimal batch mode active learning and adaptive submodular optimization. In *ICML*, 2013.

Yuxin Chen, S Hamed Hassani, Amin Karbasi, and Andreas Krause. Sequential information maximization: When is greedy near-optimal? In *COLT*, 2015.

Abhimanyu Das and David Kempe. Submodular meets Spectral: Greedy Algorithms for Subset Selection, Sparse Approximation and Dictionary Selection. In *ICML*, 2011.

Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *STOC*, 2009.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, pages 265–284, 2006.

Cynthia Dwork, Guy N. Rothblum, and Salil P. Vadhan. Boosting and differential privacy. In *FOCS*, 2010.

Ethan Elenberg, Alexandros Dimakis, Moran Feldman, and Amin Karbasi. Streaming weak submodularity: Interpreting neural networks on the fly. *NeurIPS*, 2017.

Alina Ene and Huy L. Nguyen. Submodular Maximization with Nearly-optimal Approximation and Adaptivity in Nearly-linear Time. In *SODA*, 2019.

Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. Submodular Optimization Over Sliding Windows. In *WWW*, 2017.

Hossein Esfandiari, Amin Karbasi, and Vahab Mirrokni. Adaptivity in adaptive submodularity. In *arXiv preprint arXiv:2002.03503*, 2020.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Non-monotone Submodular Maximization with Nearly Optimal Adaptivity Complexity. *CoRR*, abs/1808.06932, 2018.

Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Submodular Maximization with Nearly Optimal Approximation, Adaptivity and Query Complexity. In *SODA*, 2019.

U. Feige, V. Mirrokni, and J. Vondrak. Maximizing non-monotone submodular functions. In *FOCS*, 2007.

Uriel Feige. On maximizing welfare when utility functions are subadditive. *SIAM Journal on Computing*, 39:122–142, 2009.

Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *FOCS*, 2011.

Moran Feldman, Christopher Harshaw, and Amin Karbasi. Greed Is Good: Near-Optimal Submodular Maximization via Greedy Optimization. In *COLT*, 2017.

Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do Less, Get More: Streaming Submodular Maximization with Subsampling. In *NeurIPS*, 2018.

Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions - II. *Mathematical Programming Study*, (8), 1978.

Kaito Fujii and Shinsaku Sakaue. Beyond Adaptive Submodularity: Approximation Guarantees of Greedy Policy with Adaptive Submodularity Ratio. In *ICML*, 2019.

Satoru Fujishige. *Submodular functions and optimization.* Elsevier Science, 2nd edition, 2005.

Victor Gabillon, Branislav Kveton, Zheng Wen, Brian Eriksson, and S. Muthukrishnan. Adaptive submodular maximization in bandit settings. In *NeurIPS*, 2013.

Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *SODA*, 2011.

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.

Daniel Golovin, Andreas Krause, and Debajyoti Ray. Near-optimal bayesian active learning with noisy observations. In *NeurIPS*, 2010.

Manuel Gomez Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *KDD*, 2010.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. In *MIT Press*, 2016.

Alkis Gotovos, Amin Karbasi, and Andreas Krause. Non-monotone adaptive submodular maximization. In *IJCAI*, 2015.

Andrew Guillory and Jeff Bilmes. Interactive submodular set cover. In *ICML*, 2010.

Anupam Gupta, Katrina Ligett, Frank McSherry, Aaron Roth, and Kunal Talwar. Differentially private combinatorial optimization. In *SODA*, 2010.

Ran Haba, Ehsan Kazemi, Moran Feldman, and Amin Karbasi. Streaming submodular maximization under a k-set system constraint. In *arXiv preprint arXiv:2002.03352*, 2020.

F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 2015.

Hamed Hassani, Mahdi Soltanolkotabi, and Amin Karbasi. Gradient methods for submodular maximization. In *NeurIPS*, 2017.

Avinatan Hassidim and Yaron Singer. Robust guarantees of stochastic greedy algorithms. In *ICML*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Ralf Herbrich, Neil D Lawrence, and Matthias Seeger. Fast sparse gaussian process methods: The informative vector machine. In *NeurIPS*, 2003.

Andrew Dean Ho, Justin Reich, Sergiy O Nesterko, Daniel Thomas Seaton, Tommy Mullaney, Jim Waldo, and Isaac Chuang. Harvardx and mitx: The first year of open online courses. *ssrn.com/abstract=2381263*, 2014.

Sepp Hochreiter and Jrgen Schmidhuber. Long short-term memory. In *Neural Computation*, 1997.

Thibaut Horel and Yaron Singer. Maximization of approximately submodular functions. In *NeurIPS*, 2016.

Stefanie Jegelka and Jeff A. Bilmes. Online submodular minimization for combinatorial structures. In *ICML*, Bellevue, Washington, 2011.

T. A. Jenkyns. The efficacy of the "greedy" algorithm. In *South Eastern Conference on Combinatorics, Graph Theory and Computing*, 1976.

Mohammad Reza Karimi, Mario Lucic, Hamed Hassani, and Andreas Krause. Stochastic submodular maximization: The case of coverage functions. In *NeurIPS*, 2017.

Ehsan Kazemi, Morteza Zadimoghaddam, and Amin Karbasi. Scalable deletion-robust submodular maximization: Data summarization with privacy and fairness constraints. *ICML*, 2018.

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 *ICML*, 2019.

Ehsan Kazemi, Shervin Minaee, Moran Feldman, and Amin Karbasi. Regularized submodular maximization at scale. In *arXiv preprint arXiv:2002.03503*, 2020.

David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD*, 2003.

Maurice Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

Katrin Kirchhoff and Jeff Bilmes. Submodularity for data selection in statistical machine translation. In *EMNLP*, 2014.

Guy Kortsarz and David Peleg. On Choosing a Dense Subgraph (Extended Abstract). In *FOCS*, 1993.

Andreas Krause and Carlos Guestrin. Near-optimal nonmyopic value of information in graphical models. In *UAI*, 2005.

Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. In *Journal of Machine Learning Research*, volume 9, 2008.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. In *SPAA*, 2013.

Yan LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. In *Nature*, 2015.

Pan Li and Olgica Milenkovic. Inhomogeneous hypergraph clustering with applications. *NeurIPS*, 2017.

Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *ACL*, 2011.

Hui Lin and Jeff Bilmes. Learning mixtures of submodular shells with application to document summarization. In *UAI*, 2012.

Paul Liu and Jan Vondrák. Submodular Optimization in the MapReduce Model. *CoRR*, abs/1810.01489, 2018.

Pasin Manurangsi. Almost-polynomial Ratio ETH-hardness of Approximating Densest K-subgraph. In *STOC*, pages 954–961, 2017.

J. McAuley, C. Targett, J. Shi, and A. van den Hengel. Image-based recommendations on styles and substitutes. In *SIGIR*, 2015.

Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *FOCS*, 2007.

Julián Mestre. Greedy in approximation algorithms. In *ESA*. 2006.

Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Proc. of the 8th IFIP Conference on Optimization Techniques*. Springer, 1978.

Vahab Mirrokni and Morteza Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *STOC*, 2015.

Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *NeurIPS*, 2013.

Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrak, and Andreas Krause. Lazier than lazy greedy. In *AAAI*, 2015.

Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *ICML*, 2016a.

Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization. *Journal of Machine Learning Research (JMLR)*, 2016b.

Baharan Mirzasoleiman, Morteza Zadimoghaddam, and Amin Karbasi. Fast distributed submodular cover: Public-private data summarization. In *NeurIPS*, 2016c.

Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Deletion-Robust Submodular Maximization: Data Summarization with "the Right to be Forgotten". In *ICML*, 2017.

Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming Non-Monotone Submodular Maximization: Personalized Video Summarization on the Fly. In *AAAI*, 2018.

Marko Mitrovic, Mark Bun, Andreas Krause, and Amin Karbasi. Differentially Private Submodular Maximization: Data Summarization in Disguise. In *ICML*, 2017a.

Marko Mitrovic, Moran Feldman, Andreas Krause, and Amin Karbasi. Submodularity on hypergraphs: From sets to sequences. *AISTATS*, 2018a.

Marko Mitrovic, Ehsan Kazemi, Morteza Zadimoghaddam, and Amin Karbasi. Data summarization at scale: A two-stage submodular approach. In *ICML*, 2018b.

Marko Mitrovic, Ehsan Kazemi, Moran Feldman, Andreas Krause, and Amin Karbasi. Adaptive Sequence Submodularity. In *NeurIPS*, 2019.

Slobodan Mitrovic, Ilija Bogunovic, Ashkan Norouzi-Fard, Jakub M Tarnawski, and Volkan Cevher. Streaming Robust Submodular Maximization: A Partitioned Thresholding Approach. In *NeurIPS*, 2017b.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. In *Nature*, 2015.

Aryan Mokhtari, Hamed Hassani, and Amin Karbasi. Conditional gradient method for stochastic submodular maximization: Closing the gap. In *AISTATS*, 2018.

George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Mathematical Programming*, 1978.

NHANESDataset. National health and nutrition examination survey (2007 - 2014). URL `https://wwwn.cdc.gov/nchs/nhanes/default.aspx`.

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 *ICML*, 2018.

Christos H. Papadimitriou, Michael Schapira, and Yaron Singer. On the hardness of being truthful. In *FOCS*, 2008.

Burr Settles. Active learning. In *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2012.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. In *Nature*, 2016.

Adish Singla, Ilija Bogunovic, Gábor Bartók, Amin Karbasi, and Andreas Krause. Near-optimally teaching the crowd to classify. In *ICML*, 2014.

Serban Stan, Morteza Zadimoghaddam, Andreas Krause, and Amin Karbasi. Probabilistic submodular maximization in sub-linear time. In *ICML*, 2017.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NeurIPS*, 2014.

Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction, 2nd edition. In *MIT Press*, 2018.

Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2008.

Sebastian Tschiatschek, Adish Singla, and Andreas Krause. Selecting sequences of items via submodular maximization. In *AAAI*, 2017.

UberDataset. Uber pickups in new york city. URL `https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city`.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *CVPR*, 2015.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Kuttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado Van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft ii: A new challenge for reinforcement learning. In *arXiv preprint arXiv:1708.04782*, 2017.

Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. Using document summarization techniques for speech data subset selection. In *Proceedings of Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2013.

Robert West, Joelle Pineau, and Doina Precup. An online game for inferring semantic distances between concepts. In *IJCAI*, 2009.

Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

Yisong Yue and Carlos Guestrin. Linear submodular bandits and its application to diversified retrieval. In *NeurIPS*, 2011.

Zhenliang Zhang, Edwin K. P. Chong, Ali Pezeshki, and William Moran. String submodular functions with curvature constraints. *IEEE Transactions on Automatic Control*, 61(3):601–616, 2016.

Appendix A. Introduction Appendix

A.1 Greedy

Let f be a given non-negative, monotone submodular function we want to maximize. Let $S^* = \arg\max_{|S| \leq k} f(S)$ be the true optimal set and let S_i be the set chosen by the greedy algorithm after i iterations. We want to show that, under a cardinality constraint, the greedy algorithm achieves a $(1 - \frac{1}{e})$ approximation to the optimal set.

We will first use a proof by induction to show that:

$$f(S^*) - f(S_i) \leq (1 - \frac{1}{k})^i f(S^*)$$

We start off with the base case. When $i = 0$, we want to show that

$$\begin{aligned} f(S^*) - f(S_0) &\leq (1 - \frac{1}{k})^0 f(S^*) \\ &= f(S^*) \end{aligned}$$

This simplifies to $f(S_0) \geq 0$, which we know is true because f is assumed to be non-negative.

Next, we have the inductive step where we assume that for all $i <= l$:

$$f(S^*) - f(S_i) \leq (1 - \frac{1}{k})^i f(S^*)$$

We now prove this statement to be true for $i = l + 1$.

First, assume there are $m \leq k$ items from the optimal set that are not yet included in S_l ($m \leq k$ because the optimal set has k total items). That is, $S^* \setminus S_l = \{v_1^*, v_2^*, \ldots, v_m^*\}$. For notational simplicity, we will also define $S_l^j = S_l \cup \{v_1^*, \ldots, v_j^*\}$ to be S_l unioned with the first j missing optimal items (note that the order of the missing optimal items is not important, any order will work). Lastly, let v_{l+1} be the item selected by the greedy algorithm at step $l + 1$.

With these definitions, we make the following observations:

1. $f(S_l^0) = f(S_l)$. This is because S_l^0 is S_l without any items added, so clearly they will have the

same value.

2. $f(S_l^m) \geq f(S^*)$. This is because S_l^m is S_l with all of the missing optimal items added. Therefore, $S^* \subseteq S_l^m$, and by monotonicity we have $f(S^*) \leq f(S_l^m)$.

3. $f(S_l^m) = f(S_l^{m-1}) + f(v_m^* \mid S_l^{m-1})$. This is just saying that the value of S_l^m is equal to the value of S_l^{m-1} plus the marginal gain of the next element v_m^*.

Using these observations, we can expand out $f(S^*) - f(S_l)$:

$$
\begin{aligned}
f(S^*) - f(S_l) &= f(S^*) - f(S_l^0) && \text{By observation 1} \\
&\leq f(S_l^m) - f(S_l^0) && \text{By observation 2} \\
&= \big[f(S_l^{m-1}) + f(v_m^* \mid S_l^{m-1})\big] - f(S_l^0) && \text{By observation 3} \\
&= f(v_m^* \mid S_l^{m-1}) + \ldots + f(v_1^* \mid S_l^0) + f(S_l^0) - f(S_l^0) && \text{By observation 3} \\
&= f(v_m^* \mid S_l^{m-1}) + \ldots + f(v_1^* \mid S_l^0) \\
&\leq f(v_m^* \mid S_l^0) + \ldots + f(v_1^* \mid S_l^0) && \text{By submodularity} \\
&\leq f(v_{l+1} \mid S_l^0) + \ldots + f(v_{l+1} \mid S_l^0) && \text{By greedy choice of } v_{l+1} \\
&\leq k f(v_{l+1} \mid S_l^0) && \text{Because } m \leq k \\
&= k f(v_{l+1} \mid S_l) && \text{By observation 1}
\end{aligned}
$$

Therefore, we have:

$$
f(v_{l+1} \mid S_l) \geq \frac{1}{k}\Big[f(S^*) - f(S_l)\Big] \qquad \text{Call this observation 4}
$$

Next, notice that:

$$
\begin{aligned}
f(S^*) - f(S_{l+1}) &= f(S^*) - \big(f(S_l) + f(v_{l+1} \mid S_{l-1})\big) \\
&= f(S^*) - f(S_l) - f(v_{l+1} \mid S_{l-1}) \\
&\leq f(S^*) - f(S_l) - \frac{1}{k}\Big[f(S^*) - f(S_l)\Big] && \text{By observation 4} \\
&= \big(1 - \frac{1}{k}\big)\Big[f(S^*) - f(S_l)\Big] \\
&\leq \big(1 - \frac{1}{k}\big)^{l+1} f(S^*) && \text{By inductive hypothesis}
\end{aligned}
$$

This completes our proof by induction, which means that

$$
f(S^*) - f(S_i) \leq (1 - \frac{1}{k})^i f(S^*)
$$

holds for all i.

Therefore, consider the termination of the greedy algorithm where $i = k$. We have:

$$\begin{aligned} f(S^*) - f(S_k) &\leq (1 - \frac{1}{k})^k f(S^*) \\ &\leq \frac{1}{e} f(S^*) \end{aligned}$$

Therefore,

$$f(S_k) \geq (1 - \frac{1}{e}) f(S^*)$$

Appendix B. Submodularity and Differential Privacy Appendix

B.1 Additional Details About Large Margin Mechanism

Formally, given a quality function $q : V \times X^n \to \mathbb{R}$ and parameters $\ell \in \mathbb{N}, \gamma > 0$, a dataset D satisfies the (ℓ, γ)-margin condition if $q(v_{\ell+1}, D) < q(v_1, D) - \gamma$.

For each $\ell = 1, \ldots, |V|$, define

$$\begin{aligned} g_\ell &= \lambda \cdot \left(3 + \frac{4 \ln(2\ell/\delta)}{\varepsilon}\right) \\ G_\ell &= \frac{8\lambda \ln(2/\delta)}{\varepsilon} + \frac{16\lambda \ln(7\ell^2/\delta)}{\varepsilon} + g_\ell. \end{aligned}$$

The Laplace distribution $\text{Lap}(b)$ is specified by the density function $\frac{1}{2b} \exp(-|x|/b)$, and a sample $Z \sim \text{Lap}(b)$ obeys the tail bound $\Pr[Z > t] = \frac{1}{2} \exp(-t/b)$ for all $t > 0$.

Algorithm 16 Large Margin Mechanism (LMM)

Input: Quality function $q : V \times X^n \to \mathbb{R}$, dataset D, privacy parameters $\varepsilon, \delta > 0$
Output: Item $\hat{v} \in V$

1. Sort the elements of V so that $q(v_1, D) \geq \cdots \geq q(v_{|V|}, D)$
2. Let $m = q(v_1, D) + Z$ for $Z \sim \text{Lap}(8\lambda/\varepsilon)$
3. For $\ell = 1, \ldots, |V|$:
   - Sample $Z_\ell \sim \text{Lap}(16\lambda/\varepsilon)$
   - If $m - q(v_{\ell+1}, D) > G_\ell + Z_\ell$: Break, reporting ℓ
4. Return $\hat{v} \in \{v_1, \ldots, v_\ell\}$ sampled w.p. $\propto \exp(\varepsilon q(v_i, D)/4\lambda)$.

Proposition B.1. *Let $\varepsilon, \delta > 0$. Consider the Large Margin Mechanism described in Algorithm 16. Then*

- *Algorithm* LMM *is* (ε, δ)*-differentially private.*
- *Suppose* $D \in X^n$ *satisfies the* (ℓ, γ)*-margin condition for*

$$\gamma = \frac{24\lambda \ln(1/\eta)}{\varepsilon} + G_\ell$$

for some $\eta > 0$*. Then there exists an event* E *with* $\Pr[E] \geq 1 - \eta$ *such that*

$$\mathbb{E}[q(\hat{v}, D)|E] \geq \mathrm{OPT} - \frac{4\lambda \cdot \ln \ell}{\varepsilon},$$

where $\hat{v}$ *is the output of* $\mathrm{LMM}(D)$*.*

Our presentation of Algorithm 16 differs slightly from that of Chaudhuri et al. (2014). Namely, we simplify the choice of the noisy maximum m, and redistribute the algorithm's use of the privacy budget ε with an eye toward better performance in applications. Because of these small changes, we sketch the proof of Proposition B.1 for completeness.

Privacy Analysis of Proposition B.1. Algorithm 16 can be thought of as releasing two items in stages: First, the margin parameter ℓ in Step 3, and second, the item $\hat{v}$ sampled via the exponential mechanism in Step 4. We first claim that releasing the margin parameter ℓ guarantees $(\varepsilon/2, 0)$-differential privacy. This follows because Steps 2 and 3 taken together are an instantiation of the "AboveThreshold" algorithm, as presented by Dwork and Roth (Dwork and Roth, 2014, Theorem 3.23), with respect to the sensitivity-(2λ) functions $q(v_1, D) - q(v_{\ell+1}, D)$. Denote the output ℓ of the algorithm at Step 3 by $S(D)$.

We now establish that Step 4 provides differential privacy. Following Chaudhuri et al. (2014), we let $A(\ell, D)$ capture the behavior of the algorithm in Step 4, where on receiving ℓ from Step 3, it samples from the exponential mechanism on the top ℓ elements. They proved the following lemma about $A(\ell, D)$:

Lemma B.2 ((Chaudhuri et al., 2014, Lemma 5))**.** *If* D *satisfies the* (ℓ, γ)*-margin condition with*

$$\gamma \geq 2\lambda \left(1 + \frac{2\ln(\ell/\delta')}{\varepsilon}\right)$$

for some $\delta' > 0$, then for every neighbor $D' \sim D$ and any $T \subseteq V$, we have

$$\Pr[A(\ell, D) \in T] \leq e^{\varepsilon/2} \Pr[A(\ell, D') \in T] + \delta'.$$

Now fix neighboring datasets $D \sim D'$. Let $\mathcal{L}$ denote the set of ℓ for which $q(v_1, D) - q(v_{\ell+1}, D) \geq g_\ell$. By definition, if $\ell = S(D) \in \mathcal{L}$, then D indeed satisfies the (ℓ, g_ℓ)-margin condition. Moreover, by tail bounds on the Laplace distribution,

$$\begin{aligned}
\Pr[S(D) \notin \mathcal{L}] &\leq \Pr\left[Z > 8\lambda \ln(2/\delta)/\varepsilon \vee \left(\exists \ell \in \{1, \ldots, |V|\} : Z_\ell < -16\lambda \ln(7\ell^2/\delta)/\varepsilon\right)\right] \\
&\leq \frac{\delta}{4} + \sum_{\ell=1}^{|V|} \frac{6\delta}{4\pi^2 \ell^2} \\
&\leq \frac{\delta}{2}.
\end{aligned}$$

Hence, we have that for any $T \subseteq V$,

$$\begin{aligned}
\Pr[\mathrm{LMM}(D) \in T] &\leq \sum_{\ell \in \mathcal{L}} \Pr[\mathrm{LMM}(D) \in T | S(D) = \ell] \cdot \Pr[S(D) = \ell] + \Pr[S(D) \notin \mathcal{L}] \\
&\leq \sum_{\ell \in \mathcal{L}} \Pr[\mathrm{LMM}(D) \in T | S(D) = \ell] \cdot e^{\varepsilon/2} \Pr[S(D') = \ell] + \frac{\delta}{2} \\
&\leq \sum_{\ell \in \mathcal{L}} (e^{\varepsilon/2} \Pr[\mathrm{LMM}(D') \in T | S(D') = \ell] + e^{-\varepsilon/2} \frac{\delta}{2}) \cdot e^{\varepsilon/2} \Pr[S(D') = \ell] + \frac{\delta}{2} \\
&\qquad \text{by Lemma B.2} \\
&\leq e^{\varepsilon} \Pr[\mathrm{LMM}(D') \in T] + \delta
\end{aligned}$$

This completes the privacy proof of Proposition B.1.

