An Efficient Framework for Balancing Submodularity and Cost

Sofia Maria Nikolakaki
Boston University
Boston, USA
smnikol@bu.edu

Alina Ene
Boston University
Boston, USA
aene@bu.edu

Evimaria Terzi
Boston University
Boston, USA
evimaria@bu.edu

ABSTRACT

In the classical *selection problem*, the input consists of a collection of elements and the goal is to pick a subset of elements from the collection such that some objective function f is maximized. This problem has been studied extensively in the data-mining community and it has multiple applications including influence maximization in social networks, team formation and recommender systems. A particularly popular formulation that captures the needs of many such applications is one where the objective function f is a monotone and non-negative submodular function. In these cases, the corresponding computational problem can be solved using a simple greedy $(1 - \frac{1}{e})$-approximation algorithm.

In this paper, we consider a generalization of the above formulation where the goal is to optimize a function that maximizes the submodular function f minus a linear cost function c. This formulation appears as a more natural one, particularly when one needs to strike a balance between the value of the objective function and the cost being paid in order to pick the selected elements. We address variants of this problem both in an offline setting, where the collection is known apriori, as well as in online settings, where the elements of the collection arrive in an online fashion. We demonstrate that by using simple variants of the standard greedy algorithm (used for submodular optimization) we can design algorithms that have provable approximation guarantees, are extremely efficient and work very well in practice.

CCS CONCEPTS

• **Theory of computation → Approximation algorithms analysis**.

KEYWORDS

team formation, influence maximization, recommender systems, approximation algorithms

ACM Reference Format:
Sofia Maria Nikolakaki, Alina Ene, and Evimaria Terzi. 2021. An Efficient Framework for Balancing Submodularity and Cost. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467367

1 INTRODUCTION

The *element selection* problem is central in the data-mining community and it essentially seeks to pick a set of elements from a collection so that some objective is optimized. The applications of such a general formulation abound. The most relevant to this work are those related to *influence maximization* in social networks [20, 24, 25], *team formation* [1, 18, 23] and *recommender systems* [8, 10, 19]. For example, in influence maximization the goal is to pick a subset of the nodes of the network so that once they adopt an item (e.g., idea or product) the spread of this item in the network is maximized. Similarly, in team formation, given a collection of experts the goal is to pick a subset of them such that they cover the skills required for a task and also optimize a social objective. Finally, in recommender systems the goal is to pick a subset of items from a collection (e.g., restaurants or movies) such that the selected items best summarize the collection or best match the users' interests.

In many of the above examples the problem is formulated as a submodular-optimization problem where the goal is to pick a subset of k elements Q from a collection V such that $f(Q)$ is maximized, where $f(Q)$ is a monotone and non-negative submodular function. Subsequently, an easy-to-implement and practical greedy algorithm is used to solve such problems and provide a solution with approximation guarantee $(1 - \frac{1}{e})$.

Conceptual contributions: In this paper, we consider a generalization of this framework, where the goal is again to pick a set of elements Q from an input collection V. However, our goal is to not only maximize the function $f(Q)$, but also to strike a balance between the benefits of choosing Q, as quantified by $f(Q)$, and the cost of picking Q, denoted as $c(Q)$. Therefore, our goal is to find $Q \subseteq V$ to maximize the combined function:

$$g(Q) = f(Q) - c(Q), \tag{1}$$

where $f(Q)$ is the monotone and non-negative submodular function and $c(Q)$ is the *sum* of the costs of the elements in the solution, i.e., a non-negative linear function.

In the case of influence maximization, the goal is to optimize the expected spread of a product or an idea for a seed of nodes Q minus the cost of picking such nodes. Similarly, in the team-formation scenario, this would mean that we want to optimize the coverage of the task skills that the experts in Q cover minus the cost of hiring these experts. Finally, in recommender systems the goal could be to maximize the diversity between proposed movies minus the cost of their distance from a particular year of being produced.

In order to capture the demands of such application domains we consider two variants of the general problem outlined in Equation (1): the *constrained* and the *unconstrained.* The former refers to cases where the maximum number of elements we aim to pick is given as part of the input; the latter finds the optimal number of elements to be picked as part of the solution.

Although for the constrained version we only discuss the cardinality constraint, our methods extend to handle general matroid constraints as well. While we leave this discussion for an extended version of this manuscript, we note that this version is relevant to scenarios where the elements in the collection are partitioned into groups; then, a matroid constraint would impose an upper bound on the number of elements that can be selected from every group.

Finally, we also consider the online version of the above problems. In this setting the elements in the collection become available in an online fashion. The constant addition of data in online platforms makes this setting increasingly relevant.

Algorithmic contributions: The structure of the combined objective in Equation (1) in the offline setting and under the cardinality constraint has been addressed by previous works [14, 17] as well. What distinguishes us from these works is that we intentionally design algorithms with slightly weaker theoretical approximation guarantees that allow the use of runtime acceleration techniques, such as lazy greedy evaluations. As a result, we experimentally show that our proposed algorithms can achieve significant speedups compared to previous approaches in the constrained and unconstrained settings, respectively, while we also show that in practice they perform equally well. In addition, we propose algorithms with provable guarantees for the online and streaming settings as well as algorithms for more general constraints such as a matroid constraint.

Experimental results: We experimentally evaluate our algorithms on real datasets from a variety of domains such as social networks, crowdsourcing platforms and recommender systems. For our experiments we use different instances of the submodular function f and the cost function c – chosen appropriately for the specific application domain we experiment with. Our experiments show that our algorithms obtain solutions with quality at least as good as existing algorithms while being significantly faster.

2 RELATED WORK

In this section, we highlight the relationship between our work and research done in application and theoretical domains.

Influence maximization: The seminal work of Kempe *et al.* [20] ignited a lot of subsequent research on influence maximization on social graphs [25]. In all of these works, there is an underlying information propagation model and a social network that captures the degree of influence that every node has on others. The goal is to identify k nodes in the network to maximize the expected spread of any item (e.g., idea or product) that these nodes adopt. Based on this general idea, there has been a set of followup works [7, 9, 24, 26]. All these works assume some diffusion model such that the expected spread is a submodular function and therefore a greedy algorithm can be deployed to maximize it. Our work generalizes all these works as we want to maximize the expected spread minus the cost for convincing these nodes to adopt the particular item.

Team formation: The classic team-formation problem [1, 2, 6, 18, 23, 35] assumes that there is a pool of experts and a subset of them is selected to cover the requirements of a task, while some criteria related to the team functionality (e.g., communication cost as captured in their collaboration network) are optimized. At the heart of all team-formation problems defined today is a *set cover* problem where the goal is to cover the skills of the input task. In our formulation we consider an extension of this setting where not all skills need to be covered and we seek to strike a balance between the covered skills and the cost of building a team.

Recommender systems: Recommendations in recommender systems have often been formulated as a submodular function optimization problem, with the goal being to maximize coverage (e.g., of product attributes being addressed by reviews), while maximizing the diversity of the items being recommended [3, 8, 32], or improving the summarization of a set of items (e.g., restaurants available) [10, 28]. In all of the above cases, the goal has been to maximize a submodular function and not the combined function of the benefit minus the cost associated with these recommendations. Kazemi *et al.* [19] model recommender systems as maximizing benefit minus cost, and we consider their applications in our experimental evaluation. We discuss these applications in Section 3.

