Scaling Submodular Maximization via Pruned Submodularity Graphs

Tianyi Zhou
University of Washington
tianyizh@uw.edu

Hua Ouyang
Apple
huaouyang@gmail.com

Jeff Bilmes
University of Washington
bilmes@uw.edu

Yi Chang
Huawei Research America
yichang@acm.org

Carlos Guestrin
University of Washington
guestrin@uw.edu

Abstract

We propose a new randomized pruning method (called "submodular sparsification (SS)") to reduce the cost of submodular maximization. The pruning is applied via a "submodularity graph" over the n ground elements, where each directed edge is associated with a pairwise dependency defined by the submodular function. In each step, SS prunes a $1-1/\sqrt{c}$ (for $c>1$) fraction of the nodes using weights on edges computed based on only a small number ($O(\log n)$) of randomly sampled nodes. The algorithm requires $\log_{\sqrt{c}} n$ steps with a small and highly parallelizable per-step computation. An accuracy-speed tradeoff parameter c, set e.g. as $c=8$, leads to a fast shrink rate $\sqrt{2}/4$ and small iteration complexity $\log_{2\sqrt{2}} n$. Analysis shows that w.h.p., the greedy algorithm on the pruned set of size $O(\log^2 n)$ can achieve a guarantee similar to that of processing the original dataset. In news and video summarization tasks, SS is able to substantially reduce both computational costs and memory usage, while maintaining (or even slightly exceeding) the quality of the original (and more costly) greedy algorithm.

1 Introduction

Machine learning applications benefit from the existence of large volumes of data. The recent explosive growth of data, however, poses serious challenges both to humans and machines. One of the primary goals of a summarization process is to select a representative subset that reduces redundancy but preserves fidelity to the original data [19]. Any further processing on only a summary (a small representative set) by either a human or machine thus reduces computation, memory requirements, and overall effort. Summarization has many applications such as news digesting, photo stream presenting, data subset selection, and video thumbnailing. A summarization algorithm, however, involves challenging combinatorial optimization problems, whose quality and speed heavily depend on the objective that assigns quality scores to candidate summaries.

Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, Florida, USA. JMLR: W&CP volume 54. Copyright 2017 by the author(s).

Submodular functions [11, 19] are broadly applied as objectives for summarization, since they naturally capture redundancy amongst groups of data elements. A submodular function is a set function $f: 2^V \to \mathbb{R}$ with a diminishing returns property, i.e., given a finite "ground" set V, and any $A \subseteq B \subseteq V$ and a $v \notin B$, we have:

$$f(v \cup A) - f(A) \geq f(v \cup B) - f(B). \qquad (1)$$

This implies v is more important to the smaller set A than to the larger set B. The increase $f(v \cup A) - f(A)$ reflects the importance of v to A and is called the "marginal gain" $f(v|A)$ of v conditioned on A. The objective $f(\cdot)$ can be chosen from a large family of functions (e.g., including but not limited to facility location and set cover functions). Usually one requires a small summary, so a cardinality-based budget is used. Hence, a summarization task can be cast as the following:

$$\max_{S \subseteq V, |S| \leq k} f(S). \qquad (2)$$

Knapsacks and matroids are also often used as constraints. In this paper, however, we will primarily be concerned with cardinality constraints, but our methods do generalize to other constraints as well.

Though submodular maximization is NP-hard, a near optimal solution of (2) can be achieved via the greedy algorithm, having an approximation factor of $1-1/e$ [25]. The greedy algorithm starts with $S \leftarrow \emptyset$, and selects the next element with the largest marginal gain $f(v|S)$ from $V \backslash S$, i.e., $S \leftarrow S \cup \{v^*\}$ where $v^* \in \operatorname{argmax}_{v \in V \backslash S} f(v|S)$, and this repeats until $|S| = k$. It is simple to implement and usually outperforms other methods, e.g., those based on integer linear programming.

Scaling up the greedy algorithm to very large data sizes (where $|V| = n$ is big) is a nontrivial practical problem. The per-step computation of greedy is expensive: each step needs to re-evaluate the marginal gains of all elements in $V \backslash S$ conditioned on the new S, and thus requires $O(n)$ function evaluations. In addition, each step depends on the results from previous steps, so the computation does not trivially parallelize. Moreover, one typically must keep all n elements in memory until the end of the algorithm, since any element might become the one with the largest marginal gain $f(v|S)$ as S grows. To overcome this problem, it would be helpful to have an economical screening method to reduce the data size before the costly submodular maximization is performed. While related work is described in §1.2, we next describe the contributions of this work.