□

Utility Analysis of Proposition B.1. Suppose D satisfies the (ℓ, γ)-margin condition with

$$\gamma \geq \frac{24\lambda \ln(1/\eta)}{\varepsilon} + G_\ell,$$

for some $\eta > 0$. By the tail bound for the Laplace distribution and a union bound, we have that

with probability at least $1 - \eta$,

$$Z \geq \frac{8}{\lambda} \ln \frac{1}{\eta} \quad \text{and} \quad Z_\ell \leq \frac{16}{\lambda} \ln \frac{1}{\eta}.$$

Let E be the event where this occurs. If E occurs, then indeed we have

$$(q(v_1, D) + Z) - q(v_{\ell+1}, D) > G_\ell + Z_\ell,$$

and hence Step 3 terminates outputting some $\ell' \leq \ell$. By Proposition 3.2, it follows that

$$\mathbb{E}[q(\hat{v}, D)|E] \geq \mathrm{OPT} - \frac{4\lambda \cdot \ln \ell}{\varepsilon}.$$

□

Replacing the exponential mechanism with the large margin mechanism gives analogues of our results for monotone submodular maximization with a cardinality constraint, monotone submodular maximization over a p-extendible system, and non-monotone submodular maximization with a cardinality constraint:

Theorem B.3. *Suppose $f_D : 2^V \to \mathbb{R}$ is monotone and has sensitivity λ. Then instantiating Algorithm 2 with $\mathcal{O} = \mathrm{LMM}$ and parameters $\varepsilon_0, \delta_0 = 0$ provides $(k\varepsilon_0, k\delta_0)$-differential privacy. It also provides $(\varepsilon, \delta' + k\delta_0)$-differential privacy for every $\delta' > 0$ with $\varepsilon = k\varepsilon^2/2 + \varepsilon \cdot \sqrt{2k \ln(1/\delta')}$.*

Moreover, for every $D \in X^n$, there exists an event E with $\Pr[E] \geq 1 - \beta$ such that

$$\mathbb{E}\left[f_D(S_k)|E\right] \geq \left(1 - \frac{1}{e}\right) \mathrm{OPT} - \sum_{i=1}^{k} \frac{4\lambda \ln \ell_i}{\varepsilon_0}$$

where $S_k \leftarrow_R \mathcal{G}^{\mathrm{LMM}}(D)$, and D satisfies the (ℓ_i, γ_i)-margin condition with respect to every function of the form $q_i(v, D) = f_D(\hat{S}_{i-1} \cup \{v\}) - f_D(\hat{S}_{i-1})$, with $\gamma_i = 24\lambda \ln(k/\beta)/\varepsilon + G_{\ell_i}$.

Theorem B.4. *Instantiating Algorithm 3 with $\mathcal{O} = \mathrm{LMM}$ under all of the conditions of Theorem B.3 gives the same privacy guarantee (replacing k with $r(\mathcal{I})$) and gives*

$$\mathbb{E}\left[f_D(S)|E\right] \geq \frac{1}{p+1} \cdot \mathrm{OPT} - \sum_{i=1}^{r(\mathcal{I})} \frac{4\lambda \ln \ell_i}{\varepsilon_0}.$$

Theorem B.5. *Instantiating Algorithm 4 with $\mathcal{O} = \mathrm{LMM}$ under all of the conditions of Theorem B.3*

gives the same privacy guarantee and gives

$$\mathbb{E}\left[f_D(S_k)|E\right] \geq \frac{1}{e}\left(1-\frac{1}{e}\right)\mathrm{OPT} - \sum_{i=1}^{k}\frac{4\lambda \ln \ell_i}{\varepsilon_0}.$$

Moreover, if f_D is monotone, then

$$\mathbb{E}\left[f_D(S_k)|E\right] \geq 0.468\,\mathrm{OPT} - \sum_{i=1}^{k}\frac{4\lambda \ln \ell_i}{\varepsilon_0}.$$

B.2 Submodular Maximization with Differential Privacy Proofs

B.2.1 Proof of Theorem 3.3

Proof. The privacy guarantee of Theorem 3.3 follows immediately from the $(\varepsilon, 0)$-differential privacy of the exponential mechanism, together with Theorem 3.1.

To simplify notation in the utility proofs in this section, we suppress the dependence of the submodular function of interest on D, i.e. we write $f = f_D$. We also introduce the notation $f_S(T) = f(S \cup T) - f(S)$ to denote the marginal gain by adding T to the set S.

To argue that the algorithm achieves good utility, recall that in each step i, the exponential mechanism guarantees a solution v_i with

$$\mathbb{E}[f_{S_{i-1}}(v_i)] \geq \max_{v \in V \setminus S_{i-1}} f_{S_{i-1}}(v) - \alpha \tag{7}$$

where $\alpha = 2\lambda \cdot \ln|V|/\varepsilon$.

Let S^* denote any set of size k with $f(S^*) = \mathrm{OPT}$. Below, let us condition on having obtained some set S_{i-1} of elements after the first $i-1$ iterations of our algorithm. Then

$$\begin{aligned}
\mathbb{E}[f_{S_i}(v_i)] &= \max_{v \in V \setminus S_{i-1}} f_{S_{i-1}}(v) - \alpha && \text{(by Condition (7))} \\
&\geq \frac{1}{k}\left(\sum_{v \in S^*} f_{S_{i-1}}(v)\right) - \alpha \\
&\geq \frac{f(S^* \cup S_{i-1}) - f(S_{i-1})}{k} - \alpha && \text{(by submodularity of } f\text{)} \\
&\geq \frac{\mathrm{OPT} - f(S_{i-1})}{k} - \alpha && \text{(by monotonicity of } f\text{)}
\end{aligned}$$

We now unfix from conditioning on having obtained a specific S_{i-1} by taking the expectation over

all choices of such a set. This gives

$$\mathbb{E}[f_{S_{i-1}}(v_i)] \geq \frac{\text{OPT} - \mathbb{E}[f(S_{i-1})]}{k} - \alpha$$

Rearranging yields

$$\text{OPT} - \mathbb{E}[f(S_i)] \leq \left(1 - \frac{1}{k}\right)(\text{OPT} - \mathbb{E}[f(S_{i-1})]) + \alpha$$

Recursively applying this bound yields

$$\begin{aligned}\text{OPT} - \mathbb{E}[f(S_i)] &\leq \left(1 - \frac{1}{k}\right)^i (\text{OPT} - \mathbb{E}[f(S_0)]) + \sum_{j=0}^{i-1} \left(1 - \frac{1}{k}\right)^j \cdot \alpha \\ &\leq \left(1 - \frac{1}{k}\right)^i \text{OPT} + \alpha.\end{aligned}$$

Hence, we conclude

$$\begin{aligned}\mathbb{E}[f(S_k)] &\geq \left[1 - \left(1 - \frac{1}{k}\right)^k\right] \text{OPT} - \alpha \\ &\geq \left(1 - \frac{1}{e}\right) \text{OPT} - \alpha.\end{aligned}$$

□

B.2.2 Proof of Theorem 3.4

Proof. The privacy guarantee of Theorem 3.4 follows from Theorem 3.1.

In our proof of utility, we again suppress the dataset D, and use the notation $f_S(T)$ to denote $f(S \cup T) - f(S)$. Our proof applies to any greedy algorithm that, in each round i, selects an item v_i with

$$\mathbb{E}\left[f_{S_{i-1}}(v_i)\right] \geq \max_{v : S_{i-1} \cup \{v\} \in \mathcal{I}} f_{S_{i-1}}(v_i) - \alpha \tag{8}$$

for some error term $\alpha > 0$.

We follow the proof outlined by Calinescu et al. (2011). Fix an optimal solution $O \in \mathcal{I}$, i.e. $f(O) = \text{OPT}$. Let $S_1, \ldots, S_r$ be any sequence representing the output of the algorithm, where $r = r(\mathcal{I})$. (If the algorithm terminates in an earlier round $k < r$, then extend its output by setting $S_i = S_k$ for each $i = k+1, \ldots, r$.) To find such a sequence, we define a partition $O_1, \ldots, O_r$ of O via Algorithm 17.

Algorithm 17 Partition construction algorithm

Input: Optimal solution O, sets $S_1, \ldots, S_r$
Output: A partition $O_1, O_2, \ldots, O_r$ of O

1. Initialize $T_0 = O$
2. For $i = 1, 2, \ldots, r$:
   (a) If $v_i \in T_{i-1}$, set $O_i = \{v_i\}$;
   Else, let $O_i \subseteq T_{i-1}$ be the smallest subset s.t. $((S_{i-1} \cup T_{i-1}) \setminus O_i) \cup \{v_i\} \in \mathcal{I}$
   (b) Set $T_i = T_{i-1} \backslash O_i$
3. Return $O_1, O_2, \ldots, O_r$

To see that $O_1, \ldots, O_r$ is indeed a partition, observe that $S_i \cup T_i \in \mathcal{I}$ and $S_i \cap T_i = \emptyset$ for every i. Therefore, it must be the case that $T_r = \emptyset$, since $S_r \cup T_r \in \mathcal{I}$ and S_r is maximal when the algorithm terminates. Hence, the disjoint sets $O_1, \ldots, O_r$ do in fact exhaust O.

Lemma B.6. *For every $i = 1, \ldots, r$, we have $\mathbb{E}\left[f_{S_{i-1}}(v_i)\right] \geq \frac{1}{p}\mathbb{E}\left[f_{S_{i-1}}(O_i)\right] - \alpha$.*

Before proving Lemma B.6, we show how to use it to complete the proof of Threorem 3.4. Recursively applying the lemma shows that for every i,

$$\mathbb{E}\left[f(S_i)\right] \geq \frac{1}{p}\sum_{j=1}^{i}\mathbb{E}\left[f_{S_{j-1}}(O_j)\right] - i\alpha.$$

Hence, we obtain

$$\begin{aligned}
\mathbb{E}\left[f(S_r)\right] &\geq \frac{1}{p}\sum_{i=1}^{r}\mathbb{E}\left[f_{S_{i-1}}(O_i)\right] - r\alpha \\
&\geq \frac{1}{p}\sum_{i=1}^{r}\mathbb{E}\left[f_{S_r}(O_i)\right] - r\alpha && \text{(by submodularity)} \\
&\geq \frac{1}{p}\mathbb{E}\left[f_{S_r}(O)\right] - r\alpha && \text{(by linearity of expectation and submodularity)} \\
&\geq \frac{1}{p}\left(f(O) - \mathbb{E}\left[f(S_r)\right]\right) - r\alpha. && \text{(by monotonicity)}
\end{aligned}$$

Rearranging gives the desired result $\mathbb{E}\left[f(S_r)\right] \geq \frac{1}{p+1}f(O) - \frac{p}{p+1}r\alpha$.

□

Proof of Lemma B.6. The partition construction algorithm that every set O_i satisfies $|O_i| \leq p$; this follows from the definition of p-extendibility and the fact that $S_{i-1} \cup \{v_i\} \in \mathcal{I}$. Moreover, any element in O_i is a candidate for inclusion in S_i, since $S_{i-1} \cup \{v\} \in \mathcal{I}$ for every $v \in O_i$.

It is also clear from the partition construction that for each $v \in O_i$, we have $S_{i-1} \cup \{v\} \in \mathcal{I}$. Below,

fix a choice of i and condition on the algorithm's history up to iteration $i-1$. This fixes choices of the sets $S_1, \ldots, S_{i-1}$, as well as $T_1, \ldots, T_i$ and $O_1, \ldots, O_i$.

Then since $\mathbb{E}\left[f_{S_{i-1}}(v_i)\right] \geq f_{S_{i-1}}(v) - \alpha$ for every $v \in O_i$, we have

$$\begin{aligned}\mathbb{E}\left[f_{S_{i-1}}(v_i)\right] &\geq \frac{1}{|O_i|} f_{S_{i-1}}(O_i) - \alpha && \text{(by submodularity)}\\ &\geq \frac{1}{p} f_{S_{i-1}}(O_i) - \alpha.\end{aligned}$$

Taking the expectation over the conditioned event gives the asserted result.

□

B.2.3 Proof of Theorem 3.5

The analysis below will work generally for any random selection procedure guaranteeing that in every round $i = 1, \ldots, k$,

$$\mathbb{E}\left[f_{S_{i-1}}(v_i)\right] \geq \max_{v \in (V_i \cup \{u_i\})} f_{S_{i-1}}(v) - \alpha$$

for some parameter $\alpha > 0$. We begin by fixing an optimal solution S^* with $f(S^*) = \mathrm{OPT}$.

Claim B.7 ((Buchbinder et al., 2014, Observation 3.2))**.** *For every $i = 0, \ldots, k$, we have* $\mathbb{E}\left[f(S^* \cup S_i)\right] \geq (1 - 1/k)^i \cdot \mathrm{OPT}$.

Proof. For every iteration $i = 1, \ldots k$, the subsampling step ensures that every element in $V \cup U$ is selected for inclusion in S_i with probability at most $1/k$. Hence, for every $i = 0, 1, \ldots, k$, each element is included in S_i with probability at most $1 - (1 - 1/k)^i$. Define $g : 2^V \to \mathbb{R}$ by $g(S) = g(S^* \cup S)$. Then g is a submodular function, and

$$\mathbb{E}\left[f(S^* \cup S_i)\right] = \mathbb{E}\left[g(S_i \setminus S^*)\right] \geq (1 - 1/k)^i g(\emptyset) = (1 - 1/k)^i \,\mathrm{OPT}\,.$$

The inequality here follows from the fact that for any set T and any random subset $T' \subseteq T$ that includes every element of T with probability p, we have $\mathbb{E}\left[g(T')\right] \geq (1-p) \cdot g(\emptyset) + p \cdot g(T)$ for any submodular function g Feige et al. (2007).

□

Claim B.8.

$$\mathbb{E}[f_{S_{i-1}}(v_i)] \geq \left(1 - \frac{1}{e}\right) \cdot \left(\frac{\mathbb{E}[f(S^* \cup S_{i-1})] - \mathbb{E}[f(S_{i-1})]}{k}\right) - \alpha.$$

Proof. Begin by conditioning on a fixed choice of the set S_{i-1}. Let $M \subseteq (V \cup U)$ denote a set of k items which maximizes the quantity $\sum_{v \in M} f_{S_{i-1}}(v)$. That is, M consists of the k items in $(V \cup U)$ which result in the largest marginal gain for $f_{S_{i-1}}$.

Let G denote the event that the subsampled set $V_i \cup \{u_i\}$ contains at least one element in M. Observe that even if G does not occur, we have

$$\mathbb{E}\left[f_{S_{i-1}}(v_i) | \overline{G}\right] \geq f_{S_{i-1}}(u_i) - \alpha \geq -\alpha. \tag{9}$$

We claim moreover that

$$\mathbb{E}\left[f_{S_{i-1}}(v_i) | G\right] \geq \frac{1}{k} \sum_{v \in M} f_{S_{i-1}}(v) - \alpha. \tag{10}$$

To see this, sort the items in $V \cup U$ as $v^{(1)}, \ldots, v^{(m)}, v^{(m+1)}, \ldots, v^{(m+k)}$, where $m = |V|$ and $f_{S_{i-1}}(v^{(j)}) \geq f_{S_{i-1}}(v^{(j+1)})$ for every $j = 1, \ldots, m+k-1$. Break ties in such a way that $M = \{v^{(1)}, \ldots, v^{(k)}\}$, and that there is some $t \in \{0, \ldots, k\}$ such that $v^{(1)}, \ldots, v^{(t)} \in V$ and $v^{(t+1)}, \ldots, v^{(k)} \in U$ (that is, real elements come before dummy elements).

Let A_j denote the event that j is the smallest index such that $v^{(j)} \in V_i \cup \{u_i\}$. Then the events $A_1, \ldots, A_{m+k}$ are mutually exclusive and exhaustive. Moreover, by the definition of G, we have $\sum_{j=1}^{k} \Pr[A_j] = \Pr[G]$.

It is easy to see that

$$\begin{aligned}
\Pr[A_1] &= \frac{1}{k} \\
\Pr[A_j] &= \frac{\binom{m-j}{m/k-1}}{\binom{m}{m/k}} && j = 2, \ldots, t, \\
\Pr[A_j] &= \frac{\binom{m-t}{m/k}}{\binom{m}{m/k}} \cdot \frac{1}{k} \cdot \left(1 - \frac{1}{k}\right)^{j-t-1} && j = t+1, \ldots, k.
\end{aligned}$$

Moreover, $\Pr[A_j]$ is a decreasing function $j = 1, \ldots, k$. Hence, $\Pr[A_j|G] = \Pr[A_j]/\Pr[G]$ is a decreasing function of $j = 1, \ldots, k$ as well. Moreover, $\Pr[A_1|G] \geq \Pr[A_1] = 1/k$. This allows us to calculate

$$\begin{aligned}
\mathbb{E}\left[f_{S_{i-1}}(v_i)|G\right] &= \sum_{j=1}^{k} \mathbb{E}\left[f_{S_{i-1}}(v_i)|A_j\right] \cdot \Pr[A_j|G] \\
&\geq \frac{1}{k}\sum_{j=1}^{k}\left(f_{S_{i-1}}(v^{(j)}) - \alpha\right) && \text{(by Chebyshev's sum inequality)} \\
&= \frac{1}{k}\sum_{v\in M} f_{S_{i-1}}(v) - \alpha.
\end{aligned}$$

This establishes the claimed inequality (10).

To estimate $\mathbb{E}\left[f_{S_{i-1}}(v_i)|G\right]$, it remains to calculate $\Pr[G]$. Suppose M consists of t elements from V and $k-t$ dummy elements from U. Then

$$\begin{aligned}
\Pr[G] &= 1 - \Pr[M \cap (V_i \cup \{u_i\}) = \emptyset] \\
&= 1 - \frac{\binom{m-t}{m/k}}{\binom{m}{m/k}} \cdot \frac{t}{k} \\
&= 1 - \frac{(m-(m/k))(m-(m/k)-1)\ldots(m-(m/k)-t+1)}{m(m-1)\ldots(m-t+1)} \cdot \frac{t}{k} \\
&= 1 - \left(1-\frac{1}{k}\right)\left(1-\frac{1}{k}\cdot\frac{m}{m-1}\right)\ldots\left(1-\frac{1}{k}\cdot\frac{m}{m-t+1}\right)\cdot\frac{t}{k} \\
&\geq 1 - \left(1-\frac{1}{k}\right)^t \cdot \frac{t}{k} \\
&\geq 1 - \frac{te^{-t/k}}{k} \\
&\geq 1 - \frac{1}{e},
\end{aligned}$$

where the last inequality follows from the fact that the function $r(x) = xe^{-x}$ is maximized at $x = 1$, where it takes the value $1/e$.

Let M' be the set containing $S^* \setminus S_{i-1}$ together with enough dummy elements to have size exactly

k. We conclude that

$$\begin{aligned}
\mathbb{E}\left[f_{S_{i-1}}(v_i)\right] &= \Pr[G] \cdot \mathbb{E}\left[f_{S_{i-1}}(v_i)|G\right] + (1 - \Pr[G]) \cdot \mathbb{E}\left[f_{S_{i-1}}(v_i)|\bar{G}\right] \\
&\geq \left(1 - \frac{1}{e}\right)\left(\frac{1}{k}\sum_{v \in M} f_{S_{i-1}}(v) - \alpha\right) - \frac{1}{e} \cdot \alpha && \text{(by (10) and (9))} \\
&\geq \left(1 - \frac{1}{e}\right)\left(\frac{1}{k}\sum_{v \in M'} f_{S_{i-1}}(v)\right) - \alpha && \text{(by definition of } M\text{)} \\
&\geq \left(1 - \frac{1}{e}\right)\left(\frac{f(S^* \cup S_{i-1}) - f(S_{i-1})}{k}\right) - \alpha. && \text{(by submodularity)}
\end{aligned}$$

Unconditioning from S_{i-1} by taking the expectation over its choice proves the claim. □

Proof of Theorem 3.5. Let f be any (possibly non-monotone) submodular function. We show by induction that for every $i = 0, \ldots, k$, we have

$$\mathbb{E}\left[f(S_i)\right] \geq \left(1 - \frac{1}{e}\right) \cdot \frac{i}{k} \cdot \left(1 - \frac{1}{k}\right)^{i-1} \cdot \mathrm{OPT} - i\alpha. \tag{11}$$

This clearly holds for the base case of $i = 0$. Assuming it holds in iteration $i - 1$, we calculate

$$\begin{aligned}
\mathbb{E}[f(S_i)] &= \mathbb{E}[f(S_{i-1})] + \mathbb{E}[f_{v_i}(S_{i-1})] \\
&\geq \mathbb{E}[f(S_{i-1})] + \left(1 - \frac{1}{e}\right)\left(\frac{\mathbb{E}[f(S^* \cup S_{i-1})] - \mathbb{E}[f(S_{i-1})]}{k}\right) - \alpha && \text{(by Claim B.8)} \\
&\geq \mathbb{E}[f(S_{i-1})] + \left(1 - \frac{1}{e}\right)\left(\frac{(1 - \frac{1}{k})^{i-1}\,\mathrm{OPT} - \mathbb{E}[f(S_{i-1})]}{k}\right) - \alpha && \text{(by Claim B.7)} \\
&= \left(1 - \frac{1}{k}\right)\mathbb{E}[f(S_{i-1})] + \left(1 - \frac{1}{e}\right) \cdot \left(1 - \frac{1}{k}\right)^{i-1} \cdot \frac{1}{k} \cdot \mathrm{OPT} - \alpha \\
&\geq \left(1 - \frac{1}{k}\right)\left(1 - \frac{1}{e}\right) \cdot \frac{i-1}{k} \cdot \left(1 - \frac{1}{k}\right)^{i-2} \cdot \mathrm{OPT} + \left(1 - \frac{1}{e}\right) \cdot \left(1 - \frac{1}{k}\right)^{i-1} \cdot \frac{1}{k} \cdot \mathrm{OPT} - i\alpha \\
&\qquad\qquad \text{(by the inductive hypothesis)} \\
&= \left(1 - \frac{1}{e}\right) \cdot \frac{i}{k} \cdot \left(1 - \frac{1}{k}\right)^{i-1} \cdot \mathrm{OPT} - i\alpha.
\end{aligned}$$

Hence, in iteration k, we have

$$\mathbb{E}\left[f(S_k)\right] \geq \left(1 - \frac{1}{e}\right) \cdot \left(1 - \frac{1}{k}\right)^{k-1} \cdot \mathrm{OPT} - k\alpha \geq \left(1 - \frac{1}{e}\right) \cdot \frac{1}{e} \cdot \mathrm{OPT} - k\alpha$$

as we wanted to show.