Maximizing submodularity minus cost: Several algorithms have been developed for maximizing both monotone and general submodular functions, but they achieve provable approximation guarantees only for non-negative functions, whereas the objective in Equation (1) is potentially negative. Existing hardness results imply that no multiplicative approximation guarantees are possible in polynomial time for maximizing a potentially negative submodular function with or without constraints [13, 31].[1] Nevertheless, the objective function we consider has some structure that has been exploited in previous works [14, 17, 34]. These works have shown that in this case we should aim for a weaker notion of approximation and find a solution Q satisfying

$$f(Q) - c(Q) \geq \alpha \cdot f(\mathrm{OPT}) - c(\mathrm{OPT}),$$

for some $\alpha \leq 1$. The aforementioned works propose algorithms that achieve $\alpha = (1 - 1/e)$ in the offline setting, which is the best guarantee achievable in polynomial time for a cardinality constraint [13, 29]. One of the main downsides of these algorithms is that the running time can be prohibitive. The works [14, 34] propose algorithms based on the continuous greedy algorithm that maximizes the multilinear extension, a continuous function extending the submodular function to the domain $[0, 1]^n$. The multilinear extension is expensive to evaluate and the continuous greedy algorithm requires many iterations to converge. As a result, the algorithms have very high running times. The algorithm of [14] is based on the standard discrete greedy algorithm, which is much more efficient, but it applies the greedy approach to a distorted objective that changes throughout the algorithm and we cannot use techniques such as lazy evaluations to speed up the algorithm. Thus the running time of the algorithm of [14] is $\Theta(n^2)$, where n is the size of the ground set, whereas the standard greedy algorithm can be implemented to run in nearly-linear time using approximate lazy evaluations. Moreover, the implementation of the standard greedy algorithm using exact lazy evaluations achieves significant

[1]One can observe that it is **NP**-hard to decide whether the optimum value of a submodular objective is positive or not, since we could use such a subroutine and binary search over the optimum value to obtain arbitrarily good approximate solutions, which contradicts existing hardness of approximation results for problems such as maximum cut and maximum coverage [13, 31].

speedups in practice without affecting the approximation guarantee [27]. Additionally, these algorithms are only for the offline problem.

In this paper, we give a novel approach that overcomes these limitations. Our main insight is very simple but very effective: instead of maximizing the original objective $f(Q) - c(Q)$, we maximize a *scaled* objective $f(Q) - s \cdot c(Q)$, where $s > 1$ is an absolute constant. Unlike the approach of [14], our objective does not change throughout the algorithm and thus we can use lazy evaluations. Moreover, we can leverage a wide-range of existing algorithmic approaches, such as the standard greedy algorithm in the offline setting and variants of greedy in the online setting. As a result, we obtain faster offline algorithms for the cardinality-constrained problem, algorithms for the online and streaming settings, and algorithms for more general constraints such as a matroid constraint.

Our problem formulation can be viewed as a Lagrangian relaxation of the problem of maximizing a submodular function subject to a knapsack constraint. Several algorithms have been proposed for the latter problem, including algorithms achieving the optimal $1 - 1/e$ approximation guarantee [33]. However, these algorithms have very high running times and they are primarily of theoretical interest. For example, the algorithm of [33] has running time $\Theta(n^5)$, where n is the size of the ground set. Thus, even if we used lazy evaluations to speed up the algorithm, the enumeration will still be a significant bottleneck. Obtaining fast and practical algorithms for the knapsack problem remains an outstanding open problem (see e.g. [11] and references therein). Thus, our problem formulation also comes with significant algorithmic benefits compared to the formulation with a hard budget constraint.

In contemporaneous work, Kazemi *et al.* [19] develop streaming and distributed algorithms for the cardinality-constrained problem. The streaming algorithms developed in their work and ours are conceptually very similar and they achieve the same approximation guarantee.

3 PROBLEM DEFINITION

Throughout the paper we will assume a set of n elements $V = \{1, \ldots, n\}$. We also assume two functions: $f : 2^V \to \mathbb{R}$ and $c : 2^V \to \mathbb{R}$, such that f is a monotone and non-negative submodular function and c is a non-negative linear function.

Recall that a set function $h : 2^V \to \mathbb{R}$ is *monotone* if

$$h(S) \leq h(T) \quad \forall S \subseteq T \subseteq V$$

The set function $h : 2^V \to \mathbb{R}$ is *submodular* if it satisfies the following diminishing returns property:

$$h(T \cup \{u\}) - h(T) \leq h(S \cup \{u\}) - h(S) \quad \forall S \subseteq T, u \in V \setminus T$$

An equivalent definition of submodularity is the following:

$$h(S) + h(T) \geq h(S \cap T) + h(S \cup T) \quad \forall S, T \subseteq V$$

Given the above, we define our *objective function* $g : 2^V \to \mathbb{R}$ as follows:

$$g(Q) = \lambda \cdot f(Q) - c(Q). \tag{2}$$

Note that function g is also submodular but it can take both negative and positive values and it is not monotone. The problems we aim to solve in this paper are related to optimizing this function and can be defined as follows.

Problem 1 (k-Constrained). *Given a set of elements V and an integer k, find $Q \subseteq V$ such that $|Q| \leq k$ and*

$$g(Q) = \lambda \cdot f(Q) - c(Q) \tag{3}$$

is maximized.

Our approach extends to a general matroid constraint. The algorithm we discuss in Section 4 can be minimally modified to work for general matroid constraints, including its running time and approximation bounds. Due to space constraints, we defer this result to the extended version of our paper.

We also consider the unconstrained version of the above problem, which is defined as follows.

Problem 2 (Unconstrained). *Given a set of elements V find $Q \subseteq V$ such that*

$$g(Q) = \lambda \cdot f(Q) - c(Q) \tag{4}$$

is maximized.

Online problems: In addition to the offline setting, we study the above problems in online and streaming models of computation. We consider Problem 2 in the online model where the experts arrive in an online fashion, one at a time, in an arbitrary (adversarial) order. When an expert arrives, we need to decide whether to add it to the solution, and this decision is irrevocable. We refer to this problem as Online-Unconstrained.

We also consider Problem 1 in the streaming model where the experts arrive one at a time as in the online setting but we are allowed to store a small set of experts in memory and select the final solution from this set. We refer to this problem as Streaming-k-Constrained.

The normalization coefficient λ: In the above definitions, λ is a normalization coefficient that encodes our bias between the prizes and the costs. One can also think of λ as a way to convert the two quantities into the same units. Determining its value is application-dependent and is discussed in Section 7.2. Our algorithmic analysis is independent of this coefficient, and therefore from now on we will use $f(Q)$ to refer to $\lambda \cdot f(Q)$. We will also refer to $g(Q) = f(Q) - c(Q)$ as the *combined objective function.*

Approximation guarantees: Note that while function f is monotone submodular and non-negative, the combined objective function g is a *potentially negative* submodular function. As discussed in the introduction, no multiplicative factor approximation is possible for the problem of maximizing a submodular function that is potentially negative. Similarly to previous work (see Section 2), our algorithms construct solutions with the following kind of weaker approximation guarantees:

$$f(Q) - c(Q) \geq \alpha\, f(\mathrm{OPT}) - c(\mathrm{OPT}),$$

where OPT is an optimal solution to the problem and $\alpha \leq 1$.

Problem instances: In our experimental evaluation, we consider several instantiations of the monotone submodular function f and the linear function c, arising in influence maximization in social networks, team formation, and recommender systems.

Influence maximization in social networks: The ground set V corresponds to social network nodes and the goal is to pick a subset of the nodes $Q \subseteq V$ such that the spread of a product (or an idea) in the network is maximized. The function $f(Q)$ corresponds to the expected number of people that adopt the product given seed set Q. The way the expectation is computed depends on the *information propagation model* being used. In this paper, we focus on the independent cascade and linear-threshold models [20] which makes $f(Q)$ submodular. We also use a non-negative linear function $c(Q)$ to quantify the sum of the costs of convincing each node to adopt a product. Assuming that influential nodes are more expensive to convince, in our experiments, we model the cost for convincing each individual as being proportional to the node's degree.

Recommender systems: Kazemi *et al.* [19] consider several applications to recommender systems and show that they can be modeled as instances of the problem k-Constrained. In our experimental evaluation, we use the following two problem instances proposed by them, which we describe here for completeness. In both applications, the ground set V corresponds to items — e.g., restaurants or movies — and each item i is associated with a set of features which are then used to compute the distance between two items $d(i, j)$. A similarity matrix $\mathbf{M}$ between items is formed by setting $\mathbf{M}(i, j) = e^{-d(i,j)}$.

The first application considers restaurant recommendations. The items are restaurants. The submodular function f is defined as:

$$f(Q) = \sum_{i=1}^{n} \max_{j \in Q} \mathbf{M}(i, j). \tag{5}$$

For the linear cost function $c(Q) = \sum_{i \in Q} c_i$, the cost c_i corresponds to the distance of restaurant i to the center of the city.

