Data Summarization at Scale: A Two-Stage Submodular Approach

Marko Mitrovic [*1] **Ehsan Kazemi** [*1] **Morteza Zadimoghaddam** [2] **Amin Karbasi** [1]

Abstract

The sheer scale of modern datasets has resulted in a dire need for summarization techniques that can identify representative elements in a dataset. Fortunately, the vast majority of data summarization tasks satisfy an intuitive diminishing returns condition known as submodularity, which allows us to find nearly-optimal solutions in linear time. 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 this paper, 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.

1. Introduction

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. In this paper, we focus on a class of functions called **submodular** functions. These functions exhibit a mathematical diminishing returns property that allows us to find nearly-optimal solutions 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.

[*]Equal contribution [1]Department of Computer Science, Yale University, New Haven, Connecticut, USA [2]Google Research, Zurich, Switzerland. Correspondence to: Ehsan Kazemi <ehsan.kazemi@yale.edu>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

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.

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. In this

Table 1. 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.

Algorithm	**Approx.**	**Time Complexity**	**Setup**	**Function**
LOCALSEARCH (Balkanski et al., 2016)	$^1/_2(1 - ^1/_e)$	$O(km\ell n^2 \log n)$	Centralized	Coverage functions only
REPLACEMENT-GREEDY (Stan et al., 2017)	$^1/_2(1 - ^1/_{e^2})$	$O(km\ell n)$	Centralized	Submodular functions
REPLACEMENT-STREAMING (ours)	$^1/_7$	$O(kmn \log \ell)$	Streaming	Submodular functions
REPLACEMENT-DISTRIBUTED (R) (ours)	$^1/_4(1 - ^1/_{e^2})$	$O(^{km\ell n}/_{\mathsf{M}} + \mathsf{M}km\ell^2)$	Distributed	Submodular functions
DISTRIBUTED-FAST (R) (ours)	0.107	$O(^{kmn \log \ell}/_{\mathsf{M}} + \mathsf{M}km\ell^2 \log \ell)$	Distributed	Submodular functions

paper, we build on existing work to provide solutions for two-stage submodular maximization in both the streaming and distributed settings. Table 1 summarizes the theoretical results of this paper and compares them with the previous state of the art. **The proofs of all the theoretical results are deferred to the Supplementary Material.**

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 & Bilmes (2014) and Feldman et al. (2018) also worked on submodular image summarization, while Lin & Bilmes (2011) and Wei et al. (2013) focused on document summarization.

In addition to data summarization, submodularity appears in a wide variety of other machine learning applications including variable selection (Krause & Guestrin, 2005), recommender systems (Gabillon et al., 2013), crowd teaching (Singla et al., 2014), neural network interpretability (Elenberg et al., 2017), robust optimization (Kazemi et al., 2017), network monitoring (Gomez Rodriguez et al., 2010), and influence maximization in social networks (Kempe et al., 2003).

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 & Zadimoghaddam (2015), as well as Kumar et al. (2015). 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 when required.

Repeated optimization of related submodular functions has been a well-studied problem with works on structured prediction (Lin & Bilmes, 2012), submodular bandits (Yue & Guestrin, 2011; Chen et al., 2017), and online submodular optimization (Jegelka & 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.

3. Problem Definition

In general, if we want to optimally choose ℓ out of n items, we need to consider every single one of the exponentially many possibilities. This makes the problem intractable for any reasonable number of elements, let alone the billions of elements that are common in modern datasets. Fortunately, many data summarization formulations satisfy an intuitive diminishing returns property known as **submodularity**.

More formally, a set function $f : 2^V \to \mathbb{R}$ is **submodular** (Fujishige, 2005; Krause & Golovin, 2012) if, for all sets $A \subseteq B \subseteq V$ and every element $v \in V \setminus B$, we have $f(A + v) - f(A) \geq f(B + v) - f(B)$.[1] That is, the marginal contribution of any element v to the value of $f(A)$ diminishes as the set A grows.

Moreover, a submodular function f is said to be **monotone** if $f(A) \leq f(B)$ for all sets $A \subseteq B \subseteq V$. That is, adding elements to a set cannot decrease its value. Thanks to a celebrated result by Nemhauser et al. (1978), we know that if our function f is monotone submodular, then the classical greedy algorithm will obtain a $(1 - 1/e)$-approximation to the optimal value. Therefore, we can nearly-optimize monotone submodular functions in linear time.