Now we consider the special case where f is monotone. In this case, we have

$$\begin{aligned}\mathbb{E}[f_{v_i}(S_{i-1})] &\geq \left(1-\frac{1}{e}\right)\left(\frac{\mathbb{E}[f(S^* \cup S_{i-1})] - \mathbb{E}[f(S_{i-1})]}{k}\right) - \alpha && \text{(by Claim B.8)}\\ &\geq \left(1-\frac{1}{e}\right)\left(\frac{\mathrm{OPT} - \mathbb{E}[f(S_{i-1})]}{k}\right) - \alpha && \text{(by monotonicity).}\end{aligned}$$

Rearranging gives us

$$\mathrm{OPT} - \mathbb{E}\left[f(S_i)\right] \leq \left(1 - \frac{(1-1/e)}{k}\right) \cdot \left(\mathrm{OPT} - \mathbb{E}\left[f(S_{i-1})\right]\right) + \alpha.$$

Recursively applying this bound yields

$$\begin{aligned}\mathrm{OPT} - \mathbb{E}[f(S_i)] &\leq \left(1 - \frac{(1-1/e)}{k}\right)^i \left(\mathrm{OPT} - \mathbb{E}[f(S_0)]\right) + \sum_{j=0}^{i-1}\left(1-\frac{1}{k}\right)^j \alpha\\ &\leq \left(1 - \frac{(1-1/e)}{k}\right)^i \mathrm{OPT} + i\alpha.\end{aligned}$$

Hence, we conclude

$$\begin{aligned}\mathbb{E}[f(S_k)] &\geq \left[1 - \left(1 - \frac{(1-1/e)}{k}\right)^k\right]\mathrm{OPT} - k\alpha\\ &\geq \left(1 - e^{-(1-1/e)}\right)\mathrm{OPT} - k\alpha.\end{aligned}$$

□

Appendix C. Two-Stage Submodular Maximization Appendix

C.1 Proof of Theorem 4.1

Let S^t represent the set of chosen elements at step t. Also, we define $T_i^t \subseteq S^t$ as the current solution for function f_i at step t. We also define $A_i^t = \bigcup_{1\leq j\leq t} T_i^t$, i.e., A_i^t is the set of all the elements have been in the set T_i till step t. Note that this set includes elements that have been in T_i at some point and might be deleted at later steps. We first lower bound $f_i(T_i^t)$ based on value of $f_i(A_i^t)$.

Lemma C.1. *For all $1 \leq i \leq m$, we have*

$$f_i(T_i^t) \geq \frac{\alpha}{\alpha+1} f_i(A_i^t).$$

Proof. We proof this lemma by induction. For the first k additions to set T_i^t, the two sets T_i^t and A_i^t are exactly the same, i.e., we have $f_i(T_i^t) = f_i(A_i^t)$. Therefore the lemma is correct for them. Next we show that lemma is correct for cases after the first k additions, i.e., when an incoming element u^t replaces one element of T_i^{t-1}. We have the following lemma.

Lemma C.2. *For $1 \leq i \leq m$ and all u^t, we have:*

$$\Delta_i(u^t, T_i^{t-1}) \geq f_i(u^t|A_i^{t-1}) - f_i(T_i^{t-1})/k.$$

Proof. To prove this lemma we have the following

$$\begin{aligned}
\Delta_i(u^t, T_i^{t-1}) &= f_i(T_i^{t-1} + u^t - \textsc{Rep}_i(u^t, T_i^{t-1})) - f_i(T_i^{t-1}) \\
&\overset{(a)}{\geq} \frac{\sum_{u \in T_i^{t-1}} f_i(T_i^{t-1} + u^t - u)) - f_i(T_i^{t-1})}{k} \\
&= \frac{\sum_{u \in T_i^{t-1}} f_i(T_i^{t-1} + u^t - u) - f_i(T_i^{t-1} - u) + f_i(T_i^{t-1} - u) - f_i(T_i^{t-1})}{k} \\
&\overset{(b)}{\geq} \frac{\sum_{u \in T_i^{t-1}} f_i(T_i^{t-1} + u^t) - f_i(T_i^{t-1})}{k} + \frac{\sum_{u \in T_i^{t-1}} f_i(T_i^{t-1} - u) - f_i(T_i^{t-1})}{k} \\
&\overset{(c)}{\geq} f_i(u^t|T_i^{t-1}) - f_i(T_i^{t-1})/k \overset{(d)}{\geq} f_i(u^t|A_i^{t-1}) - f_i(T_i^{t-1})/k.
\end{aligned}$$

Inequality (a) is true because $\textsc{Rep}_i(u^t, T_i^{t-1})$ is the element with the largest increment when it is exchanged with u^t. Therefore, it should be at least equal to the average of all possible exchanges. Note that T_i^{t-1} has at most k elements. Inequalities (b) and (d) result from submodularity of f_i. Also, from submodularity of f_i, we have $f_i(T_i^{t-1}) - f_i(\emptyset) \geq \sum_{u \in T_i^{t-1}} f_i(T_i^{t-1}) - f_i(T_i^{t-1} - u)$ which results in inequality (c). □

Now, assume Lemma C.1 is true for time $t-1$, i.e., $f_i(T_i^{t-1}) \geq \frac{\alpha}{\alpha+1} f_i(A_i^{t-1})$. We prove that it is also true for time t. First note that if u^t is not accepted by the algorithm for the i-th function then $T_i^t = T_i^{t-1}$ and $A_i^t = A_i^{t-1}$; therefore the lemma is true for t. If u^t is chosen to be added to T_i^{t-1}, from the definition of $\nabla(u^t, T_i^{t-1})$, we have $\Delta_i(u^t, T_i^{t-1}) > \alpha/k \cdot f_i(T_i^{t-1})$. From this fact and Lemma C.2, we have:

$$\begin{aligned}
f_i(T_i^t) - f_i(T_i^{t-1}) &\geq \max\{f_i(u^t|A_i^{t-1}) - f_i(T_i^{t-1})/k, \alpha/k \cdot f_i(T_i^{t-1})\} \\
&\geq \frac{\alpha \cdot (f_i(u^t|A_i^{t-1}) - f_i(T_i^{t-1})/k) + \alpha/k \cdot f_i(T_i^{t-1})}{\alpha+1} \\
&\geq \frac{\alpha}{\alpha+1} \cdot f_i(u^t|A_i^{t-1}) = \frac{\alpha}{\alpha+1} \cdot \left[f_i(A_i^t) - f_i(A_i^{t-1})\right] \rightarrow f_i(T_i^t) \geq \frac{\alpha}{\alpha+1} \cdot f_i(A_i^t).
\end{aligned}$$

□

Corollary 1. *If* $\Delta_i(u^t, T_i^{t-1}) < \alpha/k \cdot f_i(T_i^{t-1})$ *then we have:*

$$f_i(u^t|A_i^n) \overset{(a)}{\leq} f_i(u^t|A_i^{t-1}) \overset{(b)}{\leq} \frac{\alpha+1}{k} \cdot f_i(T_i^{t-1}) \overset{(c)}{\leq} \frac{\alpha+1}{k} \cdot f_i(T_i^n).$$

Proof. Inequality (a) is true because of submodularity of f_i and the fact that $A_i^{t-1} \subseteq A_i^n$. Inequality (b) concludes form Lemma C.2. Since $f_i(T_i^t)$ is a nondecreasing function of t, then (c) is true. □

Next, we use Lemmas C.1 and C.2 and Corollary 1, to prove the approximation factor of the algorithm. Note that if at the end of algorithm $|S^n| = \ell$, then we have:

$$\frac{1}{m}\sum_{i=1}^{m} f_i(T_i^n) = \frac{1}{m}\sum_{t=1}^{n}\sum_{i=1}^{m}\left[f_i(T_i^t) - f_i(T_i^{t-1})\right] = \frac{1}{m}\sum_{t=1}^{n} \mathbb{1}_{\{u^t \in S^n\}} \cdot \nabla_i(u^t, T_i^t) \geq \frac{\mathrm{OPT}}{\beta}. \tag{12}$$

This is true because the additive value after adding an element to S^t is at least $\frac{\mathrm{OPT}}{\beta\ell}$. Next consider the case where $|S| < \ell$. First note that for an element $u^t \in S_i^{m,\ell}$, which does not belong to set A_i^n, we have two different possibilities: (i) $\Delta_i(u^t, T_i^{t-1}) < \alpha/k \cdot f_i(T_i^{t-1})$, or (ii) $\Delta_i(u^t, T_i^{t-1}) \geq \alpha/k \cdot f_i(T_i^{t-1})$ and $\frac{1}{m}\sum_{i=1}^{m} \nabla_i(u^t, T_i^{t-1}) < \frac{\mathrm{OPT}}{\beta\ell}$. Therefore, we have

$$\begin{aligned}
\sum_{i=1}^{m} f_i(S_i^{m,\ell}) &\leq \sum_{i=1}^{m}\left[f_i(A_i^n) + \sum_{u^t \in S_i^{m,\ell} \setminus A_i^n} f_i(u^t|A_i^n)\right] \\
&= \sum_{i=1}^{m} f_i(A_i^n) + \sum_{i=1}^{m}\sum_{u^t \in S^{m,\ell} \setminus A_i^n} \mathbb{1}_{\{u^t \in S_i^{m,\ell}\}} \cdot f(u^t|A_i^n) \\
&= \sum_{i=1}^{m} f_i(A_i^n) + \sum_{i=1}^{m}\sum_{u^t \in S^{m,\ell}} \mathbb{1}_{\{u^t \in S_i^{m,\ell}\}} \cdot \\
&\Big[\mathbb{1}_{\{\Delta_i(u^t, T_i^{t-1}) < \alpha/k \cdot f_i(T_i^{t-1})\}} \cdot f_i(u^t|A_i^n) + \\
&\mathbb{1}_{\{\Delta_i(u^t, T_i^{t-1}) \geq \alpha/k \cdot f_i(T_i^{t-1}) \text{ and } \sum_{i=1}^{m} \nabla_i(u^t, T_i^{t-1}) < \frac{\mathrm{OPT}}{\beta\ell}\}} \cdot f_i(u^t|A_i^n)\Big].
\end{aligned} \tag{13}$$

For the three terms on the rightmost side of Equation 13 we have the following inequalities. For the first term, from Lemma C.1, we have:

$$\sum_{i=1}^{m} f_i(A_i^n) \leq \frac{\alpha+1}{\alpha}\sum_{i=1}^{m} f_i(T_i^n). \tag{14}$$

For the second term, we have:

$$\sum_{i=1}^{m} \sum_{u^t \in S^{m,\ell}} \mathbb{1}_{\{u^t \in S_i^{m,\ell}\}} \cdot \mathbb{1}_{\{\Delta_i(u^t, T_i^{t-1}) < \alpha/k \cdot f_i(T_i^{t-1})\}} \cdot f_i(u^t | A_i^n) \\ \overset{(a)}{\leq} \sum_{i=1}^{m} \sum_{u^t \in S_i^{m,\ell}} \frac{\alpha+1}{k} f_i(T_i^n) \overset{(b)}{\leq} (\alpha+1) \cdot \sum_{i=1}^{m} f_i(T_i^n). \tag{15}$$

Inequality (a) is the result of Corollary 1. Inequality (b) is true because we have at most k elements in set $S_i^{m,\ell}$. Note that for u^t with $\sum_{i=1}^{m} \nabla_i(u^t, T_i^{t-1}) < \frac{\text{OPT}}{\beta\ell}$ we have:

$$\frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{\{u^t \in S_i^{m,\ell}\}} \cdot \mathbb{1}_{\{\Delta_i(u^t, T_i^{t-1}) \geq \alpha/k \cdot f_i(T_i^{t-1})\}} \left[f_i(u^t | A_i^{t-1}) - f_i(T_i^{t-1})/k \right] \\ \overset{(a)}{\leq} \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{\{u^t \in S_i^{m,\ell}\}} \nabla_i(u^t, T_i^{t-1}) \overset{(b)}{\leq} \frac{1}{m} \sum_{i=1}^{m} \nabla_i(u^t, T_i^{t-1}) \\ < \frac{\text{OPT}}{\beta\ell}. \tag{16}$$

Inequality (a) results from Lemma C.2 and (b) is true because $\nabla_i(u^t, T_i^{t-1}) \geq 0$ for $1 \leq i \leq m$. Therefore, from Equation 16 and submodularity of f_i and its non-negativity, we have:

$$\frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{\{u^t \in S_i^{m,\ell}\}} \cdot \mathbb{1}_{\{\Delta_i(u^t, T_i^{t-1}) \geq \alpha/k \cdot f_i(T_i^{t-1}) \text{ and } \sum_{i=1}^{m} \nabla_i(u^t, T_i^{t-1}) < \frac{\text{OPT}}{\beta\ell}\}} \cdot f_i(u^t | A_i^n) \\ \leq \frac{\text{OPT}}{\beta\ell} + \frac{1}{km} \cdot \sum_{i=1}^{m} \mathbb{1}_{\{u^t \in S_i^{m,\ell}\}} \cdot f_i(T_i^n).$$

Consequently,

$$\frac{1}{m} \sum_{i=1}^{m} \sum_{u^t \in S^{m,\ell}} \mathbb{1}_{\{u^t \in S_i^{m,\ell}\}} \cdot \mathbb{1}_{\{\Delta_i(u^t, T_i^{t-1}) \geq \alpha/k \cdot f_i(T_i^{t-1}) \text{ and } \sum_{i=1}^{m} \nabla_i(u^t, T_i^{t-1}) < \frac{\text{OPT}}{2\ell}\}} \cdot f_i(u^t | A_i^n) \\ \leq \frac{1}{m} \sum_{u^t \in S^{m,\ell}} \left[\frac{\text{OPT}}{\beta\ell} + \frac{1}{k} \cdot \sum_{i=1}^{m} \mathbb{1}_{\{u^t \in S_i^{m,\ell}\}} \cdot f_i(T_i^n) \right] \leq \frac{\text{OPT}}{\beta} + \frac{1}{m} \sum_{i=1}^{m} f_i(T_i^n). \tag{17}$$

Using Equations 14, 15, and 17 we have:

$$\text{OPT} = \frac{1}{m} \sum_{i=1}^{m} f_i(S_i^{m,\ell}) \leq \frac{\alpha+1}{\alpha} \cdot \frac{1}{m} \sum_{i=1}^{m} f_i(T_i^n) + (\alpha+1) \cdot \frac{1}{m} \sum_{i=1}^{m} f_i(T_i^n) + \frac{\text{OPT}}{\beta} + \frac{1}{m} \sum_{i=1}^{m} f_i(T_i^n). \tag{18}$$

This results in

$$\frac{\alpha \cdot (\beta - 1) \cdot \text{OPT}}{\beta \cdot ((\alpha + 1)^2 + \alpha)} \leq \frac{1}{m} \sum_{i=1}^{m} f_i(T_i^n). \tag{19}$$

Combination of Equations 12 and 19 proves the theorem.

C.2 Proof of Theorem 4.4

We first prove Lemmas 4.2 and 4.3.

Proof of Lemma 4.2: The lower bound is trivial. For the upper bound we have

$$\text{OPT} = \frac{1}{m} \sum_{i=1}^{m} \sum_{u \in S_i^{m,\ell}} f_i(u) \leq \frac{1}{m} \sum_{u \in S^{m,\ell}} \sum_{i=1}^{m} f_i(u) \leq \ell \cdot \delta.$$

Proof of Lemma 4.3: We have

$$\frac{1}{m} \sum_{i=1}^{m} \nabla_i(u^t, T_i^{t-1}) \overset{(a)}{\leq} \frac{1}{m} \sum_{i=1}^{m} f_i(u^t | T_i^{t-1}) \overset{(b)}{\leq} \frac{1}{m} \sum_{i=1}^{m} f_i(u^t) \overset{(c)}{\leq} \delta_t.$$

For inequality (a) first note that $f_i(u^t|T_i^{t-1}) \geq 0$; therefore it suffices to show that for all $\nabla_i(u^t, T_i^{t-1}) > 0$ we have $\nabla_i(u^t, T_i^{t-1}) \leq f_i(u^t|T_i^{t-1})$. So, for $\nabla_i(u^t, T_i^{t-1}) > 0$, consider the two following cases: (i) if $|T_i^{t-1}| < k$, then $\nabla_i(u^t, T_i^{t-1}) = f_i(u^t|T_i^{t-1})$. (ii) if $|T_i^{t-1}| < k$, then $\nabla_i(u^t, T_i^{t-1}) = \Delta_i(u^t, T_i^{t-1}) = f_i(T_i^{t-1} + u^t - \textsc{Rep}_i(u^t, T_i^{t-1})) - f_i(T_i^{t-1}) \leq f_(T_i^{t-1} + u^t) - f_i(T_i^{t-1})$, where the last inequality follows from the monotonicity of f_i. Inequality (b) results from the submodularity of f_i. The inequality (c) follows from the definition of δ_t.

Proof of Theorem 4.4: Note that there exists an instance of algorithm with a threshold τ in Γ^n such that $\frac{\text{OPT}}{1+\epsilon} \leq \tau_l \leq \text{OPT}$. For this instance, it suffices to replace OPT with $\frac{\text{OPT}}{1+\epsilon}$ in the proof of Theorem 4.1. This proves the approximation guarantee of the theorem. For each instance of the algorithm we keep at most ℓ items. Since we have $O(\frac{\log \ell}{\epsilon})$ thresholds, the total memory complexity of the algorithm is $O(\frac{\ell \log \ell}{\epsilon})$. The update time per each element u^t for each instance is $O(km)$. This is true because we compute the gain of exchanging u^t with all the k elements of T_i^{t-1} for each function $f_i, 1 \leq i \leq m$. Therefore, the total update time per elements is $O(\frac{km \log \ell}{\epsilon})$.

C.3 Proof of Theorem 4.5

First recall that we defined:

$$S^{m,\ell} = \operatorname*{arg\,max}_{S\subseteq\Omega,|S|\leq\ell} \frac{1}{m}\sum_{i=1}^{m} \max_{|T|\leq k, T\subseteq S} f_i(T),$$

and

$$S_i^{m,\ell} = \operatorname*{arg\,max}_{S\subseteq S^{m,\ell},|S|\leq k} f_i(S) \text{ and } \mathrm{OPT} = \frac{1}{m}\sum_{i=1}^{m} f_i(S_i^{m,\ell}).$$

Let $\mathcal{V}(1/\mathsf{M})$ denote the distribution over random subsets of Ω where each element is picked independently with a probability $\frac{1}{\mathsf{M}}$. Define vector $\boldsymbol{p}\in[0,1]^n$ such that for $e\in\Omega$, we have

$$\boldsymbol{p}_e = \begin{cases} \mathbb{P}_{A\sim\mathcal{V}(1/\mathsf{M})}[e\in \textsc{Replacement-Greedy}(A\cup\{e\})] \text{ if } e\in S^{m,\ell}, \\ 0 \qquad \text{otherwise.} \end{cases}$$

We also define vector $\boldsymbol{p_i}$ such that for $e\in V$, we have:

$$\boldsymbol{p_i}_e = \begin{cases} \boldsymbol{p}_e \text{ if } e\in S_i^{m,\ell}, \\ 0 \qquad \text{otherwise.} \end{cases}$$

Denote by V^l the set of elements assigned to machine l. Also, let $O^l = \{e\in S^{m,\ell} : e\notin \textsc{Replacement-Greedy}(V^l\cup\{e\})\}$. Furthermore, define $O_i^l = O^l\cap S_i^{m,\ell}$. The next lemma plays a crucial role in proving the approximation guarantee of our algorithm.

Lemma C.3. *Let $A\subseteq\Omega$ and $B\subseteq\Omega$ be two disjoint subsets of Ω. Suppose for each element $e\in B$, we have* $\textsc{Replacement-Greedy}(A\cup\{e\}) = \textsc{Replacement-Greedy}(A)$. *Then we have:*

$$\textsc{Replacement-Greedy}(A\cup B) = \textsc{Replacement-Greedy}(A).$$

Proof. We proof lemma by contradiction. Assume

$$\textsc{Replacement-Greedy}(A\cup B) \neq \textsc{Replacement-Greedy}(A).$$

At each iteration the element with the highest additive value is added to set S. In Replacement-

GREEDY, the additive value of each element depends on sets $T_i \subseteq S$. Note that sets $T_i \subseteq S$ are deterministic functions of elements of S while considering their order of additions to S. Let's assume e is the first element such that REPLACEMENT-GREEDY$(A \cup B) \neq$ REPLACEMENT-GREEDY(A). First note that $e \notin A$. Also, we conclude REPLACEMENT-GREEDY$(A \cup \{e\}) \neq$ REPLACEMENT-GREEDY(A). This contradicts with the assumption of lemma. □

From the definition of set O^l and Lemma C.3, we have:

$$\text{REPLACEMENT-GREEDY}(V^l) = \text{REPLACEMENT-GREEDY}(V^l \cup O^l).$$

Lemma C.4. *We have:*

$$\frac{1}{m}\sum_{i=1}^{m} f_i(T_i^l) \geq \alpha \cdot \frac{1}{m}\sum_{i=1}^{m} f_i(O_i^l),$$

where α is the approximation factor of REPLACEMENT-GREEDY*.*

Proof. Let OPT_i^l denote the optimum value for function f_i on the dataset $V^l \cup O^l$ for the two-stage submodular maximization problem. We have:

$$\frac{1}{m}\sum_{i=1}^{m} f_i(T_i^l) \geq \alpha \cdot \frac{1}{m}\sum_{i=1}^{m} \text{OPT}_i^l \geq \alpha \cdot \frac{1}{m}\sum_{i=1}^{m} f_i(O_i^l).$$

□

This is true because (i) REPLACEMENT-GREEDY$(V^l) =$ REPLACEMENT-GREEDY$(V^l \cup O^l)$, (ii) approximation guarantee of REPLACEMENT-GREEDY is α, and (iii) O^l and $\{O_i^l\}$ is a valid solution for the two-stage submodular maximization problem over set $V^l \cup O^l$. Assume f_i^- is the Lovász extension of a submodular function f_i.