The second application considers movie recommendations. The items are movies. The submodular function f is defined as:

$$f(Q) = \log \det(\mathbf{I} + \alpha \mathbf{M}_Q), \tag{6}$$

where $\mathbf{M}_Q$ is the principal submatrix of $\mathbf{M}$ indexed by Q, $\mathbf{I}$ is the identity matrix and α is a positive scalar. Informally, this objective aims to diversify the vectors in Q. For the linear cost function $c(Q) = \sum_{i \in Q} c_i$, the costs are given by $c_i = 10 - \text{rating}_i$, where rating_i denotes the average rating that movie i has received.

Team formation: The ground set V is a set of experts and each expert i is associated with a set of skills $S_i \subseteq S$, where S is a universe of skills. Given a task $T \subseteq S$ and a set of experts $Q \subseteq V$ we define the *coverage* function to be the number of skills in T that is covered by at least one expert in Q. Thus

$$f(Q) = \left| \left(\cup_{i \in Q} S_i \right) \cap T \right|. \tag{7}$$

Each expert i is also associated with a *cost* c_i needed to hire the expert. The cost of hiring a team is the sum of the expert costs, i.e., $c(Q) = \sum_{i \in Q} c_i$.

4 THE COST-SCALED GREEDY ALGORITHM

In this section, we consider the cardinality-constrained problem k-Constrained. Our algorithm for this problem is shown in Algorithm 1.[2] Throughout the paper, by elements we mean the elements of the ground set V. For a set function h, we use the notation $h(e|Q) := h(Q \cup \{e\}) - h(Q)$ to denote the marginal gain of e on top of Q. Our approach is very simple but effective: we apply the standard Greedy algorithm to the scaled objective $\tilde{g}(Q) = f(Q) - 2c(Q)$, and we stop adding elements once the marginal gains become negative

Algorithm 1 The CSG algorithm for the cardinality-constrained problem k-Constrained.

Input: Ground set V, scaled objective $\tilde{g}(Q) = f(Q) - 2c(Q)$, cardinality k.
Output: Team Q.

```
1: $Q \leftarrow \emptyset$
2: for $i = 1, \dots, k$ do
3:     $e_i = \arg\max_{e \in V} \tilde{g}(e|Q)$
4:     if $\tilde{g}(e_i|Q) \leq 0$ then
5:         break
6:     end if
7:     $Q \leftarrow Q \cup \{e_i\}$
8: end for
9: return $Q$
```

(line 5). As we discuss below, the algorithm can be implemented using lazy evaluations, which leads to a very efficient and practical algorithm. In Appendix A, we show the following guarantee:

Theorem 4.1. *Algorithm 1 returns a solution Q of size at most k satisfying $f(Q) - c(Q) \geq \frac{1}{2} f(\mathrm{OPT}) - c(\mathrm{OPT})$.*

Running time and speedups: Similarly to the standard Greedy algorithm, the running time of CSG is $O(nk)$ evaluations of the functions f and c, where n is the number of experts and k is the cardinality constraint: there are k iterations and, in each iteration, we spend $O(n)$ function evaluations to compute all of the marginal gains and find the expert with maximum marginal gain.

The computational bottleneck of the algorithm is in finding the element with maximum marginal gain $\tilde{g}(e|Q)$ in every iteration. To speed up these computations and avoid unnecessary evaluations, we deploy the lazy evaluations technique introduced by Minoux [27] for the standard Greedy algorithm. That is, we store each element in a maximum priority queue with a key $v(e)$. We initialize the keys to $v(e) = \tilde{g}(e|\emptyset)$. The keys are storing potentially outdaded marginal gains and the algorithm updates them in a lazy fashion. Since $\tilde{g}$ is submodular, marginal gains can only decrease as the solution Q grows and, as a result, the keys are always an upper bound on the corresponding marginal gains. In each iteration, the algorithm uses the queue to find the element with maximum marginal gain as follows. We remove from the queue the element e with maximum key and evaluate its marginal gain $\tilde{g}(e|Q)$ with respect to the current solution Q. We then compare the marginal gain $\tilde{g}(e|Q)$ to the key $v(e')$ of the element e' that is now at the top of the queue (before removing e from the queue, e' was the element with the second-largest key). If $\tilde{g}(e|Q) \geq v(e')$, then e is the element with largest marginal gain, since the key of every element is an upper bound on its current marginal gain. Otherwise, we reinsert e into the queue with key $\tilde{g}(e|Q)$ and repeat.

We use CSLG to refer to the implementation of CSG with lazy evaluations. The correctness of CSLG follows directly from submodularity, and the solution constructed is the same as that of CSG. While the worst-case running time of CSLG and CSG are the same, lazy evaluations lead to significant speedups in practice.

We note that there is also an approximate version of the lazy evaluations technique that allows us to obtain worst-case running time

[2]The algorithm and its analysis extend to a general matroid constraint. We defer this result to an extended version of this paper.

Algorithm 2 The `Online-CSG` algorithm.

Input: Stream of elements V, scaled objective $\tilde{g} = f - 2c$.
Output: Team Q.

```
1: $Q \leftarrow \emptyset$
2: for each arriving element $e$ do
3:     if $\tilde{g}(e|Q) > 0$ then
4:         $Q \leftarrow Q \cup \{e\}$
5:     end if
6: end for
7: return $Q$
```

Algorithm 3 The `Streaming-CSG` algorithm.

Input: Stream of elements V, scaled objective $\tilde{g} = f - s \cdot c$ ($s \geq 1$ is an absolute constant), cardinality k, threshold τ.
Output: Team Q.

```
1: $Q \leftarrow \emptyset$
2: while stream not empty do
3:     $e$ ←next stream element
4:     if $\tilde{g}(e|Q) \geq \tau$ and $|Q| < k$ then
5:         $Q \leftarrow Q \cup \{e\}$
6:     end if
7: end while
8: return $Q$
```

that is nearly-linear at a small loss in the approximation guarantee [5]. We do not consider this variant in this paper.

5 THE `ONLINE` ALGORITHM

We now turn our attention to the Unconstrained problem in the *online* model where the elements arrive one at a time. When an element arrives, we need to decide whether to add it to the solution, and this decision is irrevocable. Algorithm 2 considers the scaled objective $\tilde{g}(Q) = f(Q) - 2c(Q)$ and it accepts every element that has positive marginal gain with respect to this scaled objective. The following theorem states our approximation guarantee. We defer the proof to an extended version of this paper.

Theorem 5.1. *Algorithm 2 returns a solution Q satisfying $f(Q) - c(Q) \geq \frac{1}{2} f(\mathrm{OPT}) - c(\mathrm{OPT})$.*

6 THE `STREAMING` ALGORITHM

This section considers the k-Constrained problem in the streaming model. The algorithm is an extension of the online algorithm from Section 5. As before, we consider the scaled objective $\tilde{g}(Q) = f(Q) - s \cdot c(Q)$, where $s \geq 1$ is an absolute constant (the right choice for s is no longer 2, see Theorem 6.1 below). Now, instead of picking elements whose scaled marginal gain is positive, we pick elements whose scaled marginal gain is above a suitable threshold. In other words, we apply the single-threshold Greedy algorithm [4, 22] to the scaled objective. The resulting algorithm is shown in Algorithm 3. The following theorem shows that there is a way to set τ and s so that Algorithm 3 returns a good approximate solution, and we give its proof in Appendix B.

Theorem 6.1. *When run with scaling constant $s = \frac{1}{2}\left(3 + \sqrt{5}\right)$ and threshold $\tau = \frac{1}{k}\left(\frac{1}{2}(3 - \sqrt{5}) f(\mathrm{OPT}) - c(\mathrm{OPT})\right)$, Algorithm 3 returns a solution Q such that $|Q| \leq k$ and*

$$f(Q) - c(Q) \geq \frac{1}{2}\left(3 - \sqrt{5}\right) f(\mathrm{OPT}) - c(\mathrm{OPT})$$

Setting the threshold as suggested by the above theorem requires knowing $\hat{g}(\mathrm{OPT})$, where $\hat{g}(Q) := \frac{1}{2}(3 - \sqrt{5}) f(Q) - c(Q)$. To remove this assumption, we use the standard technique introduced by [4] and run several copies of the basic algorithm in parallel with different guesses for $\hat{g}(\mathrm{OPT})$. We only lose ϵ in the approximation due to guessing and we use $O(k \log k/\epsilon)$ total space to store the $O(\log k/\epsilon)$ solutions.