Now we formally re-state the problem we are going to solve.

Problem Statement. 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:

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

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 $F = (f_1, \dots, f_m)$ drawn from $\mathbb{D}$. Therefore, the best approximation we have is to optimize the following related function:

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

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 the paper we will use T_i to denote the approximately-optimal size k solution we select for each f_i. Table 2 (Appendix A) 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).

[1]For notational convenience, we use $A + v = A \cup \{v\}$.

4. Streaming Algorithm

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 paper, 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, $\text{REP}_i(x, A)$ 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.,

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

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

Algorithm 1 EXCHANGE

```
1: Input: $u, S, \{T_i\}, \tau$ and $\alpha$      $\{\nabla_i$ terms use $\alpha\}$
2: if $|S| < \ell$ then
3:   if $\frac{1}{m}\sum_{i=1}^{m} \nabla_i(u, T_i) \geq \tau$ then
4:     $S \leftarrow S + u$
5:     for $1 \leq i \leq m$ do
6:       if $\nabla_i(u, T_i) > 0$ then
7:         if $|T_i| < k$ then
8:           $T_i \leftarrow T_i + u$
9:         else
10:          $T_i \leftarrow T_i + u - \text{REP}(u, T_i)$
```

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 1 explains EXCHANGE in detail.

Now we define the optimum solution to Eq. (2) by

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

where the optimum solution to each function is defined by

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

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

In Section 4.1, 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 4.2, 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. Knowing OPT

If OPT is somehow known a priori, we can use REPLACEMENT-STREAMING-KNOW-OPT. As shown in Algorithm 2, 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

Algorithm 2 REPLACEMENT-STREAMING-KNOW-OPT

```
1: Input: OPT, $\alpha$ and $\beta$
2: Output: Sets $S$ and $\{T_i\}_{1 \leq i \leq m}$, where $T_i \subset S$
3: $S \leftarrow \varnothing$ and
4: $T_i \leftarrow \varnothing$ for all $1 \leq i \leq m$
5: for every arriving element $u^t$ do
6:   EXCHANGE($u^t, S, \{T_i\}, \frac{\text{OPT}}{\beta\ell}, \alpha$)
7: Return: $S$ and $\{T_i\}_{1 \leq i \leq m}$
```

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 2.

Theorem 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* $1/6$.

4.2. 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 1, which provides bounds on OPT.

Lemma 1. *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 1, 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 2, 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 2. *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

Algorithm 3 REPLACEMENT-STREAMING

```
1:  $\Gamma^0 = \{(1+\epsilon)^l | l \in \mathbb{Z}\}$
2:  For each $\tau \in \Gamma^0$ set $S_\tau \leftarrow \varnothing$ and $T_{\tau,i} \leftarrow \varnothing$ for all
    $1 \le i \le m$   {Maintain the sets lazily}
3:  $\delta^0 \leftarrow 0$
4:  for every arriving element $u^t$ do
5:      $\delta^t = \max\{\delta^{t-1}, \frac{1}{m}\sum_{i=1}^m f_i(u^t)\}$
6:      $\Gamma^t = \{(1+\epsilon)^l \mid l \in \mathbb{Z}, \frac{\delta^t}{(1+\epsilon)\cdot\beta\cdot\ell} \le (1+\epsilon)^l \le \delta^t\}$
7:      Delete all $S_\tau$ and $T_{\tau,i}$ such that $\tau \notin \Gamma^t$
8:      for all $\tau \in \Gamma^t$ do
9:          EXCHANGE($u^t, S_\tau, \{T_{\tau,i}\}_{1\le i\le m}, \tau, \alpha$)
10: Return: $\arg\max_{\tau\in\Gamma^n}\{\frac{1}{m}\sum_{i=1}^m f_i(T_{\tau,i})\}$
```