Lemma C.5 (Lemma 1, Barbosa et al. (2015))**.** *Let A be random set, and suppose that $\mathbb{E}[\mathbf{1}_A] = \lambda \cdot \boldsymbol{p}$ for a constant value of $\lambda \in [0, 1]$. Then, $\mathbb{E}[f(S)] \geq \lambda \cdot f^-(\boldsymbol{p})$.*

For each element $e \in S^{m,\ell}$ we have:

$$\mathbb{P}[e \in O^l] = 1 - \mathbb{P}[e \notin O^l] = 1 - \boldsymbol{p}_e,$$

$$\mathbb{E}[\mathbf{1}_{O^l}] = \mathbf{1}_{S^{m,\ell}} - \boldsymbol{p},$$

$$\mathbb{E}[\mathbf{1}_{O_i^l}] = \mathbf{1}_{S_i^{m,\ell}} - \boldsymbol{p_i}.$$

Therefore, we have:

$$\mathbb{E}[\frac{1}{m}\sum_{i=1}^{m} f_i(T_i^l)] \geq \alpha \cdot \mathbb{E}[\frac{1}{m}\sum_{i=1}^{m} f_i(O_i^l)] \geq \frac{\alpha}{m} \cdot \sum_{i=1}^{m} f_i^-(\mathbf{1}_{S_i^{m,\ell}} - \boldsymbol{p_i}).$$

Furthermore, for each element $e \in S^{m,\ell}$ we have

$$\begin{aligned}\mathbb{P}[e \in \bigcup_l S^l | e \text{ is assigned to machine } l] &= \mathbb{P}[e \in \textsc{Replacement-Greedy}(V^l) | e \in V^l] \\ &= \mathbb{P}_{A \sim \mathcal{V}(1/\mathsf{M})}[e \in \textsc{Replacement-Greedy}(A) | e \in A] \\ &= \mathbb{P}_{B \sim \mathcal{V}(1/\mathsf{M})}[e \in \textsc{Replacement-Greedy}(B \cup \{e\})] \\ &= \boldsymbol{p}_e.\end{aligned}$$

Therefore, we have

$$\mathbb{E}[\frac{1}{m}\sum_{i=1}^{m} f_j(T_i')] \geq \alpha \cdot \mathbb{E}[\frac{1}{m}\sum_{i=1}^{m} f_i(\bigcup_l S^l \cap S_i^{m,\ell})] \geq \frac{\alpha}{m} \cdot \sum_{i=1}^{m} f_i^-(\boldsymbol{p_i})$$

To Sum up above, we have:

$$\mathbb{E}[\frac{1}{m}\sum_{i=1}^{m} f_j(T_i^*)] \geq \frac{\alpha}{m}\sum_{i=1}^{m} f_j^-(\mathbf{1}_{S_i^{m,\ell}} - \boldsymbol{p_i}), \tag{20}$$

$$\mathbb{E}[\frac{1}{m}\sum_{i=1}^{m} f_i(T_i^*)] \geq \frac{\alpha}{m}\sum_{i=1}^{m} f_i^-(\boldsymbol{p_i}). \tag{21}$$

And therefore we have:

$$\mathbb{E}[\frac{1}{m}\sum_{i=1}^{m} f_i(T_i^*)] \geq \frac{\alpha}{2m}\sum_{i=1}^{m}\left[f_i^-(\boldsymbol{p_i}) + f_i^-(\mathbf{1}_{S_i^{m,\ell}} - \boldsymbol{p_i})\right] \overset{(a)}{\geq} \frac{\alpha}{2m}\sum_{i=1}^{m} f_i^-(\mathbf{1}_{S_i^{m,\ell}}) \geq \frac{\alpha}{2m}\sum_{i=1}^{m} f_i(S_i^{m,\ell}).$$

The inequality (a) results from the convexity of Lovász extensions for submodular functions. Note that the approximation guarantee of Replacement-Greedy is $\alpha = \frac{1}{2}(1-\frac{1}{e^2})$ (Stan et al., 2017).

C.4 Proof of Theorem 4.6

In this section, we first outline Distributed-Fast (Algorithm 18) and then prove Theorem 4.6.

Algorithm 18 DISTRIBUTED-FAST

For $1 \leq l \leq \mathsf{M}$ set $V^l = \emptyset$
for $e \in \Omega$ **do**
    Assign e to a set V^l chosen uniformly at random
For $1 \leq l \leq \mathsf{M}$ sort elements of V^l based on a universal predefined ordering between elements
▷ Any consistent ordering between elements of Ω is valid.
Let V^l be the elements assigned to machine l
Run REPLACEMENT-PSEUDO-STREAMING on each machine l to obtain $\{S^l_\tau\}$ and $\{T^l_{\tau,i}\}$ for $1 \leq i \leq m$ and relevant values of τ on that machine
$l^*, \tau^* \leftarrow \arg\max_{l,\tau} \frac{1}{m} \sum_{i=1}^{m} f_i(T^l_{\tau,i})$
$S, \{T_i\} \leftarrow$ REPLACEMENT-GREEDY$(\bigcup_l \bigcup_\tau S^l_\tau)$
Return: $\arg\max\{\frac{1}{m} \sum_{i=1}^{m} f_i(T_i), \frac{1}{m} \sum_{i=1}^{m} f_i(T^{l^*}_{\tau^* i})\}$

The following lemma provides the equivalent of Lemma C.3 for REPLACEMENT-PSEUDO-STREAMING. The rest of proof is exactly the same as the proof of Theorem 4.5 with the only difference that the approximation guarantee of REPLACEMENT-PSEUDO-STREAMING is $\gamma = \frac{1}{6+\epsilon}$.

Lemma C.6. *Let $A \subseteq \Omega$ and $B \subseteq \Omega$ be two disjoint subsets of Ω. Suppose for each element $e \in B$, we have* REPLACEMENT-PSEUDO-STREAMING$(A \cup \{e\})$ = REPLACEMENT-PSEUDO-STREAMING(A). *Then we have* REPLACEMENT-PSEUDO-STREAMING$(A \cup B)$ = REPLACEMENT-PSEUDO-STREAMING(A).

Proof. First note that because of the universal predefined ordering between elements of Ω, the order of processing the elements would not change in different runs of REPLACEMENT-PSEUDO-STREAMING. Also, in the streaming setting, if an element u^t changes the set of thresholds Γ^t, then u^t would be picked by those newly instantiated thresholds. To show this, assume $\delta_{t-1} < \tau \leq \delta_t$ is one of the newly instantiated thresholds. For τ, the sets $\{T_{\tau,i}\}$ are empty and we have:

$$\tau \leq \sum_{i=1}^{m} \nabla_i(u^t|\emptyset) = \sum_{i=1}^{m} f_i(u^t) = \delta_t.$$

Therefore, u^t is added to all sets $\{T_{\tau,i}\}$. For an element $e \in B$, we have two cases: (i) e has not changed the thresholds when it is arrived, or (ii) it has instantiated new thresholds (e.g., a new threshold τ) but non of them is in the final thresholds Γ^n; because if $\tau \in \Gamma^n$, then we have $e \in S^n_\tau$, and this contradicts with the definition of set B.

Now consider REPLACEMENT-PSEUDO-STREAMING$(A \cup B)$. We prove the lemma by contradiction.

Assume

$$\textsc{Replacement-Pseudo-Streaming}(A \cup B) \neq \textsc{Replacement-Pseudo-Streaming}(A).$$

Assume e is the first element of B which is picked by $\textsc{Replacement-Pseudo-Streaming}(A \cup B)$ for a threshold in Γ^n. From the above, we know that non of the thresholds Γ^n of this running instance of the algorithm is instantiated when an element of B is arrived. So, when e is arrived, all the thresholds of Γ^n which are instantiated so far are from elements of A. Also, since the order of processing of elements are fixed, $\textsc{Replacement-Pseudo-Streaming}(A \cup B)$ and $\textsc{Replacement-Pseudo-Streaming}(A \cup \{e\})$ would pick the same set of element till the point e is arrived. If e is picked by $\textsc{Replacement-Pseudo-Streaming}(A \cup B)$ for a threshold $\tau \in \Gamma^n$, then $\textsc{Replacement-Pseudo-Streaming}(A \cup \{e\})$ would also pick e for that threshold. This contradicts with the definition of set B. □

C.5 Replacement-Greedy

In this section, in order to make the current manuscript self-contained, we describe the Replacement-Greedy from (Stan et al., 2017). We use this greedy algorithm in Section 4.1.7 as one of the building blocks of our distributed algorithms.

We first define few necessary notations. The additive value of an element x to a set A from a function f_i is defined as follows:

$$\Lambda_i(x, A) = \begin{cases} f_i(x|A) & \text{if } |A| < k, \\ \max\{0, \Delta_i(x, A)\} & \text{o.w.}, \end{cases}$$

where $\Delta_i(x, A)$ is defined in Equation 4. We also define:

$$\textsc{Rep-Greedy}_i(x, A) = \begin{cases} \emptyset & \text{if } |A| < k, \\ \emptyset & \Delta_i(x, A) < 0, \\ \textsc{Rep}_i(x, A) & \text{o.w.}, \end{cases}$$

where $\textsc{Rep}_i(x, A)$ is defined in Equation 3. Indeed, $\textsc{Rep-Greedy}_i(x, A)$ represents the element from set A which should be replaced with x in order to get the maximum (positive) additive gain, where the cardinality constraint k is satisfied. Replacement-Greedy starts with empty sets S

and $\{T_i\}$. In ℓ rounds, it greedily adds elements with the maximum additive gains $\sum_{i=1}^{m} \Lambda_i(x, T_i)$ to set S. If the gain of adding these elements (or exchanging with one element of T_i where there exists k elements in T_i) is non-negative, we also update sets T_i. Replacement-Greedy is outlined in Algorithm 19.

Algorithm 19 Replacement-Greedy

$S \leftarrow \emptyset$ and $T_i \leftarrow \emptyset$ for all $1 \leq i \leq m$
for $1 \leq j \leq \ell$ **do**
    $x^* \geq \arg\max_{x \in \Omega} \sum_{i=1}^{m} \Lambda_i(x, T_i)$
    $S \leftarrow S + x^*$
    for $1 \leq i \leq m$ **do**
        if $\Lambda_i(x^*, T_i) > 0$ **then**
            $T_i \leftarrow T_i + x^* - \text{Rep-Greedy}_i(x^*, T_i)$
Return: S and $\{T_i\}$

Appendix D. Submodular Streaming Appendix

D.1 Implications of Sieve-Streaming++

Recently, there has been several successful instances of using the idea of (Badanidiyuru et al., 2014) for designing streaming algorithms for a wide range of submodular maximization problems. In Section 4.2.3, we showed Sieve-Streaming++ (see Algorithm 9 and Theorem 4.7) reduces the memory complexity of streaming submodular maximization to $O(k)$. In this section, we discuss how the approach of Sieve-Streaming++ significantly improves the memory complexity for several important problems.

Random Order Streams Norouzi-Fard et al. (2018) studied streaming submodular maximization under the assumption that elements of a stream arrive in random order. They introduced a streaming algorithm called SALSA with an approximation guarantee better than $1/2$. This algorithm uses $O(k \log(k))$ memory. In a very straightforward way, similarly to the idea of Sieve-Streaming for lower bounding the optimum value, we are able to improve the memory complexity of this algorithm to $O(k)$. Furthermore, Norouzi-Fard et al. (2018) introduced a p-pass algorithm ($p \geq 2$) for submodular maximization subject to a cardinality constraint k. We can also reduce the memory of this p-pass algorithm by a factor $\log(k)$.

Deletion-Robust Mirzasoleiman et al. (2017) have introduced a streaming algorithm for the deletion-robust submodular maximization. Their algorithm provides a summary S of size $O(kd\log(k)/\epsilon)$ where it is robust to deletion of any set D of at most d items. Kazemi et al. (2018) were able to reduce the size of the deletion-robust summary to $O(k\log(k)/\epsilon + d\log^2(k)/\epsilon^3)$. The idea of Sieve-Streaming++ for estimating the value of OPT reduces the memory complexities of these two algorithms to $O(kd/\epsilon)$ and $O(k/\epsilon + d\log^2(k)/\epsilon^3)$, respectively. It is also possible to reduce the memory complexity of STAR-T-GREEDY (Mitrovic et al., 2017b) by at least a factor of $\log(k)$.

Two-Stage Mitrovic et al. (2018b) introduced a streaming algorithm called Replacement-Streaming for the two-stage submodular maximization problem which is originally proposed by (Balkanski et al., 2016; Stan et al., 2017). The memory complexity of Replacement-Streaming is $O(\frac{\ell\log\ell}{\epsilon})$, where ℓ is the size of the produced summary. Again, by applying the idea of Sieve-Streaming++ for guessing the value of OPT and analysis similar to the proof of Theorem 4.7, we can reduce the memory complexity of the streaming two-stage submodular maximization to $O(\frac{\ell}{\epsilon})$.

Streaming Weak Submodularity Weak submodular functions generalize the diminishing returns property.

Definition D.1 (Weakly Submodular (Das and Kempe, 2011))**.** *A monotone and non-negative set function $f : 2^V \to \mathbb{R}_{\geq 0}$ is γ–weakly submodular if for each sets $A, B \subset V$, we have*

$$\gamma \leq \frac{\sum_{e\in A\setminus B} f(\{e\} \mid B)}{f(A \mid B)},$$

where the ratio is considered to be equal to 1 *when its numerator and denominator are* 0*.*

It is straightforward to show that f is submodular if and only if $\gamma = 1$. In the streaming context subject to a cardinality constraint k, Elenberg et al. (2017) designed an algorithm with a constant factor approximation for γ–weakly submodular functions. The memory complexity of their algorithm is $O(\frac{k\log k}{\epsilon})$. By adopting the idea of Sieve-Streaming++, we could reduce the memory complexity of their algorithm to $O(\frac{k}{\epsilon})$.

Table 5 provides a summary of algorithms that we could significantly improve their memory complexity, while their approximation factors are maintained.

Problem	Algorithm	Memory	Improved Memory
Random stream	SALSA	$O(k\log(k))$	$O(k)$
Random stream	P-PASS	$O(k\log(k)/\epsilon)$	$O(k/\epsilon)$
Weak submodular	STREAK	$O(k\log(k)/\epsilon)$	$O(k/\epsilon)$
Deletion-Robust	ROBUST	$O(kd\log(k)/\epsilon)$	$O(kd/\epsilon)$
Deletion-Robust	ROBUST-STREAMING	$O(k\log(k)/\epsilon + d\log^2(k)/\epsilon^3)$	$O(k/\epsilon + d\log^2(k)/\epsilon^3)$
Two-Stage	REPLACEMENT-STREAMING	$O(\ell\log(\ell)/\epsilon)$	$O(\ell/\epsilon)$

Table 5: Streaming algorithms for several other submodular maximization problems. SALSA and P-PASS are due to Norouzi-Fard et al. (2018), STREAK is due to Elenberg et al. (2017), ROBUST is due to Mirzasoleiman et al. (2017), ROBUST-STREAMING is due to Kazemi et al. (2018), and REPLACEMENT-STREAMING is due to Mitrovic et al. (2018b).

D.2 Proof of Theorem 4.7

Proof. **Approximation guarantee** The approximation ratio is proven very similar to the approximation guarantee of SIEVE-STREAMING Badanidiyuru et al. (2014).

Let's define $S^* = \arg\max_{A\subseteq V, |A|\leq k} f(A)$, $\text{OPT} = f(S^*)$ and $\tau^* = \frac{\text{OPT}}{2k}$. We further define $\Delta = \max_{e\in V} f(\{e\})$. It is easy to observe that $\max\{\Delta, \texttt{LB}\} \leq \text{OPT} \leq k\Delta$ and there is a threshold τ such that $(1-\epsilon)\tau^* \leq \tau < \tau^*$. Now consider the set S_τ. SIEVE-STREAMING++ adds elements with a marginal gain at least τ to the set S_τ. We have two cases:

- $|S_\tau| = k$: We define $S_\tau = \{e_1, \cdots, e_k\}$ where e_i is the i-th picked element. Furthermore, we define $S_{\tau,i} = \{e_1, \cdots, e_i\}$. We have

$$f(S_\tau) = \sum_{i=1}^{k} f(\{e_i\} \mid S_{\tau,i-1}) \geq k\tau \geq (\frac{1}{2} - \epsilon)\cdot \text{OPT}.$$

  This is true because the marginal gain of each element at the time it has been added to the set S_τ is at least τ^*.

- $|S_\tau| < k$: We have

$$\text{OPT} \leq f(S^* \cup S_\tau) = f(S_\tau) + f(S^* \mid S_\tau) \overset{(a)}{\leq} f(S_\tau) + \sum_{e\in S^*\setminus S_\tau} f(\{e\} \mid S_\tau) \overset{(b)}{\leq} f(S_\tau) + k\tau^*$$
$$= f(S_\tau) + \frac{\text{OPT}}{2},$$

  where (a) is correct because f is a submodular function, and we have (b) because each element of S^* that is not picked by the algorithm has had a marginal gain of less than $\tau < \tau^*$.

Memory complexity Let S_τ be the set we maintain for threshold τ. We know that OPT is at

least $\texttt{LB} = \max_\tau f(S_\tau) \geq \max_\tau(|S_\tau| \times \tau)$ because the marginal gain of an element in set S_τ is at least τ. Note that LB is the best solution found so far. Given this lower bound on OPT (which is potentially better than Δ if we have picked enough items), we can dismiss all thresholds that are too small, i.e., remove all thresholds $\tau < \frac{\texttt{LB}}{2k} \leq \frac{\text{OPT}}{2k}$. For any remaining $\tau \geq \frac{\texttt{LB}}{2k}$, we know that $|S_\tau|$ is at most $\frac{\texttt{LB}}{\tau}$. We consider two sets of thresholds: (i) $\frac{\texttt{LB}}{2k} \leq \tau \leq \frac{\texttt{LB}}{k}$, and (ii) $\tau \geq \frac{\texttt{LB}}{k}$. For the first group of thresholds, the bound on $|S_\tau|$ is the trivial upper bound of k. Note that we have $\log_{1+\epsilon}(2) \leq \lceil \log(2)/\epsilon \rceil$ of such thresholds. For the second group of thresholds, as we increase τ, for a fixed value of LB the upper bound on the size of S_τ gets smaller. Indeed, these upper bounds are geometrically decreasing values with the first term equal to k. And they reduce by a coefficient of $(1+\epsilon)$ as thresholds increase by a factor of $(1+\epsilon)$. Therefore, we can bound the memory complexity by

$$\text{Memory complexity} \leq \left\lceil \frac{k \log(2)}{\epsilon} \right\rceil + \sum_{i=0}^{\log_{1+\epsilon}(k)} \frac{k}{(1+\epsilon)^i} = O\left(\frac{k}{\epsilon}\right).$$

Therefore, the total memory complexity is $O(\frac{k}{\epsilon})$.

Query complexity For every incoming element e, in the worst case, we should compute the marginal gain of e to all the existing sets S_τ. Because there is $O(\frac{\log k}{\epsilon})$ of such sets (the number of different thresholds), therefore the query complexity per element is $O(\frac{\log k}{\epsilon})$. □

D.3 Proof of Theorem 4.8

Proof. **Approximation Guarantee:** Assume $\mathcal{B}$ is the set of elements buffered from the stream V. Let's define $S^* = \arg\max_{A \subseteq V, |A| \leq k} f(A)$, $\text{OPT} = f(S^*)$ and $\tau^* = \frac{\text{OPT}}{2k}$. Similar to the proof of Theorem 4.7, we can show that Batch-Sieve-Streaming++ considers a range of thresholds such that for one of them (say τ) we have $\frac{\text{OPT}(1-\epsilon)}{2k} \leq \tau < \frac{\text{OPT}}{2k}$. In the rest of this proof, we focus on τ and its corresponding set of picked items S_τ. For set S_τ we have two cases:

- if $|S_\tau| < k$, we have:

$$\begin{aligned}\text{OPT} \leq f(S^* \cup S_\tau) = f(S_\tau) + f(S^* \mid S_\tau) &\overset{(a)}{\leq} f(S_\tau) + \sum_{e \in S^* \setminus S_\tau} f(\{e\} \mid S_\tau) \overset{(b)}{\leq} f(S_\tau) + k\tau^* \\ &= f(S_\tau) + \frac{\text{OPT}}{2},\end{aligned}$$

  where inequality (a) is correct because f is a submodular function. And inequality (b) is

correct because each element of the optimal set S^* that is not picked by the algorithm, i.e., it had discarded in the filtering process, has had a marginal gain of less than $\tau < \tau^*$.

- if $|S_\tau| = k$: Assume the set S_τ of size k is sampled in ℓ iterations of the while loop in Lines 2–18 of Algorithm 11, and T_i is the union of sampled batches in the i-th iteration of the while loop. Furthermore, let $T_{i,j}$ denote the j-th sampled batch in the i-th iteration of the while loop. We define $S_{\tau,i,j} = \bigcup_{i,j} T_{i,j}$, i.e, $S_{\tau,i,j}$ is the state of set S_τ after the j-th batch insertion in the i-th iteration of the while loop. We first prove that the average gain of each one of these sets T_i to the set S_τ is at least $(1-2\epsilon)\cdot|T_i|$.

  To lower bound the average marginal gain of T_i, for each $T_{i,j}$ we consider three different cases:

  - the while loop breaks at Line 7 of Algorithm 11: We know that the size of all $T_{i,j}$ is one in this case. It is obvious that $f(T_{i,j} \mid S_{\tau,i,j-1}) \geq (1-\epsilon)\cdot\tau$.
  - Threshold-Sampling passes the first loop and does not break at Line 16, i.e., it continues to pick items till $S_\tau = k$ or the buffer memory is empty: again it is obvious that

    $$f(T_{i,j} \mid S_{\tau,i,j-1}) \geq (1-\epsilon)\cdot\tau\cdot|T_{i,j}|$$

    This is true because when set $T_{i,j}$ is picked, it has passed the test at Line 15. Note that it is possible the algorithm breaks at Line 14 without passing the test at Line 15. If the average marginal gain of the sampled set $T_{i,j}$ is more than $(1-\epsilon)\cdot\tau$ then the analysis would be exactly the same as the current case. Otherwise, we handle it similar to the next case where the sampled batch does not provide the required average marginal gain.
  - it passes the first loop and breaks at Line 16. We have the two following observations:

    1. in the current while loop, from the above-mentioned cases, we conclude that the average marginal gain of all the element picked before the last sampling is at least $(1-\epsilon)\cdot\tau$, i.e.,

       $$\forall r, 1 \leq r < j : f(T_{i,r} \mid S_{\tau,i,r-1}) \geq (1-\epsilon)\cdot\tau\cdot|T_{i,r}|.$$

    2. the number of elements which are picked at the latest iteration of the while loop is at most ϵ fraction of all the elements picked so far (in the current while loop), i.e., $|T_{i,j}| \leq \epsilon\cdot|\bigcup_{1\leq r<j} T_{i,r}|$ and $|T_i| \leq (1+\epsilon)\cdot|\bigcup_{1\leq r<j} T_{i,r}|$. Therefore, from the

monotonicity of f, we have

$$f(\bigcup_{1\le r\le j} T_{i,r} \mid S) \ge f(\bigcup_{1\le r<j} T_{i,r} \mid S) \ge (1-\epsilon)\cdot\tau\cdot|\bigcup_{1\le r<j} T_{i,r}| \ge \frac{|T_i|\cdot\tau\cdot(1-\epsilon)}{(1+\epsilon)} \ge (1-2\epsilon)\cdot\tau\cdot|T_i|.$$

To sum-up, we have

$$f(S_\tau) = \sum_{i=1}^{\ell} f(T_i \mid S_{\tau,i-1}) \ge \sum_{i=1}^{\ell}(1-2\epsilon)\cdot\tau\cdot|T_i| = (1-2\epsilon)\cdot\tau\cdot k \ge \left(\frac{1}{2}-\frac{3\epsilon}{2}\right)\cdot \mathrm{OPT}.$$

Memory complexity In a way similar to analyzing the memory complexity of Sieve-Streaming++, we conclude that the required memory of Batch-Sieve-Streaming++ in order to store solutions for different thresholds is also $O(\frac{k}{\epsilon})$. Since we buffer at most $\mathtt{B}$ items, the total memory complexity is $O(\mathtt{B}+\frac{k}{\epsilon})$.