Theorem 6.2. *There is a streaming algorithm* `Streaming-CSG` *for the cardinality-constrained problem $\max_{|Q| \leq k} f(Q) - c(Q)$ that takes as input any $\epsilon > 0$ and it returns a solution Q satisfying*

$$f(Q) - c(Q) \geq \left(\frac{1}{2}\left(3 - \sqrt{5}\right) - \epsilon\right) f(\mathrm{OPT}) - c(\mathrm{OPT})$$

The algorithm uses $O(k \log k/\epsilon)$ space.

The result presented in this section was obtained independently and concurrently by [19] and a preliminary version of our work [12].

7 EXPERIMENTS

In this section, we experimentally evaluate our algorithms on real-world datasets.

7.1 Algorithms of the comparative study

We compare our algorithms to a variety of baseline methods, as well as to the state-of-the-art algorithms.

Algorithms used for the k-Constrained problem: We experimentally evaluate our main algorithms for the k-Constrained problem: the cost-scaled Greedy algorithm (`CSG`) and its variant with lazy evaluations (`CSLG`) (see Section 4). Recall that the two algorithms return the same solution, but `CSLG` is expected to be significantly faster due to the lazy evaluations. Thus, in the objective evaluation plots we only include `CSLG`. However, in the running-time evaluation plots we present both algorithms to demonstrate the speedups achieved due to lazy evaluations.

We also evaluate our streaming algorithm, `Streaming-CSG`, described in Section 6. The algorithm is designed for the more challenging setting where the elements arrive in a stream, one at a time, and the algorithm can make only one pass over the elements and store only a small number of elements in memory. We evaluate the performance of `Streaming-CSG` against offline algorithms that have complete knowledge of the input datasets and can make many passes over the elements.

The baselines we consider are algorithms proposed in prior work as well as some intuitive heuristics. We list them below:

- `DistortedGreedy` *[17]*: Builds on the Greedy approach but, instead of considering a constant scaled objective like we do, the authors design a *distorted objective* which changes throughout the algorithm. The distorted objective initially places higher relative importance on the modular cost term c, and gradually increases the relative importance of the coverage function as the algorithm progresses. DistortedGreedy makes O(nk) evaluations and returns a solution Q of size at most k satisfying $f(Q) - c(Q) \geq (1 - \frac{1}{e})f(\text{OPT}) - c(\text{OPT})$.

- StochasticDistortedGreedy *[17]*: Uses the same distorted objective as DistortedGreedy but has faster asymptotic runtime because it optimizes over a random sample in each iteration.
- Greedy: This is the greedy algorithm for maximizing submodular set functions [30]. The only difference from CSG is that instead of computing the marginal value with respect to the scaled objective we use the original objective $g = f - c$.
- TopK: This is a natural heuristic baseline algorithm that runs as follows. The algorithm gives each element $e \in V$ a linear weight $w(e) = f(\{e\}) - c(e)$ and it selects the (at most) k elements with the largest positive weights. If there are fewer than k elements with positive weight, the algorithm selects all of the elements with positive weights; otherwise, the algorithm selects the k elements with largest weights.

Algorithms used for the UNCONSTRAINED problem: For the UNCONSTRAINED problem we evaluate all the methods that we used for the k-CONSTRAINED problem by setting $k = n$. So all algorithms described above are included in the comparison.

Additionally, we evaluate our online algorithm, Online-CSG, described in Section 5. This algorithm addresses the harder online problem where the elements are presented in an online fashion, and the algorithm needs to irrevocably decide whether to include the element in the solution when the element arrives. We evaluate the algorithm's performance against offline algorithms that have complete knowledge of the input datasets.

In terms of baselines, we also consider the following:

- UnconstrainedDistortedGreedy [17]: A linear-time algorithm for the unconstrained problem that runs for n iterations and in each iteration evaluates the marginal gain of a single element sampled uniformly at random.

7.2 Experimental setup

For all our experiments we evaluate the algorithmic performances on different subsets of the original data and we report the average performance value of each algorithm over these 15 samples, denoted as its line, as well as the confidence interval of the result, denoted as the bar around the line. Our code is in Python and for all our experiments we use single-process implementations on a 64-bit MacBook Pro with an Intel Core i7 CPU at 2.6GHz and 16 GB RAM. Our experiments assume hyperparameters, whose selection process will be described in an extended version of this manuscript. For replication purposes we make the code, the datasets and the chosen hyperparameters available online.[3]

Selecting the value of the normalization coefficient λ: The combined objective introduced in Section 3 compares the gain and the cost; the gain corresponds to the value of a submodular function and the cost is the value of a linear function. The purpose of the parameter λ is to transform these two quantities into comparable units and we set it as follows. First, we use the well-known greedy algorithm [30] to find the set of elements Q^* that maximize the submodular function. Then, we define $\lambda = \beta\frac{c(Q^*)}{f(Q^*)}$, with $\beta \in \{2, 4\}$ depending on the dataset such that the gain and the cost are in a comparable scale.

Setting the algorithmic parameter ϵ: Recall that algorithms StochasticDistortedGreedy and Streaming-CSG require an error parameter ϵ as part of their input. This is a trade-off parameter between the quality of the solution and the algorithm's running time. To select an appropriate value ϵ we performed a set of experiments for different ϵ values and picked the one that achieves the best (for the algorithm) solution, without sacrificing the running time. Due to lack of space we omit these plots. Throughout the experiments we fix ϵ=0.01 and ϵ=0.05 for StochasticDistortedGreedy and Streaming-CSG respectively.

Applications and datasets: We experimentally evaluate the proposed algorithms on datasets from application domains we discussed in Section 3.

Influence Maximization: In these experiments, we follow the experimental setup of [9, 20]. We use the academic collaboration network from the "High Energy Physics-Theory" section of the e-print arXiv that was used in these prior works. We consider a network which contains 1077 nodes and 3505 edges (one of the largest components of the whole dataset) and we refer to this as the *NetHEPT* dataset. In our experiments, we use the independent cascade model, but similar results hold for the linear threshold model as well. We treat the multiplicity of edges as weights and similar to the experiments of [9, 20] we assign a uniform probability of $p = 0.01$ to each edge. The specific instantiations of functions f and c in this application are the ones described in Section 3.

Team Formation: In these experiments, we follow the experimental setup of [2, 15]. We use a real-world dataset from the online expertise-management platform guru.com that henceforth we refer to as *Guru*. All expert-related data used in this work are obtained from anonymized profiles of members registered in the marketplace. Our dataset has 6120 experts and 20 tasks, each requiring 15 skills. For each expert we also know the set of skills the expert has. Since we consider multiple tasks, we evaluate our algorithms for each task separately and report the average performance value of each algorithm over all tasks as well as the confidence interval of the results. The instantiations of functions f and c in this application are the ones described in Section 3 (see Eq. (7)).

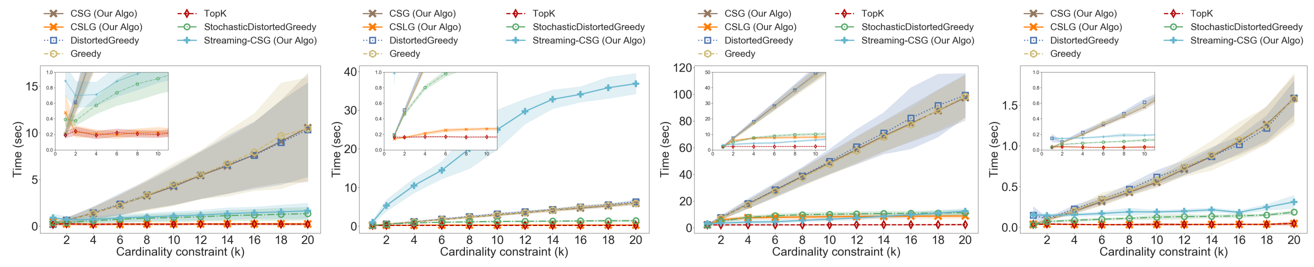
Recommender Systems: For the recommender systems application we consider two separate applications from the work of [19]: (i) restaurant summarization, and (ii) movie recommendation. We follow the experimental setup of [19].