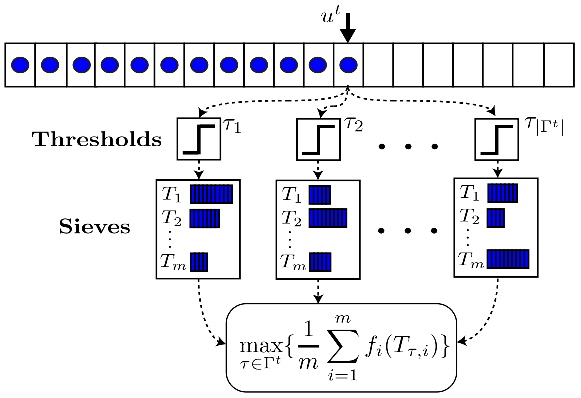


Figure 1. Illustration of REPLACEMENT-STREAMING. Stream of data arrives at any arbitrary order. At each step t, the set of thresholds Γ^t is updated based on a new estimation of δ^t. Note that at each time the number of such thresholds is bounded. For each $\tau \in \Gamma^t$ there is a running instance of the streaming algorithm.

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 2 we conclude that the marginal gain of all the $u^{t'}, t' \le t$ is less than τ and EXCHANGE would not have picked them. The REPLACEMENT-STREAMING algorithm is shown pictorially in Figure 1 and the pseudocode is given in Algorithm 3.

Theorem 2. *Algorithm 3 satisfies the following properties:*

- *It outputs sets S and $\{T_i\} \subset S$ for $1 \le i \le m$, such that $|S| \le \ell, |T_i| \le k$ and $\frac{1}{m}\sum_{i=1}^m f_i(T_i) \ge \min\{\frac{\alpha(\beta-1)}{\beta((\alpha+1)^2+\alpha)}, \frac{1}{\beta(1+\epsilon)}\} \cdot \text{OPT}$.*
- *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 $O(\frac{km\log\ell}{\epsilon})$.*

Algorithm 4 REPLACEMENT-DISTRIBUTED

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

5. Distributed Algorithm

In recent years, there have been several successful approaches to the problem of distributed submodular maximization (Kumar et al., 2015; Mirzasoleiman et al., 2013; Mirrokni & Zadimoghaddam, 2015; Barbosa et al., 2015). Specifically, Barbosa et al. (2015) proved that the following simple 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 4), 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 Appendix F for a detailed explanation of REPLACEMENT-GREEDY.

Theorem 3. *The* REPLACEMENT-DISTRIBUTED *algorithm outputs sets $S^*, \{T_i^*\} \subset S$, with $|S^*| \le \ell, |T_i^*| \le k$, such that*

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

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

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 ordering 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 Appendix E).

Theorem 4. *The* DISTRIBUTED-FAST *algorithm outputs sets* $S^*, \{T_i^*\} \subset S$, *with* $|S^*| \leq \ell, |T_i^*| \leq k$, *such that*

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

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 3 and 4, 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.

6. Applications

In this section, we evaluate the performance of our algorithms in both the streaming and distributed settings. We compare our work against several different baselines.

6.1. 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 (an example is given in Appendix G). 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 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.

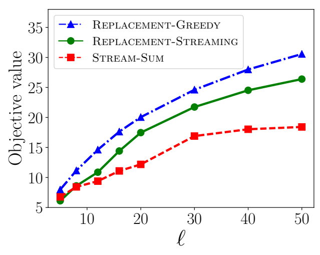
To evaluate the various algorithms, we consider two primary metrics: the objective value, which we define as $\sum_{i=1}^{m} f_i(T_i)$, and the wall-clock running time. We note that the trials were run using Python 2.7 on a quad-core Linux machine with 3.3 GHz Intel Core i5 processors and 8 GB of RAM. Figure 2 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 also see that REPLACEMENT-STREAMING outperforms the streaming baseline STREAM-SUM in both objective value and running time.

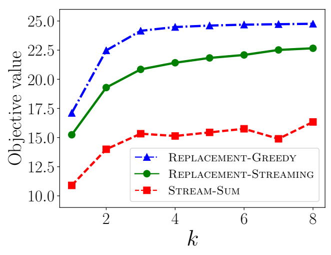
Another noteworthy observation from Figure 2(c) is that ϵ can be increased all the way up to $\epsilon = 0.5$ before we start to see loss in the objective value. Recall that ϵ is the parameter that trades off the accuracy of REPLACEMENT-STREAMING with the running time by changing the granularity of our guesses for OPT. As seen Figure 2(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 2(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.