Adaptivity Complexity of Threshold-Sampling To guarantee the adaptive complexity of our algorithm, we first upper bound the expected number of iterations of the while loop in Lines 2–18 of Algorithm 11.

Lemma D.1. *For any constant $\epsilon > 0$, the expected number of iterations in the while loop of Lines 2–18 of* Threshold-Sampling *is $O(\log(|\mathcal{B}|))$ where $\mathcal{B}$ is the set of buffered elements passed to* Threshold-Sampling.

Before, proving Lemma D.1, we discuss how this lemma translates to the total expected adaptivity of Batch-Sieve-Streaming++.

There are at most $\lceil\frac{1}{\epsilon}\rceil + \log_{1+\epsilon} k = O(\frac{\log k}{\epsilon})$ adaptive rounds in each iteration of the while loop of Algorithm 11. So, from Lemma D.1 we conclude that the expected adaptive complexity of each call to Threshold-Sampling is $O(\frac{\log(|\mathcal{B}|)\log(k)}{\epsilon})$. To sum up, the general adaptivity of the algorithm takes its maximum value when the number of times a buffer gets full is the most, i.e., when $|\mathcal{B}| = \mathtt{Threshold}\times\mathtt{B}$ for $\frac{N}{\mathtt{Threshold}\times\mathtt{B}}$ times. We assume $\mathtt{Threshold}$ is constant. Therefore, the expected adaptive complexity is $O(\frac{N\log(\mathtt{B})\log(k)}{\mathtt{B}\epsilon})$. □

Proof of Lemma D.1

Proof. Since we are only adding elements to S, using submodularity the marginal value of any

element x to S, i.e. $f(x \mid S)$, is decreasing. Therefore, once an element is removed from the buffer $\mathcal{B}$, it never comes back. As a result, the set $\mathcal{B}$ is shrinking over time. When $\mathcal{B}$ becomes empty, the algorithm terminates. Therefore it suffices to show that in every iteration of the while loop, a constant fraction of elements will be removed from $\mathcal{B}$ in expectation. The rest of the analysis follows by analyzing the expected size of $\mathcal{B}$ over time and applying Markov's inequality.

We note that to avoid confusion, we call one round of the while loop in Lines 3–18 of Threshold-Sampling an iteration. There are two other internal for loops at Lines 4–10 and Lines 11–18. Later in the proof, we call each run of these for loops a step. There are $\lceil \frac{1}{\epsilon} \rceil$ steps in the first for loop and $O(\log(k))$ steps in the second.

If an iteration ends with growing S into a size k set, that is going to be the final iteration as the algorithm Threshold-Sampling because the algorithm terminates once k elements are selected. So we focus on the other case. An iteration breaks (finishes) either in the first for the loop at Lines 4–10 or in the second for loop of Lines 11–18. We say an iteration fails if after termination less than $\epsilon/2$ fraction of elements in $\mathcal{B}$ is removed. For iteration ℓ, let $\mathcal{B}^\ell$ be the set $\mathcal{B}$ at Line 3 at the beginning of this iteration. So the first set $\mathcal{B}^1$ consists of all the input elements passed to Threshold-Sampling. So we can say that an iteration ℓ fails if $|\mathcal{B}^{\ell+1}|$ is greater than $(1-\epsilon/2)\cdot|\mathcal{B}^\ell|$.

Failure of an iteration can happen in any of the $\lceil \frac{1}{\epsilon} \rceil + O(\log(k))$ steps of the two for loops. For each step $1 \leq z \leq \lceil \frac{1}{\epsilon} \rceil + O(\log(k))$, we denote the probability that the current iteration is terminated at step z at a failed state with P_z. The probability that an iteration will not fail can then be written as

$$\prod_z (1 - P_z).$$

In the rest of the proof, we show that this quantity is at least a constant for any constant $\epsilon > 0$.

First, we show that at any of the $\lceil \frac{1}{\epsilon} \rceil$ steps of the first for loop, the probability of failing is less than $\epsilon/2$. Let us consider step $1 \leq z \leq \lceil \frac{1}{\epsilon} \rceil$. We focus on the beginning of step z and upper bound P_z for any possible outcome of the previous steps $1, \cdots, z-1$. Let S be the set of selected elements in all the first $z-1$ steps. If at least $\epsilon/2$ fraction of elements in $\mathcal{B}^\ell$ has a marginal value less than τ to S, we can say that the iteration will not fail for the rest of the steps for sure (with probability 1). We note that as S grows the marginal values of elements to it will not increase, so at least $\epsilon/2$ fraction of elements will be filtered out independent of which step the process terminates.

So we focus on the case that less than $\epsilon/2$ fraction of elements in $\mathcal{B}^\ell$ have marginal value less than τ to S. Since, in the first loop, we pick one of them randomly and look at its marginal value as a

test to whether terminate the iteration or not, the probability of termination at this step z is not more than $\epsilon/2$ and therefore P_z is also at most $\epsilon/2$.

In the second for loop, at Lines 11–18, we have a logarithmic number of steps and we can upper bound the probability of terminating the iteration in a failed state at any of these steps in a similar way. The main difference is that instead of sampling one random element from $\mathcal{B}$, we pick t random elements and look at their average marginal value together as a test to whether terminate the current iteration or not.

We want to upper bound the probability of terminating the iteration in a step $z > \lceil \frac{1}{\epsilon} \rceil$ at a failed state. This will happen if at the step z the Threshold-Sampling algorithm picks a random subset T with

- $\frac{f(T \mid S)}{|T|} \leq (1-\epsilon)\tau$, and
- also less than $\epsilon/2$ fraction of elements in $\mathcal{B}^\ell$ has a marginal value less than τ to $T \cup S$.

We look at the process of sampling T as a sequential process in which we pick t random elements one by one. We can call each of these t parts a small random experiment. We note that the first property above holds only if in at least ϵt of these smaller random experiments the marginal value of the selected element to the current set S is below τ. We also assume that we add the selected elements to S as we move on. We simulate this random process with a binomial process of tossing t independent coins. If the marginal value of the i-th sampled element to S is at least τ, we say that the associated coin toss is a head. Otherwise, we call it a tail. The probability of a tail depends on the fraction of elements in $\mathcal{B}^\ell$ with marginal value less than τ to S. If this fraction at any point is at least $\epsilon/2$, we know that the second necessary property for a failed iteration does not hold anymore and will not hold for the rest of the steps. Therefore the failure happens only if we face at least ϵt tails each with probability at most $\epsilon/2$. The rest of the analysis is applying simple concentration bounds for different values of t.

So we have a binomial distribution with t trials each with head probability at least $1 - \epsilon/2$, and we want to upper bound the probability that we get at least ϵt tails. The expected number of tails is not more than $\epsilon t/2$ so using Markov's inequality, the probability of seeing at least ϵt tails is at most 0.5. Furthermore, for larger values of t we can have much better concentration bounds.

Using Chernoff type bounds in Lemma D.2, we know the probability of observing at least ϵt tails is not more than:

$$P_z \leq \Pr(\text{tails} - \epsilon t/2 \geq \epsilon t/2) \leq e^{-\epsilon t/10}.$$

As we proceed in steps, the number of samples t grows geometrically. Consequently, the failure probability declines exponentially (double exponential in the limit).

So the number of steps it takes to reach the failure probability declining phase is a function of ϵ and therefore it is a constant number. We conclude that for any constant $\epsilon > 0$, the probability of not failing in an iteration, i.e., $\prod_z (1 - P_z)$, is lower bounded by a constant $\zeta_\epsilon > 0$. Since any iteration will terminate eventually, we can say that for any iteration with constant probability an $\epsilon/2$ fraction of elements will be filtered out of $\mathcal{B}$. So the expected size of $\mathcal{B}$ after X iterations will be at most $2^{-\Omega(X)}n$ where n is the number of input elements at the beginning of THRESHOLD-SAMPLING. So the probability of having more than $C\log(|\mathcal{B}|)$ iterations decreases exponentially with C for any coefficient C using Markov's inequality which means the expected number of iterations is $O(\log(|\mathcal{B}|))$. □

Lemma D.2 (Chernoff bounds, Bansal and Sviridenko (2006))**.** *Suppose $X_1, \ldots, X_n$ are binary random variables such that $\Pr(X_i = 1) = p_i$. Let $\mu = \sum_{i=1}^n p_i$ and $X = \sum_{i=1}^n X_i$. Then for any $a > 0$, we have*

$$\Pr(X - \mu \geq a) \leq e^{-a \min\left(\frac{1}{5}, \frac{a}{4\mu}\right)}.$$

Moreover, for any $a > 0$, we have

$$\Pr(X - \mu \leq -a) \leq e^{-\frac{a^2}{2\mu}}.$$

D.4 Proof of Theorem 4.9

For m different data streams, assume $\mathcal{B}_i$ is the set of elements buffered from the i-th stream. We define $\mathcal{B}$ to be the union of all elements from all streams. The communication cost of BATCH-SIEVE-STREAMING++ in the multi-source setting is the total number of elements sampled (in a distributed way) from all sets $\{\mathcal{B}_i\}_{1 \leq i \leq m}$ in Lines 5 and 13 of Algorithm 11.

As a result, we can conclude that the communication cost is at most twice the number of elements has been in a set S_τ at a time during the run of the algorithm. To see the reason for this argument, note that because the filtering step happens just before the for loop of Lines 4–10, the first picked sample in this for loop always passes the test and is added to S_τ. Furthermore, all the items sampled

at Line 13, irrespective of their marginal gain, are added to S_τ. So, in the worst case scenario, the communication complexity is maximum when the for loop breaks always at the second instance of the sampling process of Line 5 (after one successful try). Therefore, we only need to upper bound the total number of elements which at some point has been in a set S_τ at one of the calls to Threshold-Sampling.

The first group of thresholds the Batch-Sieve-Streaming++ algorithm considers the interval $[\Delta_0/(2k), \Delta_0]$, where in the beginning we have $\texttt{LB} = \Delta_0$. Following the same arguments as the proof of Theorem 4.7, we can show that if neither LB nor Δ changes, the total number of elements in sets $\{S_\tau\}$ is $O(\frac{k}{\epsilon})$. We define $\Delta_{\max}$ to be the largest singleton element in the whole data streams. The number of times the interval of thresholds changes because of the change in Δ is $\log_{1+\epsilon}(\Delta_{\max}/\Delta_0)$. Furthermore, by changes in LB some thresholds and their corresponding sets are deleted and new elements might be added. The number of times LB changes is upper bounded by $\log_{1+\epsilon}(\mathrm{OPT}/\Delta_0)$. Note that we have $\Delta_{\max} \leq \mathrm{OPT}$. From the fact that the number of changes in the set of thresholds is upper bounded by $\log_{1+\epsilon}(\Delta_{\max}/\Delta_0) = O(\frac{\log}{\epsilon})$ and the number of elements in $\{S_\tau\}$ at every step of the algorithm is $O(\frac{k}{e})$, we conclude the total communication cost of Batch-Sieve-Streaming++ is $O(\frac{k \log}{\epsilon^2})$.

D.5 Twitter Dataset Details

To clean the data, we removed punctuation and common English words (known as stop words, thus leaving each individual tweet as a list of keywords with a particular timestamp. To give additional value to more popular posts, we also saved the number of retweets each post received.

Therefore, any individual tweet t consists of a set of keywords K_t and a value v_t that is the number of retweets divided by the number of words in the post.

A set of tweets T can be thought of as a list of $(keyword, score)$ pairs. The keywords K_T in a set T is simply a union of the keywords of the tweets in T:

$$K_T = \bigcup_{t \in T} K_t$$

The score s_k of each keyword $k \in K_T$ is simply the sum of the values of posts containing that

keyword. That is, if $T_k \subseteq T$ is the subset of tweets in T containing the keyword k, then:

$$s_k = \sum_{t \in T_k} v_t$$

Therefore, we define our submodular function f as follows:

$$f(T) = \sum_{k \in K_T} \sqrt{s_k}$$

Intuitively, we sum over all the keyword scores because we want our set of tweets to cover as many high-value keywords as possible. However, we also use the square root to introduce a notion of diminishing returns because once a keyword already has a high score, we would prefer to diversify instead of further picking similar tweets.

Function Definition Consider a function f defined over a ground set V of items. Each item $e \in V$ consists of a positive value val_e and a set of ℓ_e keywords $W_e = \{w_{e,1}, \cdots, w_{e,\ell_e}\}$ from a general set of keywords $\mathcal{W}$. The score of a word $w \in W_e$ for an item e is defined by $\text{score}(w, e) = \text{val}_e$. If $w \notin W_e$, we define $\text{score}(w, e) = 0$. For a set $S \subseteq V$ the function f is defined as follows:

$$f(S) = \sum_{w \in \mathcal{W}} \sqrt{\sum_{e \in S} \text{score}(w, e)}. \tag{22}$$

Lemma D.3. *The function f defined in Eq. (22) is non-negative and monotone submodular.*

Proof. The not-negativity and monotonicity of f are trivial. For two sets $A \subset B$ and $e \in V \setminus B$ we show that

$$f(\{e\} \cup A) - f(A) \geq f(\{e\} \cup B) - f(B).$$

To prove the above inequality, assume $W_e = \{w_{e,1}, \cdots, w_{e,\ell_e}\}$ is the set of keywords of e. For a keyword $w_{e,i}$ define $a_{w_{e,i}} = \sum_{u \in A} \text{score}(w_{e,i}, u)$ and $b_{w_{e,i}} = \sum_{u \in B} \text{score}(w_{e,i}, u)$. It is obvious that $a_{w_{e,i}} \leq b_{w_{e,i}}$. It is straightforward to show that

$$\sqrt{a_{w_{e,i}} + \text{score}(w_{e,i}, e)} - \sqrt{a_{w_{e,i}}} \geq \sqrt{b_{w_{e,i}} + \text{score}(w_{e,i}, u)} - \sqrt{b_{w_{e,i}}}.$$

If sum over all keywords in W_e then the submodularity of f is proven. □

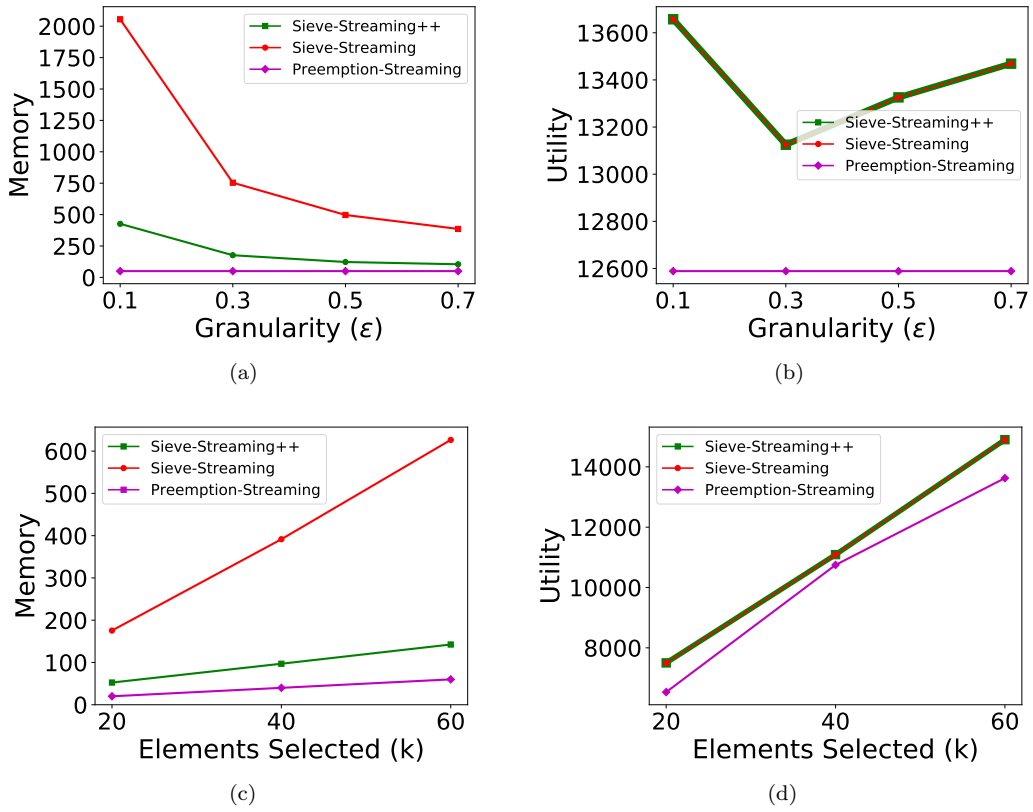


Figure 22: Single-source streaming results on the Twitter summarizaton task. In (a) and (b), $\epsilon = 0.5$. In (c) and (d), $k = 50$.

D.6 More Experimental Results

D.6.1 Single-Source Experiments

Here we present the set of graphs that we displayed in Figures 13a through 13d, except here they are run on the Twitter dataset instead. For the most part, they are showing the same trends we saw before. Sieve-Streaming++ has the exact same utility as Sieve-Streaming, which is better than Preemption-Streaming. We also see the memory requirement of Sieve-Streaming++ is much lower than that of Sieve-Streaming, as we had hoped.

The only real difference is in the shape of the utility curve as ϵ varies. In Figure 13b, the utility was decreasing as ϵ increased, which is not necessarily the case here. However, this is relatively standard because changing ϵ completely changes the set of thresholds kept by Sieve-Streaming++, so although it usually helps the utility, it is not necessarily guaranteed to do so.

Also, note that we only went up to $k = 60$ in this experiment because Preemption-Streaming

was prohibitively slow for larger k.

D.6.2 Multi-Source Experiments

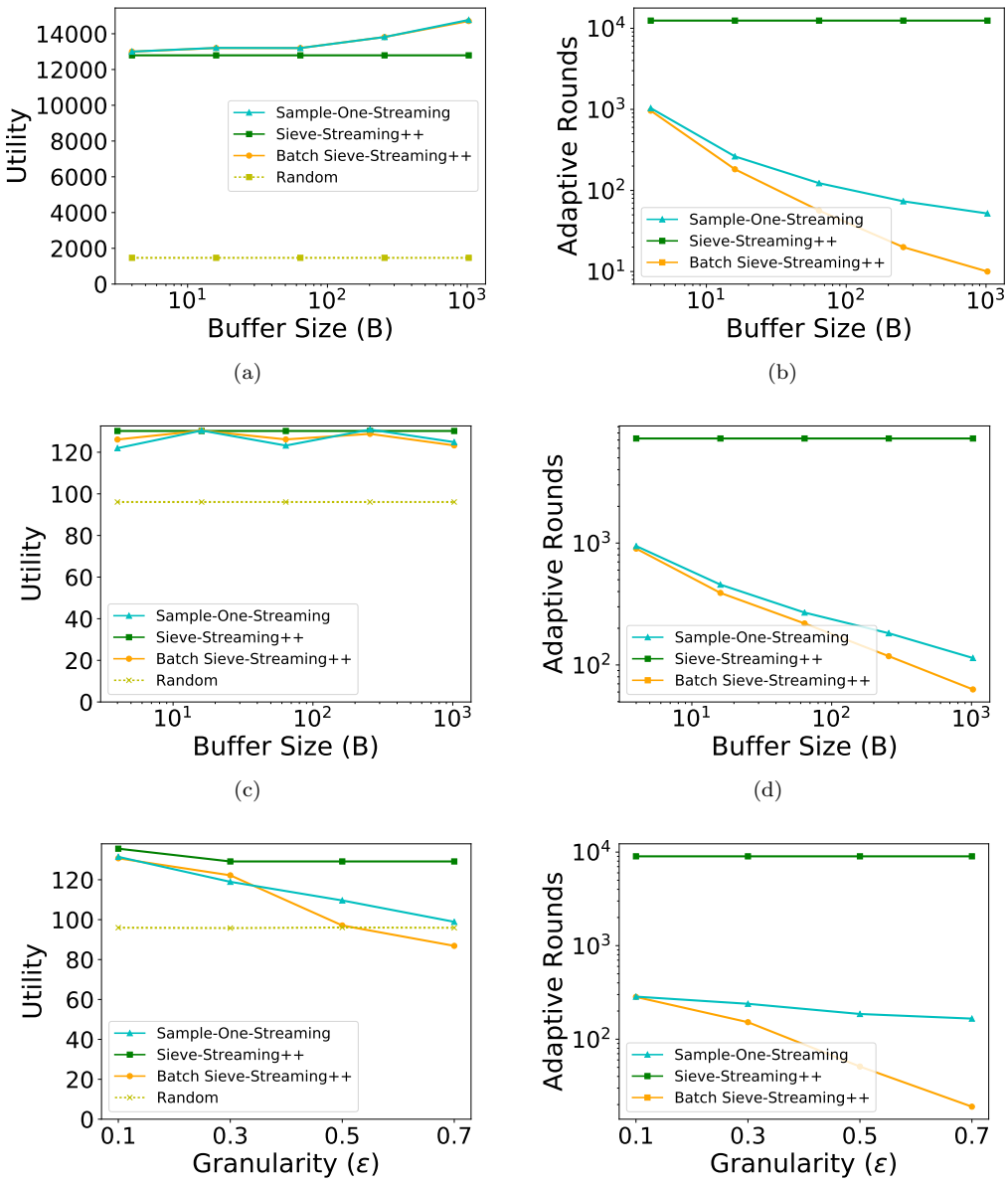


Figure 23: Additional multi-source graphs. (a) and (b) are additional graphs for the Twitter dataset, this time with $\epsilon = 0.6$ and $k = 50$. (c) through (f) are the equivalent of Figures 13e through 13h, but for the YouTube dataset. Unless they are being varied on the x-axis, we set $\epsilon = 0.25$, $\texttt{B} = 100$, and $k = 100$.

In Figure 13e, Sieve-Streaming++ had the best utility performance. In Figure 23a, we set $k = 50$ and $\epsilon = 0.6$ and now we see that Batch-Sieve-Streaming++ and Sample-One-Streaming have

higher utility, and that this utility increases as the buffer size increases. However, in this case too, the main message is that the utilities of the three algorithms are comparable, but Batch-Sieve-Streaming++ uses the fewest adaptive rounds (Figure 13f).