For the restaurant summarization task we use a subset of restaurant businesses obtained from the Yelp Academic dataset.[4] We use features that cover a range of restaurant attributes.[5] We focus on the restaurants from the metropolitan area of Las Vegas and the goal is to identify representative restaurants in that area. To evaluate our algorithms we consider random restaurant subsets of size 651 (5% of the data) and report the average performance of our algorithms over all subsets as well as their confidence interval. We

[3]https://www.dropbox.com/sh/vu87zte0p4hrybz/AACs0liWgCejxj5R9FEowhRza?dl=0

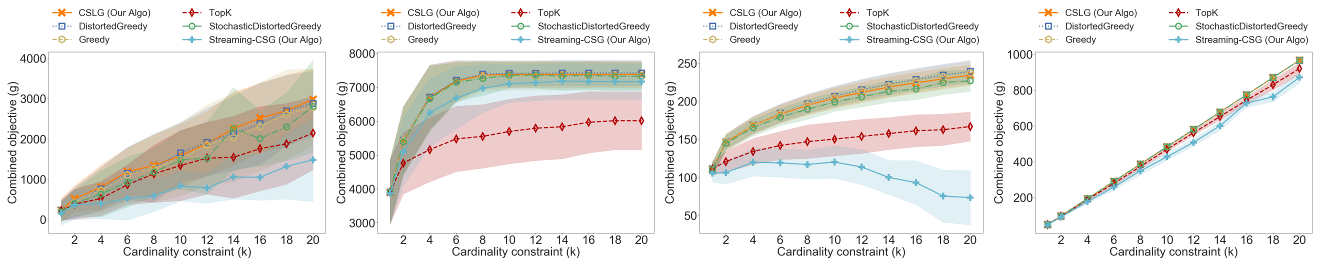
[4]https://www.yelp.com/dataset

[5]To extract the features of each restaurant we use the script provided at https://github.com/vc1492a/Yelp-Challenge-Dataset.



(a) ***NetHEPT*** (b) ***Guru*** (c) ***Yelp*** (d) ***Movielens***

Figure 1: Running time (sec) comparisons of all algorithms for the k-Constrained problem.



(a) ***NetHEPT*** (b) ***Guru*** (c) ***Yelp*** (d) ***Movielens***

Figure 2: Combined objective value (g) comparisons of all algorithms for the k-Constrained problem.

refer to this dataset as *Yelp*. The instantiations of functions f and c in this application are the ones described in Section 3 (see Eq. (5)).

For the movie recommendation task we use the MovieLens [16] dataset. In this dataset we are given user ratings for different movies. To extract the feature vector for each movie we use gradient descent [21]. At the end of this process we obtain 40 latent factors for each movie. After filtering out movies with less than 50 ratings , we consider random subsets of 658 movies to evaluate our algorithms over different inputs. We refer to this dataset as *Movielens*. The instantiations of functions f and c in this application are the ones described in Section 3 (see Eq. (6)).

7.3 Evaluation for k-Constrained

Here we evaluate the running time and empirical performance of the algorithms for the k-Constrained problem. In summary, we demonstrate that using our algorithm `CSLG` can lead to faster running time without sacrificing the quality of the solution.

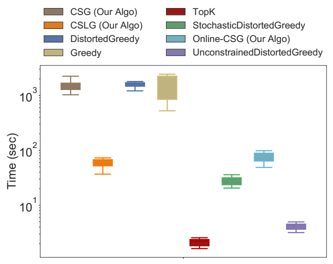
Runtime analysis: We start by evaluating the scalability of our methods. We vary the cardinality parameter k and compute the running time of each algorithm. The results of their running time performance are shown in Figure 1. The y-axis represents the running time (in sec), and the x-axis represents the cardinality k.

We note that `DistortedGreedy` and `CSG` require the most time to run and have very close performances. Next, we consider the `StochasticDistortedGreedy` algorithm, which is faster than the aforementioned algorithms because in each iteration it only evaluates the marginal gain of a subset of the elements. We now draw the attention to the computational savings when using our proposed cost scaled greedy with lazy evaluations. In all datasets and specifically for larger values of k, a reasonable setting in all of our applications, `CSLG` is more than 100x faster than both `DistortedGreedy` and `CSG`, and 10x faster than `StochasticDistortedGreedy`. The only algorithm whose running time is comparable to `CSLG` is `TopK`, but the latter algorithm achieves lower objective value.

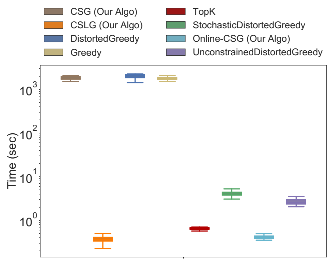
Performance evaluation: Here we show that the aforementioned computational gains are achieved without sacrificing the solution quality. For the performance evaluation we vary the cardinality parameter k and compute the combined objective function (g) of the obtained solution. We present the results in Figure 2.

We observe that the performance trends of the algorithms are overall consistent between all datasets and applications. Note that as k increases so does the objective value of the solutions found by the algorithms. For the case of *Guru* and *Yelp* we notice that the performance of the algorithms increases up until some point where it seems to stabilize. A possible explanation is that initially the algorithms benefit from adding more elements to the solution because increasing the submodular gain outweighs the cost. However, for some cardinality k the algorithms may reach a solution where adding more elements does not benefit them. This happens in two cases; (i) when we have reached the maximum possible submodular value (e.g. covering all the requirements of a task), and (ii) when the benefit from increasing the submodular value is smaller than paying the corresponding cost. We see that this is less pronounced in *NetHEPT* and is not observed at all in *Movielens*.

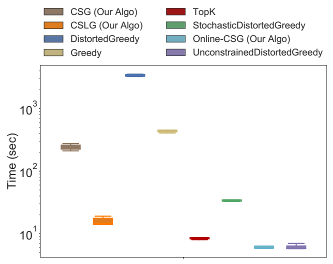
Comparison across algorithms reveals that the baseline `TopK` and `Streaming-CSG` have the worse performance. Intuitively, the latter has a lower performance because it is an online algorithm and



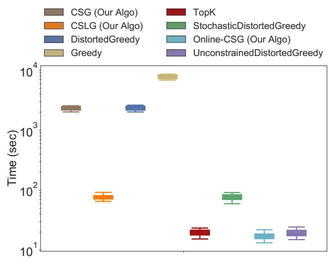
(a) ***NetHEPT***



(b) ***Guru***

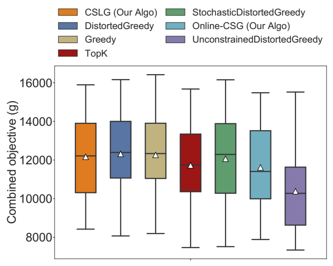


(c) ***Yelp***

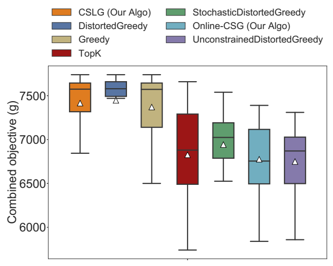


(d) ***Movielens***

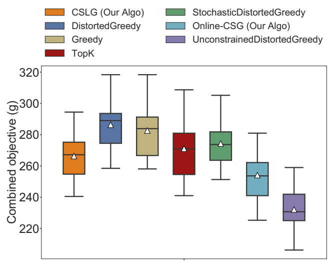
Figure 3: Running time (sec) comparisons of all algorithms for the UNCONSTRAINED problem.



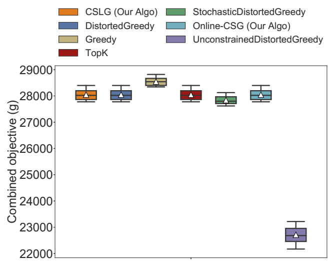
(a) ***NetHEPT***



(b) ***Guru***

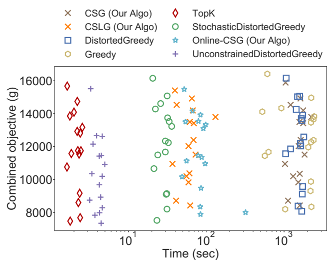


(c) ***Yelp***

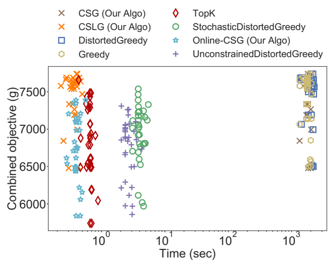


(d) ***Movielens***

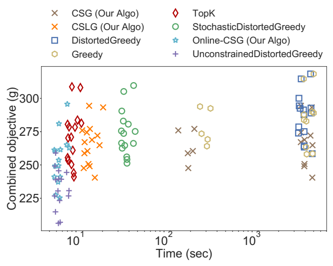
Figure 4: Combined objective value (g) comparisons of all algorithms for the UNCONSTRAINED problem.