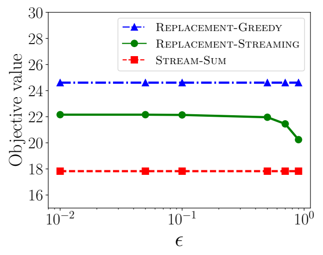
Figure 3 shows some sample images selected by REPLACEMENT-GREEDY (top) and REPLACEMENT-



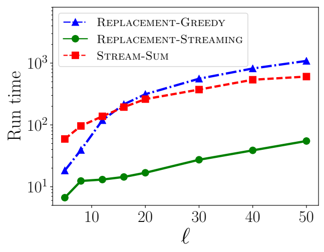
(a) $k = 5$, $\epsilon = 0.5$



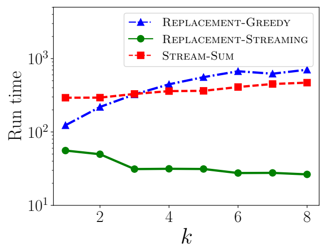
(b) $\ell = 25$, $\epsilon = 0.5$



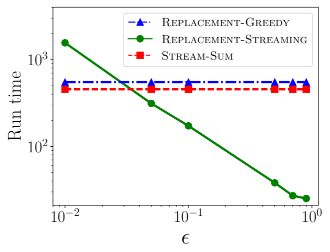
(c) $\ell = 25$, $k = 5$



(d) $k = 5$, $\epsilon = 0.5$



(e) $\ell = 25$, $\epsilon = 0.5$



(f) $\ell = 25$, $k = 5$

Figure 2. 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.

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.

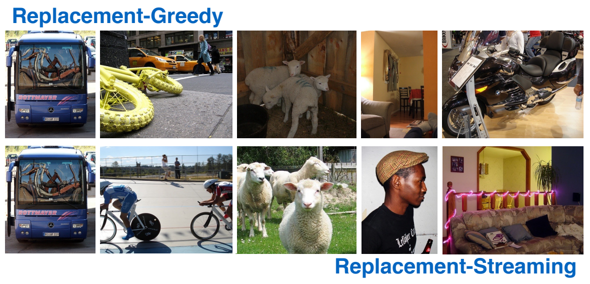


Figure 3. Representative images selected in the different settings.

6.2. 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 $R_1, \dots, R_m$ 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 4(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.