In Figures 23c through 23f, we display the same set of graphs as Figures 13e through 13h, but for the YouTube experiment. In the YouTube experiment, it is more difficult to select a set of items that is significantly better than random, so we need to use a smaller value of ϵ. We see that for smaller ϵ, the difference in adaptive rounds between Batch-Sieve-Streaming++ and Sample-One-Streaming is smaller. This is consistent with our results because the number of adaptive rounds required by Sample-One-Streaming does not change much with ϵ, while the number of adaptive rounds of Batch-Sieve-Streaming++ increases as ϵ gets smaller.

Appendix E. Submodularity on Hypergraphs Appendix

E.1 Proof of Theorem 5.1

In this section we prove Theorem 5.1, however, before we get into the proof, let us first recall the theorem itself.

Theorem 5.1. *The approximation ratio of Algorithm 12 is at least* $\frac{1-e^{-(1-\frac{1}{k})}}{2d_{in}+1}$.

We begin the proof of the theorem by defining some additional notation. First, let ℓ be the number of iterations completed by the main loop of Algorithm 12, *i.e.*, the number of iterations in which σ is updated. Then, for every $0 \leq s \leq \ell$, let σ_s be the value of σ after s iterations of this loop have been performed. In other words, σ_0 is the initial value of σ when we first get to the loop, σ_1 is the value of σ at the end of the first iteration of the loop, and so on. Note that σ_ℓ is the output of Algorithm 12. Additionally, we also denote by e_s and $\mathcal{E}_s$, for every $1 \leq s \leq \ell$, the values assigned to the variables e_{ij} and $\mathcal{E}$, respectively, at iteration number s of the above loop. Finally, we also define ℓ' as the real number of iterations performed by the above loop. Notice that $\ell' = \ell$ unless the algorithm exits the loop because $\mathcal{E} = \varnothing$, in which case $\ell' = \ell + 1$ and we define $\mathcal{E}_{\ell'} = \varnothing$.

Observation E.1. *For every* $1 \leq s \leq \ell$, $f(\sigma_s) - f(\sigma_{s-1}) \geq h(e_s \mid E(\sigma_{s-1}))$.

Proof. Notice that the way σ is updated in each iteration of Algorithm 12 guarantees that $e_s \in E(\sigma_s) \setminus E(\sigma_{s-1})$. Moreover, since σ_{s-1} is a prefix of σ_s, we also get $E(\sigma_{s-1}) \subseteq E(\sigma_s)$. Thus,

$$f(\sigma_s) - f(\sigma_{s-1}) = h(E(\sigma_s)) - h(E(\sigma_{s-1})) \geq h(E(\sigma_{s-1}) + e_s) - h(E(\sigma_{s-1})) = h(e_s \mid E(\sigma_{s-1})) \ ,$$

where the inequality follows from the monotonicity of h. □

Let σ^* denote an arbitrary (but fixed) optimal sequence. We now need to prove a few properties of σ^*.

Observation E.2. $|E(\sigma^*)| \leq d_{\text{in}}k$.

Proof. Observe that σ^* contains at most k vertices because it is feasible. This means that there can be at most $d_{\text{in}}k$ arcs that end in a vertex of σ^*, which implies the observation since every arc of $E(\sigma^*)$ must end at a vertex of σ^*. □

The next lemma studies the change in the value of $(E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})$ as a function of s.

Lemma E.3. *For every* $1 \leq s < \ell'$, $h((E(\sigma^*) \cap \mathcal{E}_{s+1}) \cup E(\sigma_s)) \geq h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) - 2d_{\text{in}} \cdot h(e_s \mid \sigma_{s-1})$.

Proof. Recall that, for every $1 \leq s \leq \ell'$, $\mathcal{E}_s$ contains the arcs of E whose end point is not in σ_{s-1}. This definition implies that $\mathcal{E}_{s+1} \subseteq \mathcal{E}_s$ because σ_{s-1} is a prefix of σ_s. In contrast, since σ_s contains at most two vertices that do not appear in σ_{s-1} and each one of these vertices can be the end point of at most d_{in} arcs, we also get $|\mathcal{E}_s \setminus \mathcal{E}_{s+1}| \leq 2d_{\text{in}}$.

Using these observations and the submodularity of h, we can now derive the following inequality.

$$\begin{aligned} h(E(\sigma^*) \cap \mathcal{E}_s \mid E(\sigma_{s-1})) - h(E(\sigma^*) \cap \mathcal{E}_{s+1} \mid E(\sigma_{s-1})) &\leq \sum_{e \in E(\sigma^*) \cap (\mathcal{E}_s \setminus \mathcal{E}_{s+1})} h(e \mid E(\sigma_{s-1})) \\ &\leq \sum_{e \in E(\sigma^*) \cap (\mathcal{E}_s \setminus \mathcal{E}_{s+1})} h(e_s \mid E(\sigma_{s-1})) = |E(\sigma^*) \cap (\mathcal{E}_s \setminus \mathcal{E}_{s+1})| \cdot h(e_s \mid E(\sigma_{s-1})) \\ &\leq |\mathcal{E}_s \setminus \mathcal{E}_{s+1}| \cdot h(e_s \mid E(\sigma_{s-1})) \leq 2d_{\text{in}} \cdot h(e_s \mid E(\sigma_{s-1})) \ , \end{aligned}$$

where the second inequality follows from the definition of e_s which guarantees that it maximizes $h(e_s \mid E(\sigma_{s-1}))$ among all the edges of $\mathcal{E}_s$.

It now remains to observe that

$$\begin{aligned} &h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) - h((E(\sigma^*) \cap \mathcal{E}_{s+1}) \cup E(\sigma_s)) \\ &\qquad \leq h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) - h((E(\sigma^*) \cap \mathcal{E}_{s+1}) \cup E(\sigma_{s-1})) \\ &\qquad = h(E(\sigma^*) \cap \mathcal{E}_s \mid E(\sigma_{s-1})) - h(E(\sigma^*) \cap \mathcal{E}_{s+1} \mid E(\sigma_{s-1})) \leq 2d_{\text{in}} \cdot h(e_s \mid E(\sigma_{s-1})) \ , \end{aligned}$$

where the first inequality follows from the monotonicity of h since the fact that σ_{s-1} is a prefix of σ_s implies $E(\sigma_{s-1}) \subseteq E(\sigma_s)$. □

We are now ready to combine all the above claims into a single lemma.

Lemma E.4. *For every $1 \leq s \leq \ell$, the following two inequalities hold:*

- $h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) \geq f(\sigma^*) - 2d_{\text{in}} \cdot f(\sigma_{s-1})$.
- $f(\sigma_s) - f(\sigma_{s-1}) \geq \frac{f(\sigma^*) - 2d_{\text{in}} \cdot [f(\sigma_{s-1}) - f(\sigma_0)] - f(\sigma_{s-1})}{d_{\text{in}} k} \geq \frac{f(\sigma^*) - (2d_{\text{in}}+1) \cdot f(\sigma_{s-1})}{d_{\text{in}} k}$.

Moreover, the first inequality holds also for $s = \ell'$.

Proof. Lemma F.8 shows that, for every $1 \leq t < \ell'$, we have

$$h((E(\sigma^*) \cap \mathcal{E}_{t+1}) \cup E(\sigma_t)) \geq h((E(\sigma^*) \cap \mathcal{E}_t) \cup E(\sigma_{t-1})) - 2d_{\text{in}} \cdot h(e_t \mid \sigma_{t-1}) \ .$$

Adding up this inequality for $1 \leq t < s$ gives us

$$\begin{aligned} h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) &\geq h((E(\sigma^*) \cap \mathcal{E}_1) \cup E(\sigma_0)) - 2d_{\text{in}} \cdot \sum_{t=1}^{s-1} h(e_t \mid \sigma_{t-1}) \\ &= f(\sigma^*) - 2d_{\text{in}} \cdot \sum_{t=1}^{s-1} h(e_t \mid \sigma_{t-1}) \geq f(\sigma^*) - 2d_{\text{in}} \cdot \sum_{t=1}^{s-1} [f(\sigma_t) - f(\sigma_{t-1})] \\ &= f(\sigma^*) - 2d_{\text{in}} \cdot [f(\sigma_{s-1}) - f(\sigma_0)] \geq f(\sigma^*) - 2d_{\text{in}} \cdot f(\sigma_{s-1}) \ . \end{aligned}$$

The first equality follows since the fact that σ_0 is empty implies $E(\sigma_0) = \varnothing$ and $E(\sigma^*) \cap \mathcal{E}_1 = E(\sigma^*)$. Additionally, the second inequality follows from Observation E.1, and the last inequality follows from the non-negativity of f. This proves that the first inequality of the lemma holds for every $1 \leq s \leq \ell'$. In the rest of the proof we aim to prove the second inequality, and thus, assume $1 \leq s \leq \ell$.

Recall now that e_s is the edge of $\mathcal{E}_s$ maximizing $h(e_s \mid E(\sigma_{s-1}))$ and that the size of $E(\sigma^*) \cap \mathcal{E}_s \subseteq E(\sigma^*)$ is at most $d_{\text{in}} k$ by Observation E.2. Thus, by the submodularity of h,

$$\begin{aligned} h(e_s \mid E(\sigma_{s-1})) &\geq \frac{\sum_{e \in E(\sigma^*) \cap \mathcal{E}_s} h(e \mid E(\sigma_{s-1}))}{|E(\sigma^*) \cap \mathcal{E}_s|} \geq \frac{h(E(\sigma^*) \cap \mathcal{E}_s \mid E(\sigma_{s-1}))}{d_{\text{in}} k} \\ &= \frac{h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) - h(E(\sigma_{s-1}))}{d_{\text{in}} k} \\ &\geq \frac{\{f(\sigma^*) - 2d_{\text{in}} \cdot [f(\sigma_{s-1}) - f(\sigma_0)]\} - f(\sigma_{s-1})}{d_{\text{in}} k} \geq \frac{f(\sigma^*) - (2d_{\text{in}} + 1) \cdot f(\sigma_{s-1})}{d_{\text{in}} k} \ . \end{aligned}$$

The lemma follows from this inequality since $h(e_s \mid E(\sigma_{s-1}))$ lower bounds $f(\sigma_s) - f(\sigma_{s-1})$ by

Observation E.1. □

Corollary E.5. *If* $2d_{\text{in}}+1 < d_{\text{in}}k$*, then* $f(\sigma_\ell) \geq \frac{f(\sigma^*)}{2d_{\text{in}}+1} + \frac{[1-(2d_{\text{in}}+1)/(d_{\text{in}}k)]^\ell}{2d_{\text{in}}+1} \cdot [(2d_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)]$.

Proof. To prove the corollary, we prove by induction the stronger claim that, for every $0 \leq s \leq \ell$,

$$f(\sigma_s) \geq \frac{f(\sigma^*)}{2d_{\text{in}}+1} + \frac{[1-(2d_{\text{in}}+1)/(d_{\text{in}}k)]^s}{2d_{\text{in}}+1} \cdot [(2d_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)] \ .$$

For $s = 0$ this inequality is true since

$$\begin{aligned} f(\sigma_0) &= \frac{f(\sigma^*)}{2d_{\text{in}}+1} + \frac{1}{2d_{\text{in}}+1} \cdot [(2d_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)] \\ &= \frac{f(\sigma^*)}{2d_{\text{in}}+1} + \frac{[1-(2d_{\text{in}}+1)/(d_{\text{in}}k)]^0}{2d_{\text{in}}+1} \cdot [(2d_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)] \ . \end{aligned}$$

Assume now that the claim holds for $s-1 \geq 0$, and let us prove it for s. By Lemma E.4,

$$f(\sigma_s) \geq f(\sigma_{s-1}) + \frac{f(\sigma^*) - (2d_{\text{in}}+1) \cdot f(\sigma_{s-1})}{d_{\text{in}}k} = \left(1 - \frac{2d_{\text{in}}+1}{d_{\text{in}}k}\right) \cdot f(\sigma_{s-1}) + \frac{f(\sigma^*)}{d_{\text{in}}k} \ .$$

Plugging in the induction hypothesis, we get

$$\begin{aligned} f(\sigma_s) \geq \ & \left(1 - \frac{2d_{\text{in}}+1}{d_{\text{in}}k}\right) \cdot \left\{ \frac{f(\sigma^*)}{2d_{\text{in}}+1} + \frac{[1-(2d_{\text{in}}+1)/(d_{\text{in}}k)]^{s-1}}{2d_{\text{in}}+1} \cdot [(2d_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)] \right\} \\ & + \frac{f(\sigma^*)}{d_{\text{in}}k} = \frac{f(\sigma^*)}{2d_{\text{in}}+1} + \frac{[1-(2d_{\text{in}}+1)/(d_{\text{in}}k)]^s}{2d_{\text{in}}+1} \cdot [(2d_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)] \ . \end{aligned}$$

□

We are now ready to prove Theorem 5.1.

Proof of Theorem 5.1. First, we need to consider the case that Algorithm 12 terminates because the set $\mathcal{E}$ becomes empty. In this case $\mathcal{E}_{\ell'} = \varnothing$, which implies

$$h((E(\sigma^*) \cap \mathcal{E}_{\ell'}) \cup E(\sigma_{\ell'-1})) = h(E(\sigma_\ell)) = f(\sigma_\ell).$$

Using Lemma E.4 for $s = \ell'$, this observation implies

$$f(\sigma_\ell) \geq f(\sigma^*) - 2d_{\text{in}} \cdot f(\sigma_\ell) \Rightarrow f(\sigma_\ell) \geq \frac{f(\sigma^*)}{2d_{\text{in}}+1} \ ,$$

which proves the theorem. Thus, in the rest of the proof we may assume that Algorithm 12 terminates because σ reaches a size larger than $k-2$.

Consider now the case that $2d_{\text{in}} + 1 \geq d_{\text{in}}k$. In this case

$$f(\sigma_\ell) \geq f(\sigma_1) = f(\sigma_0) + [f(\sigma_1) - f(\sigma_0)] \geq f(\sigma_0) + \frac{f(\sigma^*) - f(\sigma_0)}{d_{\text{in}}k}$$
$$\geq \frac{f(\sigma^*)}{d_{\text{in}}k} \geq \frac{f(\sigma^*)}{2d_{\text{in}} + 1} \geq \frac{1 - e^{-(1-\frac{1}{k})}}{2d_{\text{in}} + 1} \cdot f(\sigma^*) \ ,$$

where the first inequality holds since σ_1 is a prefix of σ_ℓ and the second inequality follows from Lemma E.4. Thus, it remains to prove the theorem in the more interesting case of $2d_{\text{in}} + 1 < d_{\text{in}}k$.

Observe that

$$[1 - (2d_{\text{in}} + 1)/(d_{\text{in}}k)]^\ell \leq e^{-(2+1/d_{\text{in}})\cdot(\ell/k)} \ .$$

Plugging this inequality into Corollary E.5 gives

$$f(\sigma_\ell) \geq \frac{f(\sigma^*)}{2d_{\text{in}} + 1} + \frac{[1 - (2d_{\text{in}} + 1)/(d_{\text{in}}k)]^\ell}{2d_{\text{in}} + 1} \cdot [(2d_{\text{in}} + 1)f(\sigma_0) - f(\sigma^*)]$$
$$\geq \frac{1 - [1 - (2d_{\text{in}} + 1)/(d_{\text{in}}k)]^\ell}{2d_{\text{in}} + 1} \cdot f(\sigma^*) \geq \frac{1 - e^{-(2+1/d_{\text{in}})\cdot(\ell/k)}}{2d_{\text{in}} + 1} \cdot f(\sigma^*) \ .$$

At this point we need a lower bound on ℓ. One can note that $|\sigma|$ starts as 0, increases by at most 2 in each iteration of the loop of Algorithm 12 and ends up with a value of at least $k - 1$ by our assumption. Thus, the number ℓ of iterations must be at least $(k - 1)/2$. Plugging this observation into the previous inequality gives

$$f(\sigma_\ell) \geq \frac{1 - e^{-(2+1/d_{\text{in}})\cdot(1-1/k)/2}}{2d_{\text{in}} + 1} \cdot f(\sigma^*) \geq \frac{1 - e^{-(1-1/k)}}{2d_{\text{in}} + 1} \cdot f(\sigma^*) \ . \qquad \square$$

E.2 Proof of Theorem 5.3

In this section we prove Theorem 5.3, however, before we get into the proof, let us first recall the theorem itself.

Theorem 5.3. *The approximation ratio of Algorithm 14 is at least* $\frac{1-e^{-(1-\frac{r}{k})}}{rd_{\text{in}}+1}$.

In the proof of this theorem we use the same notation that we used in Section F.2.2 for analyzing Algorithm 12. One can observe that the proofs of Observation E.1 and Observation E.2 are unaffected by the differences between Algorithm 12 and Algorithm 14, and thus, these two observations can also be used for towards the proof of Theorem 5.3.

The next lemma is analogous to Lemma F.8.

Lemma E.6. *For every $1 \leq s < \ell'$, $h((E(\sigma^*) \cap \mathcal{E}_{s+1}) \cup E(\sigma_s)) \geq h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) - rd_{\text{in}} \cdot h(e_s \mid \sigma_{s-1})$.*

Proof. Observe that the definition of $\mathcal{E}_s$ guarantees that $\mathcal{E}_{s+1} \subseteq \mathcal{E}_s$, for every $1 \leq s < \ell'$, because σ_{s-1} is a prefix of σ_s. In contrast, every vertex u that appears in σ_s but not in σ_{s-1} can be responsible for at most d_{in} arcs of $\mathcal{E}_s \setminus \mathcal{E}_{s+1}$ because u can be responsible for excluding an arc from $\mathcal{E}_{s+1}$ only if u is a non-first vertex of the arc. Since σ_s contains at most r vertices that do not appear in σ_{s-1}, this implies $|\mathcal{E}_s \setminus \mathcal{E}_{s+1}| \leq rd_{\text{in}}$.

Using these observations and the submodularity of h, we can now derive the following inequality.

$$\begin{aligned} h(E(\sigma^*) \cap \mathcal{E}_s \mid E(\sigma_{s-1})) - h(E(\sigma^*) \cap \mathcal{E}_{s+1} \mid E(\sigma_{s-1})) &\leq \sum_{e \in E(\sigma^*) \cap (\mathcal{E}_s \setminus \mathcal{E}_{i+1})} h(e \mid E(\sigma_{s-1})) \\ &\leq \sum_{e \in E(\sigma^*) \cap (\mathcal{E}_s \setminus \mathcal{E}_{i+1})} h(e_s \mid E(\sigma_{s-1})) = |E(\sigma^*) \cap (\mathcal{E}_s \setminus \mathcal{E}_{s+1})| \cdot h(e_s \mid E(\sigma_{s-1})) \\ &\leq |\mathcal{E}_s \setminus \mathcal{E}_{s+1}| \cdot h(e_s \mid E(\sigma_{s-1})) \leq rd_{\text{in}} \cdot h(e_s \mid E(\sigma_{s-1})) \ , \end{aligned}$$

where the second inequality follows from the definition of e_s which guarantees that it maximizes $h(e_s \mid E(\sigma_{s-1}))$ among all the edges of $\mathcal{E}_s$.

It now remains to observe that

$$\begin{aligned} &h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) - h((E(\sigma^*) \cap \mathcal{E}_{s+1}) \cup E(\sigma_s)) \\ &\qquad \leq h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) - h((E(\sigma^*) \cap \mathcal{E}_{s+1}) \cup E(\sigma_{s-1})) \\ &\qquad = h(E(\sigma^*) \cap \mathcal{E}_s \mid E(\sigma_{s-1})) - h(E(\sigma^*) \cap \mathcal{E}_{s+1} \mid E(\sigma_{s-1})) \leq rd_{\text{in}} \cdot h(e_s \mid E(\sigma_{s-1})) \ , \end{aligned}$$

where the first inequality follows from the monotonicity of h since the fact that σ_{s-1} is a prefix of σ_s implies $E(\sigma_{s-1}) \subseteq E(\sigma_s)$. □

We are now ready to prove the following analog of Lemma E.4.

Lemma E.7. *For every $1 \leq s \leq \ell$, the following two inequalities hold:*

- $h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) \geq f(\sigma^*) - rd_{\text{in}} \cdot f(\sigma_{s-1})$.
- $f(\sigma_s) - f(\sigma_{s-1}) \geq \frac{f(\sigma^*) - rd_{\text{in}} \cdot [f(\sigma_{s-1}) - f(\sigma_0)] - f(\sigma_{s-1})}{d_{\text{in}} k} \geq \frac{f(\sigma^*) - (rd_{\text{in}}+1) \cdot f(\sigma_{s-1})}{d_{\text{in}} k}$.

Moreover, the first inequality holds also for $s = \ell'$.

Proof. Lemma E.6 shows that, for every $1 \leq t < \ell'$, we have

$$h((E(\sigma^*) \cap \mathcal{E}_{t+1}) \cup E(\sigma_t)) \geq h((E(\sigma^*) \cap \mathcal{E}_t) \cup E(\sigma_{t-1})) - rd_{\text{in}} \cdot h(e_t \mid \sigma_{t-1}) \ .$$

Adding up this inequality for $1 \leq t < s$ gives us

$$\begin{aligned} h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) &\geq h((E(\sigma^*) \cap \mathcal{E}_1) \cup E(\sigma_0)) - rd_{\text{in}} \cdot \sum_{t=1}^{s-1} h(e_s \mid \sigma_{s-1}) \\ &= f(\sigma^*) - rd_{\text{in}} \cdot \sum_{t=1}^{s-1} h(e_s \mid \sigma_{s-1}) \geq f(\sigma^*) - rd_{\text{in}} \cdot \sum_{t=1}^{s-1} [f(\sigma_t) - f(\sigma_{t-1})] \\ &= f(\sigma^*) - rd_{\text{in}} \cdot [f(\sigma_{s-1}) - f(\sigma_0)] \geq f(\sigma^*) - rd_{\text{in}} \cdot f(\sigma_{s-1}) \ . \end{aligned}$$

The first equality follows since the fact that σ_0 is empty implies $E(\sigma_0) = \varnothing$ and $E(\sigma^*) \cap \mathcal{E}_1 = E(\sigma^*)$. Additionally, the second inequality follows from Observation E.1, and the last inequality follows from the non-negativity of f. This proves that the first inequality of the lemma holds for every $1 \leq s \leq \ell'$. In the rest of the proof we aim to prove the second inequality, and thus, assume $1 \leq s \leq \ell$.