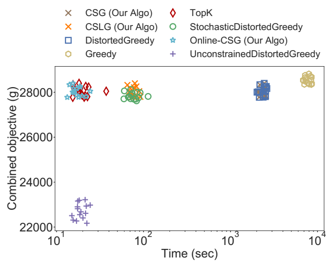
(a) ***NetHEPT***



(b) ***Guru***



(c) ***Yelp***



(d) ***Movielens***

Figure 5: Combined objective value (g) and running time (sec) comparisons of all algorithms for the UNCONSTRAINED problem.

we expect it to perform worse compared to the offline algorithms. Among the offline algorithms, the `StochasticDistortedGreedy` is slightly outperformed by `DistortedGreedy`, `CSLG` and `Greedy` in all datasets except from *Movielens* where it has the same performance. Finally, `DistortedGreedy`, `CSLG` and `Greedy` perform similarly, with the last performing slightly worse for larger k values. We note that even though `Greedy` is a heuristic without provable approximation guarantees it still performs well. The comparison between `DistortedGreedy` and `CSLG` shows that in practice the two algorithms perform the same for the cardinality constraint problem. Overall, we see that `CSLG` can achieve solutions of the same value as `DistortedGreedy` but is orders of magnitude faster as discussed above.

7.4 Evaluation for UNCONSTRAINED

Here we evaluate the algorithms for the UNCONSTRAINED problem. In this setting our algorithm can achieve even higher speedups at a slight performance cost.

Runtime analysis: We start by comparing the running times of each algorithm; the results are shown in Figure 3, where the y-axis is in the log-scale. The box plots show the medians of the runtime performance of each algorithm and the short lengths of the box plots indicate small deviations from the mean; that is, the running time of the algorithms is consistent among all random samples.

We observe that `CSG` and `DistortedGreedy` have similar running times, with the former being slightly faster. Overall however, their running times are orders of magnitude slower compared to the other algorithms. Let us now investigate the gains we get by using lazy evaluations on `CSG`. When considering our results at an application-wise level we see that the highest benefits are in the team-formation application (Figure 3b) where we see that CSLG achieves 1000x of speedup, compared to the standard greedy based approach without lazy evaluations and to DistortedGreedy. Slightly smaller gains are obtained in Figures 3a, 3c and 3d but they are still significant; i.e., in Figure 3a we see that while CSG and DistortedGreedy need half an hour to produce their results, CSLG requires only 2 minutes. In addition, we see that in these three cases a subset of the algorithms TopK, Online-CSG, DistortedGreedy and UnconstrainedDistortedGreedy is faster than CSLG but as we see next the performance of these algorithms with respect to the objective function is either similar to CSLG or worse.

Performance evaluation: For the performance evaluation we compare the combined objective value (g) of the solution of each algorithm. We present the results in Figure 4. The box plots show medians (solid line), means (triangle) and interquartile ranges for the combined objective of each algorithm.

We note that CSLG compares favorably to DistortedGreedy. The performance of the two algorithms is similar in the team-formation (Figures 4b) and movie-recommendation (Figures 4d) applications. In the remaining applications (Figures 4a and 4c), CSLG has slightly worse performance but faster running time than DistortedGreedy. Figure 5 summarizes the performance of all algorithms (across all iterations) for the two evaluation criteria: combined objective (y-axis) and running time (x-axis).

8 CONCLUSIONS

In this paper, we focused on the problem of balancing between the goal of optimizing a submodular function by picking a subset of elements from a collection with the actual cost of picking these elements. We formalized this problem as the problem of optimizing a non-negative monotone submodular function minus a linear cost function, and we designed effective and efficient algorithms with provable approximation guarantees. This framework enables us to generalize problem formulations that appear in many data-mining applications. In our experiments we demonstrate that our proposed algorithms are highly efficient and, despite their slightly weaker theoretical bounds compared to existing work, their practical performance is equivalent to the latter in terms of the objective function. Finally, although we focused here on the cardinality-constraint version of the k-Constrained problem, we point out that our results generalize to general matroid constraints, with significant practical and theoretical implications.

Acknowledgements: We acknowledge the support of NSF grants: III 1908510, III 1813406 and CAREER CCF-1750333.

REFERENCES

[1] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Online team formation in social networks. In *WWW*, pages 839–848, 2012.

[2] A. Anagnostopoulos, C. Castillo, A. Fazzone, S. Leonardi, and E. Terzi. Algorithms for hiring and outsourcing in the online labor market. In *ACM SIGKDD*, pages 1109–1118, 2018.

[3] A. Ashkan, B. Kveton, S. Berkovsky, and Z. Wen. Diversified utility maximization for recommendations. In *RecSys Posters*. Citeseer, 2014.

[4] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. Streaming submodular maximization: Massive data summarization on the fly. In *ACM SIGKDD*, pages 671–680, 2014.

[5] A. Badanidiyuru and J. Vondrák. Fast algorithms for maximizing submodular functions. In *SODA*, pages 1497–1514, 2014.

[6] A. Bhowmik, V. Borkar, D. Garg, and M. Pallan. Submodularity in team formation problem. In *SDM*, 2014.

[7] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 946–957. SIAM, 2014.

[8] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 155–166, 2012.

[9] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 199–208, 2009.

[10] A. Dasgupta, R. Kumar, and S. Ravi. Summarization through submodularity and dispersion. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1014–1022, 2013.

[11] A. Ene and H. L. Nguyen. A nearly-linear time algorithm for submodular maximization with a knapsack constraint. In *ICALP*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[12] A. Ene, S. M. Nikolakaki, and E. Terzi. Team formation: Striking a balance between coverage and cost. *arXiv preprint arXiv:2002.07782*, 2020.

[13] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45:634–652, 1998.

[14] M. Feldman. Guess free maximization of submodular and linear sums. In *Workshop on Algorithms and Data Structures*, pages 380–394. Springer, 2019.

[15] B. Golshan, T. Lappas, and E. Terzi. Profit-maximizing cluster hires. In *SIGKDD*, 2014.

[16] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[17] C. Harshaw, M. Feldman, J. Ward, and A. Karbasi. Submodular maximization beyond non-negativity: Guarantees, fast algorithms, and applications. In *ICML*, pages 2634–2643, 2019.

[18] M. Kargar, M. Zihayat, and A. An. Finding affordable and collaborative teams from a network of experts. In *SDM*, 2013.

[19] E. Kazemi, S. Minaee, M. Feldman, and A. Karbasi. Regularized submodular maximization at scale. *arXiv preprint arXiv:2002.03503*, 2020.

[20] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

[21] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[22] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in mapreduce and streaming. *ACM Transactions on Parallel Computing (TOPC)*, 2(3):1–22, 2015.

[23] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *ACM SIGKDD*, 2009.

[24] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429, 2007.

[25] Y. Li, J. Fan, Y. Wang, and K.-L. Tan. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1852–1872, 2018.

[26] Y. Li, D. Zhang, and K.-L. Tan. Real-time targeted influence maximization for online advertisements. 2015.

[27] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization techniques*, pages 234–243. Springer, 1978.

[28] B. Mirzasoleiman, A. Badanidiyuru, and A. Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *ICML*, pages 1358–1367. PMLR, 2016.

[29] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978.

[30] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions?i. *Mathematical programming*, 14(1):265–294, 1978.

[31] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of computer and system sciences*, 43(3):425–440, 1991.

[32] S. P. Parambath, N. Vijayakumar, and S. Chawla. Saga: A submodular greedy algorithm for group recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

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

[34] M. Sviridenko, J. Vondrák, and J. Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. *Mathematics of Operations Research*, 42(4):1197–1218, 2017.