1.1 Main Contribution

A submodular function f can describe higher order relationships among multiple (≥ 3) elements via $f(v|S)$. In the greedy algorithm, selecting important elements (for maximizing f) requires evaluating $f(v|S)$ for all $v \in V\backslash S$ in each step. In this paper, we show that removing unimportant elements from V only needs a rough estimate of $f(v|S)$, one that can be derived solely from pairwise relationships $f(v|u)$ for a small set of element pairs (u, v). We encode the pairwise relationships as edge weights in a "submodularity graph". By taking advantage of the properties of this graph, the size of the ground set V can be efficiently reduced from n to $O(\log^2 n)$ by randomly pruning the nodes on the graph according to a subset of the edge weights.

In particular, given objective f, we define a directed submodularity graph whose nodes are the n elements in V, and each edge $u \rightarrow v$ from tail u to head v is associated with a weight $w_{u\rightarrow v} \equiv w_{uv} = f(v|u) - f(u|V\backslash u)$ that reflects the worst-case net loss when maximizing f caused by removing v while retaining u ($f(v|u)$ is the greatest loss when removing v while retaining u while $f(u|V \setminus u)$ is the least gain of retaining u). Intuitively, removing head nodes from V with small-weight edges reduces the ground set from V to a (hopefully much) smaller V', and selecting elements from V' rather than V causes a small overall objective loss but can be much faster.

Finding, however, the smallest $V' \subseteq V$ such that the resulting objective loss can be upper bounded by some constant turns out to be another challenging non-monotone submodular maximization problem, leading to a chicken-and-egg situation. In addition, finding a near optimal solution to this problem requires computing weights on all $n(n-1) = O(n^2)$ edges. We instead propose a randomized pruning method called "submodular sparsification (SS)" to reduce the ground set. By leveraging a directed triangle inequality on the submodularity graph (Lemma 3), SS only needs to compute partial weights on a few randomly selected edges, and this only slightly increases the objective loss caused by using the reduced set V' rather than V. At each step, SS randomly samples $O(\log n)$ elements from V as probes, and removes a $1 - 1/\sqrt{c}$ fraction of head elements in V that have the smallest weights from amongst the randomly selected elements. When tradeoff parameter $c > 1$ increases, the success probability of the randomized algorithm increases, but memory size $|V'|$ also increases. With c set to $c = 8$, $1 - 1/\sqrt{c} \simeq 64.6\%$ of elements can be dropped per iteration, so the number of iterations $\log_{\sqrt{c}} n = \log_{2\sqrt{2}} n$ is small (≤ 5 in our experiments). The per-iteration complexity is dominated by pairwise edge-weight computations, which requires at most $O(n \log n)$ function evaluations. However, it is highly parallelizable, and is negligible if the objective is graph based. Hence, SS can be scaled to large data sizes. SS can be applied before any algorithm for submodular maximization even with a non-monotone objective or general constraints although the bound in Theorem 1&2 holds only in the monotone case — the other theoretical properties only rely on submodularity and can be extended to more general cases.

In experiments, we compare SS with the lazy greedy and sieve-streaming algorithm [1] on real-world news and video summarization datasets. Using the lazy greedy algorithm with an SS-reduced ground set, we achieve quality similar to that on the original ground set, but with computation and memory load greatly reduced and, in fact, comparable to a streaming algorithm whose quality is usually much worse than offline methods.

1.2 Related Work

A number of methods have been proposed to accelerate the speed of the greedy algorithm. Most of them, however, aim to reduce or distribute the computation rather than the memory, and rarely do they study how to reduce the ground set V. Therefore, their contributions are mostly complementary with SS (i.e., they can be combined with SS to further improve algorithmic scalability).

The lazy, or accelerated, greedy algorithm [20, 17] reduces the number of function evaluations per step by lazily updating a priority queue of marginal gains over all elements. At each step, the algorithm repeatedly updates $f(v|S)$ of the top element and re-inserts it to a queue until the top element does not change position in the queue — it then adds this element to the running solution. Due to submodularity, the lazy greedy algorithm has the same output and mathematical guarantee as the original greedy algorithm, but significantly reduces computation in practice, but in the worst case it is as slow (if not slower) than the original greedy algorithm.