Recall now that e_s is the edge of $\mathcal{E}_s$ maximizing $h(e_s \mid E(\sigma_{s-1}))$ and that the size of $E(\sigma^*) \cap \mathcal{E}_s \subseteq E(\sigma^*)$ is at most $d_{\text{in}}k$ by Observation E.2. Thus, by the submodularity of h,

$$\begin{aligned} h(e_s \mid E(\sigma_{s-1})) &\geq \frac{\sum_{e \in E(\sigma^*) \cap \mathcal{E}_i} h(e \mid E(\sigma_{s-1}))}{|E(\sigma^*) \cap \mathcal{E}_s|} \geq \frac{h(E(\sigma^*) \cap \mathcal{E}_s \mid E(\sigma_{s-1}))}{d_{\text{in}}k} \\ &\geq \frac{h((E(\sigma^*) \cap \mathcal{E}_s) \cup E(\sigma_{s-1})) - h(E(\sigma_{s-1}))}{d_{\text{in}}k} \\ &\geq \frac{\{f(\sigma^*) - rd_{\text{in}} \cdot [f(\sigma_{s-1}) - f(\sigma_0)]\} - f(\sigma_{s-1})}{d_{\text{in}}k} \geq \frac{f(\sigma^*) - (rd_{\text{in}} + 1) \cdot f(\sigma_{s-1})}{d_{\text{in}}k} \ . \end{aligned}$$

The second inequality of the lemma now follows by combining the last inequality with Observation E.1. □

Corollary E.8. *If* $rd_{\text{in}} + 1 < d_{\text{in}}k$, *then* $f(\sigma_\ell) \geq \frac{f(\sigma^*)}{rd_{\text{in}}+1} + \frac{[1-(rd_{\text{in}}+1)/(d_{\text{in}}k)]^\ell}{rd_{\text{in}}+1} \cdot [(rd_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)]$.

Proof. To prove the corollary, we prove by induction the stronger claim that, for every $0 \leq s \leq \ell$,

$$f(\sigma_s) \geq \frac{f(\sigma^*)}{rd_{\text{in}} + 1} + \frac{[1 - (rd_{\text{in}} + 1)/(d_{\text{in}}k)]^s}{2rd_{\text{in}} + 1} \cdot [(rd_{\text{in}} + 1)f(\sigma_0) - f(\sigma^*)] \ .$$

For $s = 0$ this inequality is true since

$$f(\sigma_0) = \frac{f(\sigma^*)}{rd_{\text{in}}+1} + \frac{1}{rd_{\text{in}}+1} \cdot [(rd_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)]$$
$$= \frac{f(\sigma^*)}{rd_{\text{in}}+1} + \frac{[1-(rd_{\text{in}}+1)/(d_{\text{in}}k)]^0}{rd_{\text{in}}+1} \cdot [(rd_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)] \ .$$

Assume now that the claim holds for $s-1 \geq 0$, and let us prove it for s. By Lemma E.7,

$$f(\sigma_s) \geq f(\sigma_{s-1}) + \frac{f(\sigma^*) - (rd_{\text{in}}+1) \cdot f(\sigma_{s-1})}{d_{\text{in}}k} = \left(1 - \frac{rd_{\text{in}}+1}{d_{\text{in}}k}\right) \cdot f(\sigma_{s-1}) + \frac{f(\sigma^*)}{d_{\text{in}}k} \ .$$

Plugging in the induction hypothesis, we get

$$f(\sigma_i) \geq \left(1 - \frac{rd_{\text{in}}+1}{d_{\text{in}}k}\right) \cdot \left\{ \frac{f(\sigma^*)}{rd_{\text{in}}+1} + \frac{[1-(rd_{\text{in}}+1)/(d_{\text{in}}k)]^{s-1}}{rd_{\text{in}}+1} \cdot [(rd_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)] \right\}$$
$$+ \frac{f(\sigma^*)}{d_{\text{in}}k} = \frac{f(\sigma^*)}{rd_{\text{in}}+1} + \frac{[1-(rd_{\text{in}}+1)/(d_{\text{in}}k)]^s}{rd_{\text{in}}+1} \cdot [(rd_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)] \ . \qquad \square$$

We are now ready to prove Theorem 5.3.

Proof of Theorem 5.3. First, we need to consider the case that Algorithm 14 terminates because the set $\mathcal{E}$ becomes empty. In this case $\mathcal{E}_{\ell'} = \varnothing$, which implies

$$h((E(\sigma^*) \cap \mathcal{E}_{\ell'}) \cup E(\sigma_{\ell'-1})) = h(E(\sigma_\ell)) = f(\sigma_\ell).$$

Using Lemma E.7 for $s = \ell' = \ell + 1$, this observation implies

$$f(\sigma_\ell) \geq f(\sigma^*) - rd_{\text{in}} \cdot f(\sigma_\ell) \Rightarrow f(\sigma_\ell) \geq \frac{f(\sigma^*)}{rd_{\text{in}}+1} \ ,$$

which proves the theorem. Thus, in the rest of the proof we may assume that Algorithm 14 terminated because σ reached a size larger than $k - r$.

Consider now the case that $rd_{\text{in}} + 1 \geq d_{\text{in}}k$. In this case

$$f(\sigma_\ell) \geq f(\sigma_1) = f(\sigma_0) + [f(\sigma_1) - f(\sigma_0)] \geq f(\sigma_0) + \frac{f(\sigma^*) - f(\sigma_0)}{d_{\text{in}}k}$$
$$\geq \frac{f(\sigma^*)}{d_{\text{in}}k} \geq \frac{f(\sigma^*)}{rd_{\text{in}}+1} \geq \frac{1 - e^{-(1-\frac{r}{k})}}{rd_{\text{in}}+1} \cdot f(\sigma^*) \ ,$$

where the first inequality holds since σ_1 is a prefix of σ_ℓ and the second inequality follows from

Lemma E.7. Thus, it remains to prove the theorem in the more interesting case of $rd_{\text{in}} + 1 < d_{\text{in}}k$.

Observe that

$$[1 - (rd_{\text{in}} + 1)/(d_{\text{in}}k)]^{\ell} \leq e^{-(r+1/d_{\text{in}})\cdot(\ell/k)} \ .$$

Plugging this inequality into Corollary E.8 gives

$$\begin{aligned} f(\sigma_\ell) &\geq \frac{f(\sigma^*)}{rd_{\text{in}}+1} + \frac{[1-(rd_{\text{in}}+1)/(d_{\text{in}}k)]^{\ell}}{rd_{\text{in}}+1} \cdot [(rd_{\text{in}}+1)f(\sigma_0) - f(\sigma^*)] \\ &\geq \frac{1-[1-(rd_{\text{in}}+1)/(d_{\text{in}}k)]^{\ell}}{rd_{\text{in}}+1} \cdot f(\sigma^*) \geq \frac{1-e^{-(r+1/d_{\text{in}})\cdot(\ell/k)}}{rd_{\text{in}}+1} \cdot f(\sigma^*) \ . \end{aligned}$$

At this point we need a lower bound on ℓ. One can note that $|\sigma|$ starts as 0, increases by at most r in each iteration of the loop of Algorithm 14 and ends up with a value of at least $k - r + 1$ by our assumption. Thus, the number ℓ of iterations must be at least $(k - r + 1)/r$. Plugging this observation into the previous inequality gives

$$f(\sigma_\ell) \geq \frac{1-e^{-(r+1/d_{\text{in}})\cdot(1-(r-1)/k)/r}}{rd_{\text{in}}+1} \cdot f(\sigma^*) \geq \frac{1-e^{-(1-r/k)}}{rd_{\text{in}}+1} \cdot f(\sigma^*) \ . \qquad \square$$

Appendix F. Adaptive Sequence Submodularity Appendix

F.1 Table of Notations

Table 6

V	Ground set of elements.
$e \in V$	An individual element from V.
ϕ	A realization, i.e., a function from elements to states.
ψ	A partial realization to encoding the current set of observations.
$\text{dom}(\psi)$	Domain of a partial realization ψ is defined as $\text{dom}(\psi) = \{e : \exists o \text{ .s.t. } (o, e) \in \psi\}$.
Φ, Ψ	A random realization and a random partial realization, respectively.
$\sim$	For a realization ϕ and a partial realization ψ: $\phi \sim \psi$ means $\psi(e) = \phi(e)$ for all $e \in \text{dom}(\psi)$.
$p(\phi)$	The probability distribution on realizations.
$p(\phi \mid \psi)$	The conditional distribution on realizations: $p(\phi \mid \psi) \triangleq \Pr[\Phi = \phi \mid \Phi \sim \psi]$.
π	A policy, which maps partial realizations to items.
$E(\pi, \phi)$	The set of all edges induced by π when run under realization ϕ.
h	An objective function of type $h : 2^V \times O^V \to \mathbb{R}_{\geq 0}$.
$\Delta(e \mid \psi)$	The conditional expected marginal benefit of e conditioned on ψ.
k	Budget on the number of selected items.

F.2 Proofs

In this section, we prove Theorems 5.5 and 5.6. Towards this goal, we first state some necessary definitions and notations, and present a few results regarding weakly adaptive submodular functions.

F.2.1 Weakly Adaptive Sequence Submodular

Notation The random variable Φ denotes a random realization with respect to the distribution $p(\Phi = \phi)$ over the items (or equivalently vertices of the graph).[5] For a set A, its partial realization (i.e., items in A and their corresponding states) is shown by $\psi_A = \{(e, O(e)) \mid e \in A\}$, where $O(e)$ gives the state of e. For a partial realization ψ, we define $\text{dom}(\psi) = \{e : \exists\ o \text{ s.t. } (o, e) \in \psi\}$. We use Ψ_A to denote a random partial realization over a set A. Note that the distribution of random variable Ψ_A is uniquely defined by the distribution of random variable Φ. A partial realization ψ is consistent with a realization ϕ (we write $\phi \sim \psi$) if they are equal, i.e., they are in the same state, everywhere in the domain of ψ. For the ease of notation, we define $h(\psi) \triangleq h(\text{dom}(\psi), O(\psi))$, where $O(\psi)$ is the state of items in the realization ψ. We also define $h_{avg}(A) \triangleq \mathbb{E}_\Phi(h(A)) \triangleq \mathbb{E}_\Phi[h(\Phi_A)]$ which is the expected utility of set A (and states of its elements) over all possible realizations of A under the probability distribution $p(\Phi = \phi)$. We define $\Delta(e \mid \psi) = \mathbb{E}_{\Phi \sim \psi}[h(\Psi_{\{e\}} + \psi) - h(\psi)]$ which is the conditional expected marginal benefit of item e conditioned on having observed the subrealization ψ. Note that the random variable $\Psi_{\{e\}}$ is the state of item e with respect to the probability distribution $p(\Phi = \phi \mid \Phi \sim \psi)$. Similarly, we define $\Delta(A \mid \psi) = \mathbb{E}_{\Phi \sim \psi}[h(\Psi_A + \psi) - h(\psi)]$ which is the expected marginal gain of set A to the partial realization ψ. Assume $E(\pi_\phi)$ is the set of edges induced by the set of items policy π selects under the realization ϕ. The expected utility of policy π is defined as $f_{avg}(\pi) \triangleq h_{avg}(E(\pi)) = \mathbb{E}_\Phi[h(E(\pi_\Phi)]$, where the expectation is taken with respect to $p(\Phi = \phi)$. For a list of all the notations used in the section refer to Table 6 in Appendix F.1.

Next, we restate the definitions for weakly adaptive set submodular and adaptive monotone functions.

Definition 5.1. *A function $h : 2^E \times Q^E \to \mathbb{R}_{\geq 0}$ is **weakly adaptive set submodular** with*

5. Note that there is a one to one correspondence between a realization ϕ over the vertices and a realization ϕ^E over the edges.

parameter γ if for all sets $A \subseteq E$ and for all $\psi \subseteq \psi'$ we have:

$$\Delta(A \mid \psi') \leq \frac{1}{\gamma} \cdot \sum_{e \in A} \Delta(e \mid \psi).$$

Definition 5.2. *A function $h : 2^E \times Q^E \to \mathbb{R}_{\geq 0}$ is **adaptive monotone** if $\Delta(e \mid \psi) \geq 0$ for all partial realizations ψ. That is, the conditional expected marginal benefit of any element is non-negative.*

Definition 5.1 is the generalization of both weak submodularity Das and Kempe (2011) and adaptive submodularity Golovin and Krause (2011) concepts.

Next, we state a few useful claims regarding weakly adaptive submodular functions.

First note for all ψ and for every set $A \subseteq V \setminus \mathrm{dom}(\psi)$, from Definition 5.1 and the fact that $\psi \subseteq \psi$, we have

$$\Delta(A \mid \psi) \leq \frac{1}{\gamma} \cdot \sum_{e \in A} \Delta(e \mid \psi). \tag{23}$$

Lemma F.1. *For all ψ and $A \subseteq B \subseteq V \setminus dom(\psi)$, we have*

$$\Delta(B \mid \psi) - \Delta(A \mid \psi) \leq \frac{1}{\gamma} \cdot \sum_{e \in B \setminus A} \Delta(e \mid \psi).$$

Proof. We have

$$\begin{aligned}
\Delta(B \mid \psi) - \Delta(A \mid \psi) &= \sum \Pr[\Psi_A = \psi' \mid \Phi \sim \psi] \cdot \sum \Pr[\Psi_{B \setminus A} = \psi'' \mid \Phi \sim \psi + \psi'] \\
&\qquad \cdot (h_{avg}(\psi + \psi' + \psi'') - h_{avg}(\psi + \psi')) \\
&= \sum \Pr[\Psi_A = \psi' \mid \Phi \sim \psi] \cdot \Delta(B \setminus A \mid \psi + \psi') \leq \frac{1}{\gamma} \cdot \sum_{e \in B \setminus A} \Delta(e \mid \psi),
\end{aligned}$$

where the inequality is derived from the definition of weakly adaptive set submodular functions (see Definition 5.1) and the fact that $\sum \Pr[\Psi_A = \psi' \mid \Phi \sim \psi] = 1$.

□

Corollary F.2. *For all ψ, $e^* = \arg\max_{e \in V} \Delta(e \mid \psi)$ and two random subsets $A \subseteq B \subseteq V \setminus dom(\psi)$*

whose randomness might depend on the realization, we have

$$\mathbb{E}[\Delta(B \mid \psi) - \Delta(A \mid \psi) \mid \Phi \sim \psi] \leq \frac{\mathbb{E}[|B \setminus A| \mid \Phi \sim \psi]}{\gamma} \cdot \Delta(e^* \mid \psi).$$

Proof. By taking expectation over the guarantee of Lemma F.1, we get

$$\begin{aligned}
\mathbb{E}[\Delta(B \mid \psi) - \Delta(A \mid \psi) \mid \Phi \sim \psi] &\leq \frac{1}{\gamma} \cdot \mathbb{E}\left[\sum_{e \in B \setminus A} \Delta(e \mid \psi) \mid \Phi \sim \psi\right] \\
&\leq \frac{1}{\gamma} \cdot \mathbb{E}\left[\sum_{e \in B \setminus A} \Delta(e^* \mid \psi) \mid \Phi \sim \psi\right] \\
&= \frac{\mathbb{E}[|B \setminus A| \mid \Phi \sim \psi]}{\gamma} \cdot \Delta(e^* \mid \psi),
\end{aligned}$$

where the second inequality follows from the fact that e^* is the element with the largest expected gain. □

The following observation is an immediate consequence of Definition 5.2.

Observation F.3. *For any two (possibly random) subsets $A \subseteq B \subseteq V$, we have*

$$\mathbb{E}_\Phi(h(A)) \leq \mathbb{E}_\Phi(h(B)).$$

Lemma F.4. *Assume h is adaptive monotone and weakly adaptive set submodular with a parameter γ with respect to the distribution $p(\phi)$, and π is a greedy policy which picks the item with the largest expected marginal gain at each step, then for all policies π^* we have*

$$h_{avg}(\pi) \geq \left(1 - e^{-1/\gamma}\right) \cdot h_{avg}(\pi^*).$$

Proof. The proof of this lemma follows the same line of argument as the proof of (Golovin and Krause, 2011, Theroem 5). □

F.2.2 Proof of Theorem 5.5

In this section, we first restate Theorem 5.5 and then prove it.

Theorem 5.5. *For adaptive monotone and weakly adaptive sequence submodular function f, the*

Adaptive Sequence Greedy policy π represented by Algorithm 15 achieves

$$f_{avg}(\pi) \geq \frac{\gamma}{2d_{\text{in}} + \gamma} \cdot f_{avg}(\pi^*),$$

where γ is the weakly adaptive submodularity parameter, π^* is the policy with the highest expected value and d_{in} is the largest in-degree of the input graph G.

We assume the function h is weakly adaptive set submodular (with a parameter γ) and monotone adaptive submodular. Furthermore, we assume π^* is the optimal policy. It means π^* maximizes the expected gain over the distribution Φ.

Let $\ell = \lceil k/2 \rceil$. For every $0 \leq s \leq \ell$, let π_s be the set of items picked by the greedy policy π after s iterations (if the algorithm does not make that many iterations because the set $\mathcal{E}$ became empty at some earlier point, then we assume for the sake of the proof that the algorithm continues to make dummy iterations after the point in which $\mathcal{E}$ becomes empty, and in the dummy iterations it picks no items). The observed partial realization of edges after s iterations of the algorithm is represented by ψ_s. The random variable representing ψ_s is Ψ_s. We define $f_{avg}(\pi_s) \triangleq h_{avg}(E(\pi_s))$, i.e., it is the expected value of items picked by the greedy policy π after s iterations. For every $1 \leq s \leq \ell$, we also denote by e_s and $\mathcal{E}_s$ the values assigned to the variables e_{ij} and $\mathcal{E}$, respectively, at iteration number s. Finally, we assume e_s is a dummy arc with zero marginal contribution to h if iteration number s is a dummy iteration (i.e., the algorithm makes in reality less than s iterations).

Observation F.5. *For every $0 \leq s_1 \leq s_2 \leq \ell$, conditioned on the partial realization ψ_{s_1}, i.e., the policy has already made its first s_1 iterations, we have $\mathcal{E}_{s_1} \supseteq \mathcal{E}_{s_2}$ and $E(\pi_{s_1}) \subseteq E(\pi_{s_2})$.*

Proof. Both properties guaranteed by the observation follow from the fact that: for all possible realization $\phi \sim \psi_{s_1}$, we have that π_{s_1} is a (possibly trivial) prefix of π_{s_2}. □

Lemma F.6. *For every $1 \leq s \leq \ell$, $f_{avg}(\pi_s) - f_{avg}(\pi_{s-1}) \geq \mathbb{E}_{\Psi_{s-1}}[\Delta(e_s \mid \Psi_{s-1})]$.*

Proof. Consider a fixed sub-realization ψ_{s-1}. If e_s is a dummy arc, then $\pi_s = \pi_{s-1}$, and the observation is trivial. Otherwise, notice that the membership of e_s in $\mathcal{E}_{s-1}$ guarantees that it does not belong to $E(\pi_{s-1}) = \text{dom}(\psi_{s-1})$, but does belong to $E(\sigma_s)$. Together with the fact that $E(\pi_{s-1}) \subseteq E(\pi_s)$ by Observation F.5, we get $E(\pi_{s-1}) + e_s \subseteq E(\pi_s)$; which implies, by the adaptive

monotonicity of h,

$$\begin{aligned} f_{avg}(\pi_s) - f(\pi_{s-1}) &= \mathbb{E}_{\Phi\sim\psi_{s-1}}[f_{avg}(\pi_s)] - h(\psi_{s-1}) \\ &\geq \mathbb{E}_{\Phi\sim\psi_{s-1}}[h(\psi_{s-1}+e_s)] - h(\psi_{s-1}) \\ &= \Delta(e_s \mid \psi_{s-1}). \end{aligned}$$ □

Note that we condition on the fact that the first $s-1$ steps of the policy π are performed, therefore we have $f_{avg}(\pi_{s-1}) = h(\psi_{s-1})$. By taking expectation over all the possible realizations of the random variable Ψ_{s-1} the lemma is proven.

Lemma F.7. *Conditioned on any arbitrary partial realization ψ, we have $\mathbb{E}_{\Phi\sim\psi}[|E(\pi^*)|] \leq (k-1)d_{\text{in}}$.*

Proof. The optimal policy under each realization of the random variable Φ chooses at most k items. Each one of these k items (except the first one) will have at most d_{in} incoming edges. Therefore, the expected number of edges is at most $(k-1)d_{\text{in}}$. □

Lemma F.8. *For every $1 \leq s \leq \ell$, we have*

$$\begin{aligned} &\mathbb{E}_\Phi[h((E(\pi^*)\cap\mathcal{E}_{s-1})\cup E(\pi_{s-1}))] \leq \\ &\qquad \mathbb{E}_\Phi[h((E(\pi^*)\cap\mathcal{E}_s)\cup E(\pi_s))] + \frac{1}{\gamma}\cdot\mathbb{E}_\Phi[|E(\pi^*)\cap(\mathcal{E}_{s-1}\setminus\mathcal{E}_s)|\cdot\Delta(e_s \mid E(\pi_{s-1}))]. \end{aligned}$$

Note that the expectation is taken over all the possible realizations of the random variable Φ.