[35] X. Yin, C. Qu, Q. Wang, F. Wu, B. Liu, F. Chen, X. Chen, and D. Fang. Social connection aware team formation for participatory tasks. *IEEE Access*, 2018.

A ANALYSIS OF ALGORITHM 1

In this section, we analyze our algorithm for the cardinality-constrained problem k-Constrained. We show the following guarantee.

Theorem A.1. *Algorithm 1 returns a solution Q of size at most k satisfying $f(Q) - c(Q) \geq \frac{1}{2} f(\mathrm{OPT}) - c(\mathrm{OPT})$.*

Proof. The starting point of our analysis is the following ordering of the elements in $Q \cup \mathrm{OPT}$, which we call the Greedy ordering. The Greedy ordering orders the elements in $Q \cup \mathrm{OPT}$ as

$$e_1, e_2, \ldots, e_{|Q \cup \mathrm{OPT}|} \qquad \text{(GreedyOrdering)}$$

where $e_i \in \arg\max_{e \in (Q \cup \mathrm{OPT}) \setminus \{e_1, \ldots, e_{i-1}\}} \tilde{g}(e_i | \{e_1, \ldots, e_{i-1}\})$ for each $i \in [|Q \cup \mathrm{OPT}|]$. That is, we select the next element e_i in the ordering to be the element from the remaining set with maximum marginal gain on top of the previously selected elements $e_1, \ldots, e_{i-1}$.

It follows from the execution of the algorithm that the first $|Q|$ elements in the Greedy ordering (GreedyOrdering) are the elements of Q in the order in which they were added to Q by the algorithm.

In the remainder of the analysis we first identify a solution, which is a prefix of the Greedy ordering, that we will analyze and show that its value is competitive with that of OPT. This solution is simply the first $\ell = |\mathrm{OPT}|$ elements in the Greedy ordering, and we denote it by $S^{(\ell)}$. We analyze this solution and relate its value to OPT. We then relate the value of the solution Q returned by the algorithm to the value of $S^{(\ell)}$.

Let $S^{(i)} = \{e_1, \ldots, e_i\}$ for all $1 \leq i \leq |Q \cup \mathrm{OPT}|$. Let $\ell = |\mathrm{OPT}|$. As noted above, the solution $S^{(\ell)}$ plays a key role in our analysis.

Relating $S^{(\ell)}$ to OPT. We now analyze the solution $S^{(\ell)}$ and relate it to OPT. To this end, we construct an appropriate mapping between $S^{(\ell)}$ and OPT as follows. Since $S^{(\ell)}$ and OPT have the same size, there is a bijection $\pi : \mathrm{OPT} \to S^{(\ell)}$ such that, for every $i \leq \ell$, $\pi^{-1}(e_i)$ appears after or at the same position as e_i in the Greedy ordering (GreedyOrdering), i.e., $\pi^{-1}(e_i) = e_j$ for some index $j \geq i$. We can obtain such a mapping π by iteratively matching each element of OPT to the earliest element of $S^{(\ell)}$ that is still unmatched. Since $|\mathrm{OPT}| = |S^{(\ell)}|$ and $S^{(\ell)}$ is comprised of the first ℓ elements in the Greedy ordering, every element $o \in \mathrm{OPT}$ will be matched to exactly one element $\pi(o) \in S^{(\ell)}$ such that $\pi(o)$ appears no later than o in the Greedy ordering, as needed.

We can use this bijective mapping π to "charge" OPT to $S^{(\ell)}$ as follows. By construction of the Greedy ordering and π, for every $i \leq \ell$, we have

$$\tilde{g}(e_i | S^{(i-1)}) \geq \tilde{g}(\pi^{-1}(e_i) | S^{(i-1)}) \qquad (8)$$

Let $\mathrm{OPT}^{(i)} = \pi^{-1}(S^{(i)})$ for all $i \leq \ell$. By submodularity and the fact that $\mathrm{OPT}^{(i)} = \mathrm{OPT}^{(i-1)} \cup \{\pi^{-1}(e_i)\}$, we have

$$\begin{aligned}\tilde{g}(\pi^{-1}(e_i) | S^{(i-1)}) &\geq \tilde{g}(\pi^{-1}(e_i) | S^{(\ell)} \cup \mathrm{OPT}^{(i-1)}) \\ &= \tilde{g}(S^{(\ell)} \cup \mathrm{OPT}^{(i-1)}) - \tilde{g}(S^{(\ell)} \cup \mathrm{OPT}^{(i-1)}) \qquad (9)\end{aligned}$$

By combining (8) and (9), we obtain

$$\tilde{g}(e_i | S^{(i-1)}) \geq \tilde{g}(S^{(\ell)} \cup \mathrm{OPT}^{(i-1)}) - \tilde{g}(S^{(\ell)} \cup \mathrm{OPT}^{(i-1)})$$

We sum up the above inequalities over all $i \leq \ell$. Note that the sums telescope. Additionally, we have $\mathrm{OPT}^{(\ell)} = \pi^{-1}(S^{(\ell)}) = \mathrm{OPT}$. Thus we obtain

$$\tilde{g}(S^{(\ell)}) - \tilde{g}(\emptyset) \geq \tilde{g}(S^{(\ell)} \cup \mathrm{OPT}) - \tilde{g}(S^{(\ell)})$$

and thus

$$\tilde{g}(S^{(\ell)}) \geq \frac{1}{2} \tilde{g}(S^{(\ell)} \cup \mathrm{OPT}) \qquad (10)$$

Relating Q to $S^{(\ell)}$. We now relate the solution Q constructed by the algorithm to the solution $S^{(\ell)}$. Recall that it follows from the execution of the algorithm that Q is a prefix of the Greedy ordering. By definition, $S^{(\ell)}$ is also a prefix of the Greedy ordering. However, Q and $S^{(\ell)}$ may be different prefixes and one may be included in the other, and we consider each of these cases in turn. To relate their values, we crucially use the following properties ensured by the algorithm: each element of Q has positive marginal gain with respect to the scaled objective $\tilde{g}$ on top of the elements that come before it in the Greedy ordering; additionally, if Q has less than k elements, all of the remaining elements have non-positive marginal gain with respect to $\tilde{g}$ on top of Q. These properties follow from the fact that, when each element is added to Q, it has positive marginal gain with respect to $\tilde{g}$. Moreover, the algorithm terminates when either it reaches the size constraint k or it terminates early on line 5 since the marginal gains of the remaining elements are non-positive with respect $\tilde{g}$.

We now give the precise analysis. We will show that $\tilde{g}(Q) \geq \tilde{g}(S^{(\ell)})$ by considering two cases: $|Q| \geq \ell$ and $|Q| < \ell$.

Suppose $|Q| \geq \ell$. We have $S^{(\ell)} \subseteq Q$. Since the algorithm only adds elements with positive marginal gain, we have

$$\tilde{g}(Q) - \tilde{g}(S^{(\ell)}) = \sum_{i=\ell+1}^{|Q|} \tilde{g}(e_i | S^{(i-1)}) \geq 0$$

Suppose $|Q| < \ell$. We have $Q \subseteq S^{(\ell)}$. Since the algorithm terminates when the marginal gain of every element becomes non-positive, we have

$$\tilde{g}(S^{(\ell)}) - \tilde{g}(Q) = \sum_{i=|Q|+1}^{\ell} \tilde{g}(e_i | S^{(i-1)}) \leq \sum_{i=|Q|+1}^{\ell} \tilde{g}(e_i | Q) \leq 0$$

Thus, in either case, we have that

$$\tilde{g}(Q) \geq \tilde{g}(S^{(\ell)}) \qquad (11)$$

Relating Q to OPT. We now put everything together and establish the approximation guarantee. By (10) and (11), we have

$$\tilde{g}(Q) \geq \frac{1}{2} \tilde{g}(S^{(\ell)} \cup \mathrm{OPT})$$

Recall that $\tilde{g} = f - 2c$. Thus

$$f(Q) - c(Q) \geq \frac{1}{2} f(S^{(\ell)} \cup \mathrm{OPT}) - c(S^{(\ell)} \cup \mathrm{OPT}) + c(Q)$$