Approximate greedy algorithms further reduce the number of function evaluations per step at a cost of a worse approximation factor. In [28, 2], each step only approximates identifying the element with the largest marginal gain $\max_{v\in V\backslash S} f(v|S)$ by finding any element whose marginal gain is larger than a fraction β of $\max_{v\in V\backslash S} f(v|S)$ of its upper bound. The "lazier than lazy greedy" approach [22] selects the element from a random subset $V' \subseteq V\backslash S$ of size $O(n/k)$ (more than $O(\log n)$) of SS in each step, so only the marginal gains of $v \in V'$ need be computed. A similar algorithm in [7] randomly selects an element from a reasonably good subset $V' \subseteq V\backslash S$ per step, and extends to the non-monotone case.

Streaming submodular maximization [1, 8, 9, 12, 4] studies how to approximate the greedy algorithm in one pass of data under a limited memory budget (i.e., the algorithm can access only a small number of elements in the stream history at a time). The best known approximation factor and hardness are both $1/2$ [1, 8], worse than the $1 - 1/e$ of the offline greedy algorithm.

Distributed and parallel greedy algorithms [23, 27, 3] typically partition the ground set into several not-necessarily disjoint pieces and assigns them to multiple machines, then run greedy on each machine, and finally combine the results. These approaches fall into the framework of composable coresets. The existence of such methods for some important submodular maximization problems is not always possible [14]. In [21], a $1/3$-randomized composable coreset method is proposed to achieve an expected bound for the combined solution. *The major difference of this paper is that we study how to reduce the ground set rather than partition it, by developing a coreset-like algorithm on a submodularity graph rather than running greedy algorithm to achieve coreset on each machine.*[1] However, by replacing the greedy algorithm on each machine with SS, we can further speed up distributed submodular maximization by speeding up the computation at each parallel node. Pruning methods by thresholding marginal gains was used for distributed submodular cover problem [24]. The major difference is that we use a graph based strategy, and our thresholding is applied to the edge weights rather than the marginal gains.

Another class of methods [16, 28] accelerates the greedy algorithm by maximizing a surrogate function whose evaluation is faster and cheaper than the original objective. The surrogate can be either a tight modular lower bound or a simpler submodular function. It can also be adaptively changed in each step to better approach the original objective. In [28], a simple pruning method is used to reduce V by exploiting $f(v|V\backslash v)$, a lower bound of $f(v|S)$ for $S \subseteq V$. E.g., element u whose singleton gain $f(u)$ is less than the k^{th} largest $f(v|V\backslash v)$ over all $v \in V$ can be safely removed. Besides exploiting the global redundancy of v via $f(v|V\backslash v)$, the weight w_{uv} used in SS further takes the pairwise relationship $f(v|u)$ into account for further ground set reduction[2]. SS can also use the pruning method in [28] as an initial step.

2 Submodularity Graph

We next introduce the "submodularity graph," a useful tool to explore the redundancy of ground sets V in a submodular maximization process.

Definition 1. *The submodularity graph is a weighted directed graph $G(V, E, w)$ defined by a normalized submodular function $f : 2^V \to \mathbb{R}_+$ where V is the set of nodes corresponding to the ground set, and each directed edge $e = (u \to v) = (u, v) \in E$ from u to v has weight:*

$$w_{uv} = f(v|u) - f(u|V\backslash u). \tag{3}$$

Intuitively, the weight w_{uv} measures the worst case net loss in maximizing $f(S)$ on a reduced set V' with v removed and u retained. In Eq. (3), $f(v|u)$ is the maximum possible gain v can offer a set involving u, while $f(u|V\backslash u)$ is the minimal possible gain u can contribute to the solution S because $f(u|S) \geq f(u|V\backslash u)$ holds by submodularity. Hence, a small $f(v|u)$ indicates v is unimportant if u is retained in a solution, while a large $f(u|V\backslash u)$ implies that u is always important. Taken together, a small w_{uv} would suggest removing v while keeping u. Note w_{uv} is a net loss, combining both the "local" importance of $f(v|u)$ and the "global" importance of $f(u|V\backslash u)$. Previous work such as [28] and curvature based methods [15] do not leverage local and global importance in the same way.

We further generalize $G(V, E)$ to a "conditional submodularity graph" $G(V, E|S)$ describing the pairwise relationships conditioned on set $S \subseteq V$. Accordingly, the edge weight on $e = (u, v)$ is:

$$w_{uv|S} = f(v|S \cup u) - f(u|V\backslash u). \tag{4}$$

$G(V, E|S)$ reduces to $G(V, E)$ when $S = \emptyset$, usually the starting set in a greedy submodular maximization procedure. Below we give a detailed analysis of how edge weight w_{uv} can be used to remove elements from V. For notational simplicity, we use "+" to denote the set union "$\cup$," and "−" for set subtraction "$\backslash$". Let's study two properties of $w_{uv|S}$.