Proof. The lemma follows by combining the two inequalities of Equation 24 and Equation 25.

$$\begin{aligned} &\mathbb{E}_\Phi[\Delta(E(\pi^*)\cap\mathcal{E}_{s-1} \mid E(\pi_{s-1}))] - \mathbb{E}_\Phi[\Delta(E(\pi^*)\cap\mathcal{E}_s \mid E(\pi_{s-1}))] \qquad (24) \\ &= \sum \Pr[\Psi_{s-1}=\psi_{s-1}]\cdot\left[\mathbb{E}_{\Phi\sim\psi_{s-1}}[\Delta(E(\pi^*)\cap\mathcal{E}_{s-1} \mid \psi_{s-1}) - \Delta(E(\pi^*)\cap\mathcal{E}_s \mid \psi_{s-1})]\right] \\ &\overset{(a)}{\leq} \frac{1}{\gamma}\sum \Pr[\Psi_{s-1}=\psi_{s-1}]\cdot\mathbb{E}_{\Phi\sim\psi_{s-1}}[|E(\pi^*)\cap(\mathcal{E}_{s-1}\setminus\mathcal{E}_s)|\cdot\Delta(e_s \mid \psi_{s-1})] \\ &= \frac{1}{\gamma}\cdot\mathbb{E}_\Phi[|E(\pi^*)\cap(\mathcal{E}_{s-1}\setminus\mathcal{E}_s)|\cdot\Delta(e_s \mid E(\pi_{s-1}))]. \end{aligned}$$

To see why inequality (a) is true, note that for every given sub realization ψ_{s-1} we have: (i) if e_s is a dummy edge, then $(E(\pi^*)\cap\mathcal{E}_{s-1})\cup E(\pi_{s-1}) = (E(\pi^*)\cap\mathcal{E}_s)\cup E(\pi_s)$, which makes (a) trivial, or (ii) when e_s is not dummy, (a) results from Corollary F.2.

$$\begin{aligned}
&\mathbb{E}_\Phi[h((E(\pi^*)\cap\mathcal{E}_{s-1})\cup E(\pi_{s-1}))] - \mathbb{E}_\Phi[h((E(\pi^*)\cap\mathcal{E}_s)\cup E(\pi_s))] \qquad (25)\\
&\le \mathbb{E}_\Phi[h((E(\pi^*)\cap\mathcal{E}_{s-1})\cup E(\pi_{s-1}))] - \mathbb{E}_\Phi[h((E(\pi^*)\cap\mathcal{E}_s)\cup E(\pi_{s-1}))])\\
&= \sum \Pr[\Psi_{s-1}=\psi_{s-1}]\cdot \mathbb{E}_{\Phi\sim\psi_{s-1}}[h((E(\pi^*)\cap\mathcal{E}_{s-1})\cup\psi_{s-1}) - h(E(\pi^*)\cap\mathcal{E}_s)\cup\psi_{s-1})]\\
&= \sum \Pr[\Psi_{s-1}=\psi_{s-1}]\cdot \mathbb{E}_{\Phi\sim\psi_{s-1}}[\Delta((E(\pi^*)\cap\mathcal{E}_{s-1})\mid\psi_{s-1}) - \Delta(E(\pi^*)\cap\mathcal{E}_s)\mid\psi_{s-1})]\\
&= \mathbb{E}_\Phi[\Delta(E(\pi^*)\cap\mathcal{E}_{s-1}\mid E(\pi_{s-1}))] - \mathbb{E}_\Phi[\Delta(E(\pi^*)\cap\mathcal{E}_s\mid E(\pi_{s-1}))]. \qquad \square
\end{aligned}$$

Lemma F.9. $\mathbb{E}_\Phi[h((E(\pi^*)\cap\mathcal{E}_\ell)\cup E(\pi_\ell))]] \le \frac{1}{\gamma}\cdot\mathbb{E}_\Phi[|E(\pi^*)\cap\mathcal{E}_\ell|\cdot\Delta(e_\ell\mid\Psi_{\ell-1})] + f_{avg}(\pi_\ell)$.

Proof. We have

$$\begin{aligned}
&\mathbb{E}_\Phi[h((E(\pi^*)\cap\mathcal{E}_\ell)\cup E(\pi_\ell)) - h(\pi_\ell)]\\
&= \sum \Pr[\Psi_\ell=\psi_\ell]\cdot\mathbb{E}_{\Phi\sim\psi_\ell}[h(E(\pi^*)\cap\mathcal{E}_\ell)\cup\psi_\ell) - h(\psi_\ell)]\\
&\overset{(a)}{\le} \frac{1}{\gamma}\sum \Pr[\Psi_\ell=\psi_\ell]\cdot\mathbb{E}_{\Phi\sim\psi_\ell}[|E(\pi^*)\cap\mathcal{E}_\ell|\cdot\Delta(e_\ell\mid\psi_{\ell-1})] = \frac{1}{\gamma}\cdot\mathbb{E}_\Phi[|E(\pi^*)\cap\mathcal{E}_\ell|\cdot\Delta(e_\ell\mid\Psi_{\ell-1})].
\end{aligned}$$

To see why inequality (a) is true, note that for every given sub realization ψ_ℓ we have: (i) if e_ℓ is a dummy edge, then $\mathcal{E}_\ell=\varnothing$, which makes inequality (a) trivial, and (ii) if e_ℓ is not a dummy edge then we conclude inequality (a) from the definition of weakly adaptive set submodular functions (see Definition 5.1).

The lemma follows by combining this inequality with the observation that $f_{avg}(\pi_\ell)=\mathbb{E}_\Phi[h(\pi_\ell)]$. $\square$

To combine the last two lemmata, we need the following observation.

Observation F.10. *For every* $2\le s\le\ell$, $\mathbb{E}_\Phi[\Delta(e_{s-1}\mid E(\pi_{s-2}))]\ge\gamma\cdot\mathbb{E}_\Phi[\Delta(e_s\mid E(\pi_{s-1}))]$.

We are now ready to prove Theorem 5.5.

Proof of Theorem 5.5. Combining Lemmata F.8 and F.9, we get

$$\begin{aligned}
f_{avg}(\pi^*) - f_{avg}(\pi_\ell) =& \mathbb{E}_\Phi[h((E(\pi^*)\cap\mathcal{E}_1)\cup E(\pi_0))] - f_{avg}(\pi_\ell)\\
&\le \frac{1}{\gamma}\cdot\sum_{s=1}^{\ell}\mathbb{E}_\Phi[|E(\pi^*)\cap(\mathcal{E}_{s-1}\setminus\mathcal{E}_s)|\cdot\Delta(e_s\mid E(\pi_{s-1}))]\\
&\qquad + \mathbb{E}_\Phi[\Delta((E(\pi^*)\cap\mathcal{E}_s)\cup E(\pi_s))] - f_{avg}(\pi_\ell)
\end{aligned}$$

$$\leq \frac{1}{\gamma}\sum_{s=1}^{\ell}\mathbb{E}_{\Phi}[|E(\pi^*)\cap(\mathcal{E}_{s-1}\setminus\mathcal{E}_s)|\cdot\Delta(e_s\mid E(\pi_{s-1}))] + \frac{1}{\gamma}\cdot\mathbb{E}_{\Phi}[|E(\pi^*)\cap\mathcal{E}_{\ell}|\cdot\Delta(e_{\ell}\mid\Psi_{\ell-1})]$$

$$= \frac{1}{\gamma}\cdot\sum_{s=1}^{\ell-1}\mathbb{E}_{\Phi}[|E(\pi^*)\cap(\mathcal{E}_0\setminus\mathcal{E}_s)|\cdot[\Delta(e_s\mid E(\pi_{s-1}))-\Delta(e_{s+1}\mid E(\pi_s))]] + \frac{1}{\gamma}\cdot\mathbb{E}_{\Phi}[|E(\pi^*)\cap\mathcal{E}_0|\cdot\Delta(e_{\ell}\mid E(\pi_{\ell-1}))], \quad (26)$$

where the first equality holds since the fact that σ_0 is an empty sequence implies $E(\sigma_0)=\varnothing$ and $\mathcal{E}_0=E$, and the second equality holds since $\mathcal{E}_s\subseteq\mathcal{E}_{s-1}$ by Observation F.5 for every $1\leq s\leq\ell$. We now observe that for every $1\leq s\leq\ell$, π_s contains at most $2s$ vertices. Since each one of these vertices can be the end point of at most d_{in} arcs, we get

$$|E(\sigma^*)\cap(\mathcal{E}_0\setminus\mathcal{E}_s)|\leq|\mathcal{E}_0\setminus\mathcal{E}_s|\leq 2sd_{\text{in}}$$

Additionally, by Lemma F.7,

$$|E(\sigma^*)\cap\mathcal{E}_0|\leq|E(\sigma^*)|\leq(k-1)d_{\text{in}}\leq 2\ell d_{\text{in}}.$$

Plugging the last two inequalities into Inequality (26) yields

$$\begin{aligned} f_{avg}(\pi^*)-f_{avg}(\pi_{\ell}) &\leq \sum_{s=1}^{\ell-1}\frac{2sd_{\text{in}}}{\gamma}\cdot\mathbb{E}_{\Phi}[\Delta(e_s\mid E(\pi_{s-1}))-\Delta(e_{s+1}\mid E(\pi_s))] + \frac{2\ell d_{\text{in}}}{\gamma}\cdot\mathbb{E}_{\Phi}[\Delta(e_{\ell}\mid E(\sigma_{\ell-1}))] \\ &= \sum_{s=1}^{\ell}\frac{2d_{\text{in}}}{\gamma}\cdot\mathbb{E}_{\Phi}[\Delta(e_s\mid E(\pi_{s-1}))] \leq \frac{2d_{\text{in}}}{\gamma}\cdot\sum_{s=1}^{\ell}[f_{avg}(\pi_s)-f_{avg}(\pi_{s-1})] \\ &= \frac{2d_{\text{in}}}{\gamma}\cdot[f_{avg}(\pi_{\ell})-f_{avg}(\pi_0)] \leq \frac{2d_{\text{in}}}{\gamma}\cdot f_{avg}(\pi_{\ell}), \end{aligned}$$

where the second inequality holds due to Lemma F.6 and the last inequality follows from the non-negativity of f. Rearranging the last inequality, we get

$$f_{avg}(\pi_{\ell})\geq\frac{\gamma}{2d_{\text{in}}+\gamma}\cdot f_{avg}(\pi^*),$$

which implies the theorem since $f_{avg}(\pi_{\ell})$ is a lower bound on the expected value of the output sequence of Algorithm 15 because σ_{ℓ} is always a prefix of this sequence. □

F.2.3 Proof of Theorem 5.6

In this section, we first restate and then prove Theorem 5.6 which guarantees the performance of our proposed policy applied to hypergraphs.

Theorem 5.6. *For adaptive monotone and weakly adaptive sequence submodular function f, the policy π' represented by Algorithm 20 achieves*

$$f_{avg}(\pi') \geq \frac{\gamma}{rd_{\text{in}} + \gamma} \cdot f_{avg}(\pi^*),$$

where γ is the weakly adaptive submodularity parameter, π^* is the policy with the highest expected value and r is the size of the largest hyperedge in the input hypergraph.

Algorithm 20 Adaptive Hyper Sequence Greedy

Require: Directed hypergraph $H(V, E)$, γ-adaptive and adaptive-monotone function $h : 2^E \times O^E \to \mathbb{R}_{\geq 0}$ and cardinality parameter k
Let $\sigma \leftarrow ()$
while $|\sigma| \leq k - r$ **do**
    $\mathcal{E} = \{e \in E \mid \sigma \cap V(e)$ is a prefix of $e\}$
    if $\mathcal{E} \neq \emptyset$ **then**
        $e^* = \arg\max_{e \in \mathcal{E}} \Delta(e \mid \psi_\sigma)$
        for every v $\in e^*$ in order **do**
            if $v \notin \sigma$ **then**
                $\sigma = \sigma \oplus v$
        Identify the state of all edges in $\mathcal{E}' = \{e \in E \mid$ all elements of $V(e)$ belong to σ and appear in the same order$\}$
        $\psi_\sigma = \psi_{\mathcal{E}'}$
    else
        break
Return σ

In the proof of this theorem we use the same notation that we used in Section F.2.2 for analyzing Algorithm 15, with the exception of $\mathcal{E}_s$, which is now defined as $\mathcal{E}_s = \{e \in E \mid \sigma_s \cap V(e)$ is a prefix of $e\}$, and ℓ, which is now defined as $\lfloor k/r \rfloor$.

The following lemma is a counterpart of Lemma F.7.

Lemma F.11. $|E(\sigma^*)| \leq (k - r + 1)d_{\text{in}}$.

Proof. For a realization ϕ, every arc of π^* must end at a vertex of π^* which is not one of the first $r - 1$ vertices. The observation follows since π^* contains at most $k - r + 1$ vertices of this kind, and at most d_{in} arcs can end at each one of them. □

One can observe that the proofs of all the other observations and lemmata of Section F.2.2 are unaffected by the differences between Algorithm 15 and Algorithm 20, and thus, these observations and lemmata can be used towards the proof of Theorem 5.6.

Proof of Theorem 5.6. The proof of this theorem is identical to the proof of Theorem 5.5 up to two changes. First, instead of getting an upper bound of $2sd_{\text{in}}$ on $|\mathcal{E}_0 \setminus \mathcal{E}_s|$ for every $1 \leq s \leq \ell$, we now get an upper bound of rsd_{in} on this expression because σ_s might contain up to rs vertices rather than only $2s$. Second, instead of getting an upper bound of $2\ell d_{\text{in}}$ on $|E(\sigma^*)|$, we now use Lemma F.11 to get an upper bound of $(k-r+1)d_{\text{in}} \leq r\ell d_{\text{in}}$ on this expression. □

F.2.4 Proof of Theorem 5.7

The approximability of the sequence submodular maximization, as a generalization of the densest k subgraph problem (DkS) (Kortsarz and Peleg, 1993), is an open theoretical question with important implications. In this section, we prove Theorem 5.7.

In the DkS problem the goal is to find a subgraph on exactly k vertices that contains the maximum number of edges. DkS as a generalization of the k-clique problem is NP-hard and the best polynomial algorithm for DkS achieves a $n^{1/4+\epsilon}$ approximation factor[6] for an arbitrary $\epsilon > 0$ (Bhaskara et al., 2010). Furthermore, there exists no polynomial time algorithm that approximates DkS within an $O(n^{1/(\log\log n)^c})$ factor unless 3-SAT has a subexponential time algorithm Manurangsi (2017).

Lemma F.12. *Any algorithm with an α approximation factor to the sequence submodular maximization problem solves the densest k subgraph problem (DkS) with at most an α approximation factor.*

Proof. To prove this lemma, we show that for each instance of DkS over a directed graph $G(V, E)$ we can build an instance of the sequence submodular maximization problem over a directed graph $H(V, E')$ such that solving the latter problem also solves the former one. We assume all vertices and edges have a single state. Therefore, the problem translates to the non-adaptive sequence submodular scenario.

Graph H is built from graph G by replacing each edge $e = (u, v)$ in E by two directed edges (u, v) and (v, u). We define $h(S) = |S|$, which is linear and therefore submodular. Finally, the sequence

6. Note that in this section we define the approximation factor as the ratio of the the optimal solution to the solution provided by the algorithm.

submodular function f is defined as $f(\sigma) = h(E(\sigma)) = |E(\sigma)|$. It remains to show that for every subset of vertices S the value of function f for an arbitrary permutation σ_S of S is equivalent to the size of subgraph G_S induced by those vertices in graph G. This is true because for every edge $(u, v) \in G_S$ we have two corresponding edges in the directed graph H and based on the order of u and v exactly one of them is considered in $E(\sigma_S)$.

As a result, maximizing the function f with a cardinality constraint k is equivalent to solving the DkS problem. Thus, any algorithm with an α approximation factor to the sequence submodular maximization problem solves DkS with at least an α approximation factor. □

Manurangsi (2017) showed that any algorithm with an $O(n^{1/(\log\log n)^c})$ approximation factor to the DkS problem (for a constant $c > 0$) would prove the exponential time hypothesis is false. Next, we directly state the result of Manurangsi (2017).

Theorem F.13 (Manurangsi (2017), Theorem 1)**.** *There is a constant $c > 0$ such that, assuming the exponential time hypothesis, no polynomial-time algorithm can, given a graph G on n vertices and a positive integer $k \leq n$, distinguish between the following two cases:*

- *There exist k vertices of G that induce a k-clique.*
- *Every k-subgraph of G has density at most $n^{-1/(\log\log n)^c}$.*

To sum-up, Theorem 5.7 is proved from the combination of the two following facts:

1. If there is an algorithm with an approximation within a $n^{1/(\log\log n)^c}$ factor to the sequence submodular maximization problem, from the result of Lemma F.11, we know that it would solve the DkS problem with at most the same factor.
2. If there is an algorithm with a $n^{1/(\log\log n)^c}$ approximation factor to the DkS problem, it could distinguish the two cases of Theorem F.13 and would prove the exponential time hypothesis to be false.

F.3 Additional Experimental Details

F.3.1 Amazon Product Recommendation

In this application, we consider the task of recommending products to users. In particular, we use the Amazon Video Games review dataset (McAuley et al., 2015), which contains 10,672 products,

24,303 users, and 231,780 confirmed purchases. We furthered focused on the products that had been purchased at least 50 times each, leaving us with a total of 958 unique products.

Although we are using a different dataset, the experimental set-up closely follows that of the movie recommendation task in Tschiatschek et al. (2017) and Mitrovic et al. (2018a). We first group and sort all the data so that each user u has an associated sequence σ_u of products that they have purchased. These user sequences are then randomly partitioned into a training set and a testing set using a 80/20 split. Note that we 5 trials to average our results.

Using the training set, we build a graph $G = (V, E)$, where V is the set of all products and E is the set of edges between these products. Each product $i \in V$ has a self-loop (i, i), where the weight (denoted w_{ii}) is the fraction of users in the training set that purchased product v_i. Similarly, for each edge (i, j), the corresponding weight w_{ij} is defined to be the conditional probability of purchasing product j given that the user has previously purchased product i.

For each sequence σ_u in the test set, we are given the first g products that user u purchased, and then we want to predict the next k products that she will purchase. After each product is recommended to the user, the state of the product is revealed to be 1 if the user has indeed purchased that product, and 0 otherwise. At the start, the g given products are known to be in state 1, while the states of the remaining products are initially unknown.

As described in Section 5.2.2, the states of the edges are determined by the states of the nodes. In this case, the state of each edge (i, j) is equal to the state of product i. The intuitive idea is that edge (i, j) encodes the value of purchasing product j after already having purchased product i. Therefore, if the user has definitely purchased product i (i.e., product i is in state 1), then they should receive the full value of w_{ij}. On the other hand, if she has definitely not purchased product i (i.e., product i is in state 0), then edge (i, j) provides no value. Lastly, if the state of product i is unknown, then the expected gain of edge (i, j) is discounted by w_{ii}, the value of the self-loop on i, which can be viewed as a simple estimate for the probability of the user purchasing product i. See Figure 20a for a small example.

We use a probabilistic coverage utility function as our monotone adaptive submodular function h. Mathematically,

$$h(E_1) = \sum_{j \in V} \Big[1 - \prod_{(i,j) \in E_1} (1 - w_{ij})\Big],$$

where $E_1 \subseteq E$ is the subset of edges that are in state 1.

F.3.2 Wikipedia Link Prediction

We use the Wikispeedia dataset (West et al., 2009), which consists of 51,138 completed search paths on a condensed version of Wikipedia that contains 4,604 pages and 119,882 links between them. We further condense the dataset to include only articles that have been visited at least 100 times, leaving us with 619 unique pages and 7,399 completed search paths.

One natural idea for scoring each algorithm would be to look at the length of the shortest path between the predicted target and the true target. However, the problem with this metric is that all the popular pages have relatively short paths to most potential targets (primarily since they have so many available links to begin with). Hence, under this scoring, just choosing a popular page like "Earth" would be competitive with many more involved algorithms.

Instead, we define a measure we call the *Relevance Distance*. The relevance distance of a page i to a target page j is calculated by taking the average shortest path length to j across all neighboring pages of i. A lower distance indicates a higher relevance. For example, if our target page is *Computer Science*, both *Earth* $\rightarrow$ *Earth Science* $\rightarrow$ *Computer Science* and *University* $\rightarrow$ *Education* $\rightarrow$ *Computer Science* have a shortest path of length 2. However, the relevance distance of *Earth* to *Computer Science* is 2.68, while the relevance distance of *University* to *Computer Science* is 2.41, which fits better with the intuition that *University* is logically closer to *Computer Science*.

F.4 Deep Learning Baseline Details

F.4.1 Feed Forward Neural Network

For both experiments, the input to the Feed Forward Neural Network is a size $|V|$ vector X. That is, there is one input for each item in the ground set. In the Amazon product recommendation task in Section 5.2.4, $X_i = 1$ if the user is known to have purchased product i and 0 otherwise. Similarly, for the Wikipedia link prediction task in Section 5.2.5, $X_i = 1$ if the user is known to have visited page i and 0 otherwise.

The output in both cases is a size $|V|$ soft-maxed vector Y. In Section 5.2.4, Y_i can be viewed as the probability that product i will be the user's next purchase. In Section 5.2.5, Y_i can be viewed as the probability that user will visit page i next.

For the Amazon product recommendation task in Section 5.2.4, each user u in the training set has an associated sequence σ_u of products she purchased. Each such sequence was split into $|\sigma_u| - 2$

training points by taking the first g products as input and the $(g+1)$-th product as the output for $g = 1, \ldots, |\sigma_u| - 1$. For each user u in the testing set, we would take the first $g = 4$ products she purchased and encode them in the vector X as described above. We would then input this vector into our trained network and output the vector Y. In the non-adaptive case we cannot get any feedback from the user, so we simply output the products corresponding to the k highest values in Y.

In the adaptive case, we would look at the largest value Y_j in our output vector and output this as our first recommendation. We then check if the corresponding product appeared somewhere later in the user's sequence σ_u. If yes, then we would update our input X so that $X_j = 1$ and re-run the network to get our next recommendation. If not, we would simply use the next highest value in Y_j as our next recommendation (since the input doesn't change). This was repeated for k recommendations. This is supposed to mimic interaction with the user where we would recommend a product, and then see whether or not the user actually purchases this product. Note that we only considered values Y_j such that $X_j = 0$ because we did not want to recommend products that we knew the user had already purchased.

The main difference for the Wikipedia task in Section 5.2.5 is that, in the testing phase, we cannot simply output the top k values in Y as we did above because they likely will not constitute a valid path. Instead, we only have an adaptive version that is similar to what was described above. We find the highest value Y_j such that $X_j = 0$ (i.e. the user had not already been to this page) and a link to page j actually exists from our current page. We output this page j as our recommendation for the user's next page. We then check if the user actually visited our predicted page j at some point in their sequence of pages. If yes, we would update X so that $X_j = 1$ and re-run the network. If not we would look to the next highest value in the output Y. This was repeated for k guesses. Note that if we reached the true target page, we would stop making guesses.

In terms of architecture, we used a single hidden layer of 256 nodes with ReLU activations. We use a batch size of 1024 at first and then go down to a batch size of 32 when we are in the low data regime (i.e. only using 1% of the available training data). We used an 80/20 training/validation split to guide our early stopping criterion during training (with minimum improvement of 0.01 and patience of 1). We used categorical cross-entropy as our loss function.

F.4.2 LSTM

The main difference between the LSTM and the feed forward network is in the input. The input to the LSTM is a sequence of one-hot encoded vectors instead of just a single vector. That is, for the LSTM, each vector in the sequence had exactly one index with value 1.

We experimented with using a long sequence of input vectors and padding with all-zero vectors, but we found better results using a fixed small sequence length g and then "pushing" the sequence back when updating. For example, if our current input was a sequence of vectors $[v_1, v_2, v_3]$ and we wanted to update it with a new vector v_4, the updated input would be $[v_2, v_3, v_4]$.

The adaptive LSTM followed the same set-up as the non-adaptive LSTM, but with the same adaptive update rules described above for the feed-forward neural network.

For all experiments, we used a single hidden layer of 8 LSTM nodes. The other hyperparameters are all the same as described for the Feed Forward network above, except we start at a batch size of 256 instead of 1024 (before also going down to a batch size of 32 in the low data regime).

It is worth noting that the datasets that we used in our experiments do not contain any explicit features about the users or the items, which is why we simply encoded sets/sequences of items as 0-1 vectors in our deep-learning baselines.