Since f is monotone, we have $f(S^{(\ell)} \cup \mathrm{OPT}) \geq f(\mathrm{OPT})$. Thus

$$f(Q) - c(Q) \geq \frac{1}{2} f(\mathrm{OPT}) - c(S^{(\ell)} \cup \mathrm{OPT}) + c(Q)$$

Thus, to finish the proof, it only remains to verify that

$$c(\mathrm{OPT}) + c(Q) \geq c(S^{(\ell)} \cup \mathrm{OPT})$$

As before, we consider two cases: $|Q| \geq \ell$ and $|Q| < \ell$. Suppose that $|Q| \geq \ell$. Then $S^{(\ell)} \subseteq Q$ and thus $c(Q) \geq c(S^{(\ell)})$, since c is non-negative. Thus

$$c(\mathrm{OPT}) + c(Q) \geq c(\mathrm{OPT}) + c(S^{(\ell)}) \geq c(\mathrm{OPT} \cup S^{(\ell)})$$

Suppose that $|Q| < \ell$. Then $Q \subseteq S^{(\ell)}$ and $S^{(\ell)} \setminus Q \subseteq \mathrm{OPT}$. Thus $\mathrm{OPT} \cup Q = \mathrm{OPT} \cup S^{(\ell)}$ and hence

$$c(\mathrm{OPT}) + c(Q) \geq c(\mathrm{OPT} \cup Q) = c(\mathrm{OPT} \cup S^{(\ell)})$$

Putting everything together, we have

$$f(Q) - c(Q) \geq \frac{1}{2} f(\mathrm{OPT}) - c(\mathrm{OPT})$$

□

B ANALYSIS OF ALGORITHM 3

In this section, we analyze our streaming algorithm for the k-Constrained problem. The following theorem assumes that the parameters can be set appropriately if we know the value of the optimal solution. This assumption can be removed using a technique due to [4].

Theorem B.1. *When run with scaling constant $s = \frac{1}{2}\left(3 + \sqrt{5}\right)$ and threshold $\tau = \frac{1}{k}\left(\frac{1}{2}(3 - \sqrt{5}) f(\mathrm{OPT}) - c(\mathrm{OPT})\right)$, Algorithm 3 returns a solution Q such that $|Q| \leq k$ and*

$$f(Q) - c(Q) \geq \frac{1}{2}\left(3 - \sqrt{5}\right) f(\mathrm{OPT}) - c(\mathrm{OPT})$$

Proof. It is clear from the execution of the algorithm that $|Q| \leq k$. Therefore we focus on analyzing the function value. We consider two cases, depending on whether $|Q| = k$ or $|Q| < k$.

Case 1: $|Q| = k$. We have

$$\tilde{g}(Q) \geq \tau k \Rightarrow f(Q) - s \cdot c(Q) \geq \tau k$$

Case 2: $|Q| < k$. For every item $o \in \mathrm{OPT} \setminus Q$, we have $\tilde{g}(o|Q) \leq \tau$. This is due to the fact that o had marginal gain less than τ when it arrived and the marginal gains can only decrease due to submodularity of $\tilde{g}$. Therefore

$$\begin{aligned}\tau|\mathrm{OPT} \setminus Q| &\geq \sum_{o \in \mathrm{OPT} \setminus Q} \tilde{g}(o|Q) \\ &\geq \tilde{g}(Q \cup \mathrm{OPT}) - \tilde{g}(Q) \\ &= \Big(\underbrace{f(Q \cup \mathrm{OPT})}_{\geq f(\mathrm{OPT})} - f(Q)\Big) - s\Big(\underbrace{c(Q \cup \mathrm{OPT}) - c(Q)}_{= c(\mathrm{OPT} \setminus Q) \leq c(\mathrm{OPT})}\Big) \\ &\geq f(\mathrm{OPT}) - f(Q) - s \cdot c(\mathrm{OPT})\end{aligned}$$

The third inequality is by monotonicity of f and non-negativity and linearity of c. The second inequality follows from submodularity as follows. Let $O = \mathrm{OPT} \setminus Q$ and let $o_1, o_2, \ldots, o_{|O|}$ be an arbitrary ordering of O. Let $O^{(i)} = \{o_1, \ldots, o_i\}$. Then

$$\tilde{g}(Q \cup O) - \tilde{g}(Q) = \sum_{i=1}^{|O|} \left(\tilde{g}(Q \cup O^{(i)}) - \tilde{g}(Q \cup O^{(i-1)})\right)$$
$$= \sum_{i=1}^{|O|} \tilde{g}(o_i | Q \cup O^{(i-1)}) \leq \sum_{i=1}^{|O|} \tilde{g}(o_i|Q)$$

where the inequality is by submodularity.

Rearranging, we obtain

$$\begin{aligned} f(Q) &\geq f(\mathrm{OPT}) - s \cdot c(\mathrm{OPT}) - \tau \underbrace{|\mathrm{OPT} \setminus Q|}_{\leq k} \\ &\geq f(\mathrm{OPT}) - s \cdot c(\mathrm{OPT}) - \tau k\end{aligned}$$

On the other hand, since the algorithm only added elements with marginal gain at least the threshold, we can show that

$$\tilde{g}(Q) \geq \tau |Q|$$

Indeed, let $e_1, e_2, \ldots, e_{|Q|}$ be the elements of Q in the order in which they were added. Let $Q^{(i)} = \{e_1, \ldots, e_i\}$. We have

$$\tilde{g}(Q) - \tilde{g}(\emptyset) = \sum_{i=1}^{|Q|} \left(\tilde{g}(Q^{(i)}) - \tilde{g}(Q^{(i-1)})\right) = \sum_{i=1}^{|Q|} \tilde{g}(e_i | Q^{(i-1)}) \geq \tau |Q|$$

Since $\tilde{g}(\emptyset) = f(\emptyset) - c(\emptyset) = f(\emptyset) \geq 0$, we have $\tilde{g}(Q) \geq \tau|Q|$. Thus

$$f(Q) - s \cdot c(Q) \geq \tau |Q| \geq 0$$

To summarize, we showed the following two inequalities:

$$f(Q) \geq f(\mathrm{OPT}) - s \cdot c(\mathrm{OPT}) - \tau k$$
$$f(Q) - s \cdot c(Q) \geq 0$$

Combining the two inequalities with coefficients $s - 1$ and 1 gives

$$f(Q) - c(Q) \geq \frac{s-1}{s}\left(f(\mathrm{OPT}) - s \cdot c(\mathrm{OPT}) - \tau k\right)$$

Setting s, τ. We now put together the two cases and set the two parameters $s \geq 1$ and τ.

In case 1, we obtain a solution Q with value

$$f(Q) - c(Q) \geq f(Q) - s \cdot c(Q) \geq \tau k$$

where the first inequality is due to $c \geq 0$ and $s \geq 1$, and the second inequality is by our analysis above.

In case 2, we obtain a solution Q with value

$$f(Q) - c(Q) \geq \frac{s-1}{s}\left(f(\mathrm{OPT}) - s \cdot c(\mathrm{OPT}) - \tau k\right)$$

Thus overall we get a solution with value at least

$$\min\left\{\tau k, \frac{s-1}{s}\left(f(\mathrm{OPT}) - s \cdot c(\mathrm{OPT}) - \tau k\right)\right\}$$

We set τ to balance the two terms:

$$\tau k = \frac{s-1}{2s-1}\left(f(\mathrm{OPT}) - s \cdot c(\mathrm{OPT})\right)$$

We set s so that the coefficient of $c(\mathrm{OPT})$ becomes 1:

$$\frac{s(s-1)}{2s-1} = 1 \Rightarrow s^2 - 3s + 1 = 0$$

The above equation has two solutions: $s_1 = \frac{1}{2}\left(3 - \sqrt{5}\right)$ and $s_2 = \frac{1}{2}\left(3 + \sqrt{5}\right)$. We want $s \geq 1$, so we pick the latter. For this choice, the threshold τ and the objective value obtained are

$$\tau = \frac{1}{k}\left(\frac{1}{2}\left(3 - \sqrt{5}\right) f(\mathrm{OPT}) - c(\mathrm{OPT})\right)$$
$$f(Q) - c(Q) \geq \frac{1}{2}\left(3 - \sqrt{5}\right) f(\mathrm{OPT}) - c(\mathrm{OPT})$$

□