Lemma 1. *If $P \subseteq S \subseteq V$, for any $u, v \in V$ such that $u, v \notin S$, $w_{uv|S} \leq w_{uv|P}$.*

Proof. Submodularity requires $f(v|S + u) \leq f(v|P + u)$. From the definition of $w_{uv|S}$ in (4), the conclusion is immediate. □

Lemma 2. *For any $u, v \in V$ and $S \subseteq V$, if $u \neq v$ and $u, v \notin S$, then*

$$f(v|S) \leq f(u|S) + w_{uv|S}. \tag{5}$$

Proof.

$$\begin{aligned} f(v|S) &= f(u|S) + f(v|u + S) - f(u|v + S) \\ &\leq f(u|S) + f(v|u + S) - f(u|V - u) \\ &= f(u|S) + w_{uv|S}. \end{aligned} \tag{6}$$

The first equality is obtained using the definition of the marginal gain, while the inequality is from submodularity and since $(v + S) \subseteq (V - u)$. □

Lemmas 1 and 2 state that the weight w_{uv} relates the two marginal gains of u and v relative to S. The gain $f(v|S)$ is important for various submodular maximization algorithms since it measures how much $f(S)$ is improved by adding v to S. Each step of the greedy algorithm selects the element with the largest $f(v|S)$, i.e., $S \leftarrow \text{argmax}_{x \in V} f(x|S) \cup S$, and $f(S)$ increases by $f(v|S)$.

[1]More discussion can be found at §5.13.

[2]A theoretical comparison is given in §5.12 of [30]

If $v \in \operatorname{argmax}_{x \in V \backslash S} f(x|S)$ should be selected by the greedy algorithm at the current step, but for some reason is missing in $V' \subseteq V$ (a reduced ground set), then greedy instead selects $u \in \operatorname{argmax}_{x \in V'} f(x|S)$. In this case, the objective $f(S)$ increases by $f(u|S) \leq f(v|S)$ rather than $f(v|S)$. By the relative optimality of u in V' and Lemma 2,

$$\begin{aligned} f(u|S) &\geq f(\operatorname*{argmin}_{x \in V' \backslash S} w_{xv|S}|S) \\ &\geq f(v|S) - \min_{x \in V' \backslash S} w_{xv|S}. \end{aligned} \tag{7}$$

Hence, the objective loss caused by removing v from V and using u instead is at most the minimal weight over all edges entering v from other elements in V'. In other words, an upper bound on the price for pruning v is $\min_{x \in V'} w_{xv|S}$, which reflects the contribution of v to the set V'. If it is small, the objective loss is, relatively speaking, negligible and v may be removed with impunity. We hence define this concept as a "divergence" of v from V' on $G(V, E|S)$:

Definition 2. *On the submodularity graph $G(V, E)$, the divergence $w_{V'v}$ of a node $v \in V$ from a set of nodes V' is defined as $w_{V'v} = \min_{x \in V'} w_{xv}$. Similarly, the divergence $w_{V'v|S}$ on the conditional submodularity graph $G(V, E|S)$ is defined as $w_{V'v|S} = \min_{x \in V' \backslash S} w_{xv|S}$.*

Although the edge weights w_{uv} are asymmetric, we next show that a directed triangle inequality holds on $G(V, E)$. This plays significant role in SS, since it provides an upper bound on an edge weight based on weights of adjacent edges, and thus avoids needing to compute all the edge weights exactly.

Lemma 3. *For $u, v, x \in V$, we have $w_{vx} \leq w_{vu} + w_{ux}$.*

The proof is given in [30]. A similar inequality also holds for $w_{uv|S}$ defined on $G(V, E|S)$.

3 Submodular Sparsification

In this section, we introduce submodular sparsification (SS), a randomized pruning algorithm that reduces V to $V' \subseteq V$ without drastically hurting the optimality of submodular maximization. Although pruning the conditional submodularity graph $G(V, E|S)$ with the greedy algorithm can rule out additional elements, here we focus on reducing V before running any submodular maximization algorithm, i.e., when $S = \emptyset$, but SS can be easily extended to $G(V, E|S)$.

3.1 Pruning as Submodular Maximization

According to Eq. (7) and Definition 2, small $w_{V'v}$ for all pruned elements $v \in V \backslash V'$ leads to small loss in the per-step increase of objective function by the greedy algorithm. By setting an upper bound ϵ for the loss, the following seeks the smallest pruned set V' for use in the maximization of f.