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 ex-

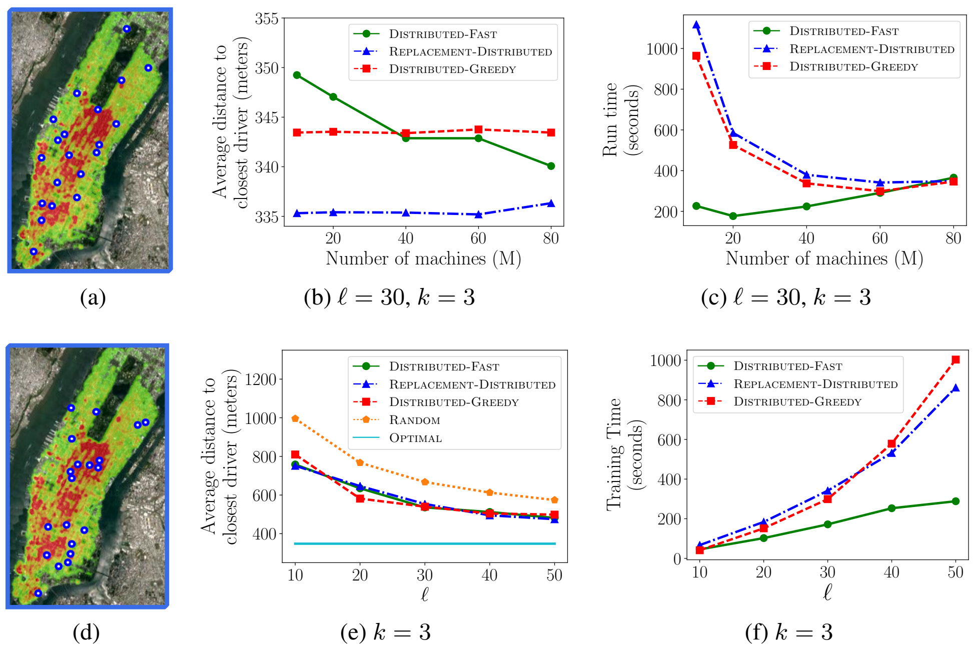


Figure 4. (a) shows a heatmap of all pick-up locations, as well as the centers of the twenty random regions that define each function f_i. (b) and (c) show the effects of changing the number of machines we use to distribute the computation. (d) shows the centers of the twenty new regions (chosen from the same distribution) used for the evaluation in (e). (f) shows the training time for each summary used in (e).

pensive. Hence, we will use this setup to evaluate the two distributed algorithms we presented in Section 5. 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 4(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 4(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 5. 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 4(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 4(d).

In Figure 4(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 4(f), we confirm that DISTRIBUTED-FAST is indeed the fastest algorithm for constructing each summary. Note that 4(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.

7. Conclusion

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

Acknowledgements

Amin Karbasi was supported by a DARPA Young Faculty Award (D16AP00046) and a AFOSR Young Investigator Award (FA9550-18-1-0160). Ehsan Kazemi was supported by the Swiss National Science Foundation (Early Postdoc.Mobility) under grant number 168574.

References

Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., and Krause, A. Streaming submodular maximization: Massive data summarization on the fly. In *ACM KDD*, 2014.

Balkanski, E., Mirzasoleiman, B., Krause, A., and Singer, Y. Learning sparse combinatorial representations via two-stage submodular maximization. In *ICML*, 2016.

Barbosa, R., Ene, A., Nguyen, H., and Ward, J. The power of randomization: Distributed submodular maximization on massive datasets. In *International Conference on Machine Learning*, pp. 1236–1244, 2015.

Buchbinder, N., Feldman, M., and Schwartz, R. Online submodular maximization with preemption. In *SODA*. Society for Industrial and Applied Mathematics, 2015.

Chen, L., Krause, A., and Karbasi, A. Interactive submodular bandit. In *NIPS*, 2017.

Elenberg, E., Dimakis, A. G., Feldman, M., and Karbasi, A. Streaming weak submodularity: Interpreting neural networks on the fly. In *NIPS*, 2017.

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

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

Feldman, M., Karbasi, A., and Kazemi, E. Do Less, Get More: Streaming Submodular Maximization with Subsampling. *CoRR*, abs/1802.07098, 2018. URL http://arxiv.org/abs/1802.07098.

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

Gabillon, V., Kveton, B., Wen, Z., Eriksson, B., and Muthukrishnan, S. Adaptive submodular maximization in bandit settings. In *NIPS*, 2013.

Gomez Rodriguez, M., Leskovec, J., and Krause, A. Inferring networks of diffusion and influence. In *KDD*, 2010.

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

Kazemi, E., Zadimoghaddam, M., and Karbasi, A. Deletion-Robust Submodular Maximization at Scale. *CoRR*, abs/1711.07112, 2017. URL http://arxiv.org/abs/1711.07112.

Kempe, D., Kleinberg, J., and Tardos, E. Maximizing the spread of influence through a social network. In *KDD*, 2003.

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

Krause, A. and Golovin, D. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2012.

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

Kumar, R., Moseley, B., Vassilvitskii, S., and Vattani, A. Fast greedy algorithms in mapreduce and streaming. *ACM Transactions on Parallel Computing*, 2(3):14, 2015.

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

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

Mirrokni, V. and Zadimoghaddam, M. Randomized composable core-sets for distributed submodular maximization. In *STOC*. ACM, 2015.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. Distributed submodular maximization: Identifying representative elements in massive data. In *NIPS*, 2013.

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

Singla, A., Bogunovic, I., Bartók, G., Karbasi, A., and Krause, A. Near-optimally teaching the crowd to classify. In *ICML*, 2014.

Stan, S., Zadimoghaddam, M., Krause, A., and Karbasi, A. Probabilistic submodular maximization in sub-linear time. In *ICML*, 2017.

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

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

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

Yue, Y. and Guestrin, C. Linear submodular bandits and their application to diversified retrieval. In *NIPS*, 2011.