Definition 3 (submodular sparsification)**.** *The submodular sparsification problem is to solve:*

$$\max_{V' \subseteq V} h(V') := |\{v \in V \backslash V' : w_{V'v} \leq \epsilon\}|. \tag{8}$$

Proposition 1. *The objective function $h(\cdot)$ in Eq. 8 is non-monotone submodular.*

The proof is in [30]. Note the problem is ϵ-dependent. Let V^* be the optimal solution of Eq. (8) and $K \triangleq |V^*|$, then K decreases when increasing ϵ. When $\epsilon = 0$, the tolerance to the loss caused by pruning is zero. So no element can be removed (i.e., $V^* = V$), and running greedy on V^* has bound $\left(1 - e^{-1}\right) f(S^*)$. The proof also shows $h(V')$ is monotone in ϵ. Running greedy on V^* rather than V yields:

Theorem 1. *Let $S^* \in \operatorname{argmax}_{S \subseteq V, |S| \leq k} f(S)$, where $f : 2^V \to \mathbb{R}_+$ is normalized non-decreasing and submodular, let S' be a greedy solution to the problem $\max_{S \subseteq V^*, |S| \leq k} f(S)$. If $|V^*| \geq k$, the following approximation bound holds for S'.*

$$f(S') \geq \left(1 - e^{-1}\right) \left(f(S^*) - k\epsilon\right). \tag{9}$$

A proof of this is given in [30]. Unfortunately, solving Eq. (8) leads to a chicken-and-egg problem: even approximately solving this unconstrained non-monotone submodular maximization requires an expensive bi-directional randomized greedy algorithm [6] having approximation factor $1/2$ and that is slow in practice. Also, when f is not a graph based submodular function (such as facility location or saturated coverage), solving Eq. (8) requires a costly computation of the weights on all $n(n-1)$ edges.

3.2 Randomized Pruning

Drawing inspiration from bi-criteria k-clustering in Euclidean space [10], we develop a randomized pruning method ("submodular sparsification (SS)") on a submodularity graph to produce a reduced ground set V' without either computing all $n(n-1)$ weights or running bi-directional greedy. The submodular sparsification procedure is given in Algorithm 1. It starts from the original ground set V and an empty set V'. At each iteration, it randomly samples a

Algorithm 1 Submodular Sparsification (SS)

```
1:  Input: $V$, $f$, $r$, $c$
2:  Output: $V'$
3:  Initialize: $V' \leftarrow \emptyset$, $n \leftarrow |V|$
4:  while $|V| > r \log n$ do
5:      Sample $r \log n$ items uniformly at random from $V$
        and place them in $U$;
6:      $V \leftarrow V \backslash U$;
7:      $V' \leftarrow V' \cup U$;
8:      for $v \in V$ do
9:          $w_{Uv} \leftarrow \min_{u \in U} [f(v|u) - f(u|V \backslash u)]$
10:     end for
11:     Remove $(1 - 1/\sqrt{c})|V|$ elements from $V$ having the
        smallest $w_{Uv}$;
12: end while
13: $V' \leftarrow V \cup V'$
```

size-$(r \log n)$ set[3] of elements U from the current V, acting as probes to test the redundancy of the remaining elements in V, that are removed from V and added to V'. It then removes the top $(1 - 1/\sqrt{c})|V|$ elements from V having the smallest divergence w_{Uv} from U on $G(V, E)$ because of their unimportance to U. The procedure repeats and the size of V shrinks exponentially fast (with a shrink rate of $1/\sqrt{c}$) until it falls below a threshold. The parameter r controls the size of a probe set U and influences the size of the final V'. In our analysis below, we set $r = O(cK)$ for $c > 1$ to produce a sufficiently large success probability. In practice, we choose $c = 8$ to produce a fast shrink rate $1/\sqrt{c} = \sqrt{2}/4 < 1/2$, since it can remove more than half ($\approx 64.6\%$) of V per step. With $r = O(cK)$, since K is unknown in practice, we find that $r = 8$, also, empirically works well (see Section 4).

Algorithm 1 finishes in $\log_{\sqrt{c}} n$ iterations. It leads to small iteration complexity $\log_{2\sqrt{2}} n$ when $c = 8$. The per iteration computation is dominated by computing w_{Uv}, which requires calculating $O(n \log n)$ pairwise relationships. This can be simplified if f is graph based[4], because the first $O(n)$ greedy step already requires all of the pairwise similarities/distances needed for further f evaluations. When f is not graph based, this can be accelerated via parallelization, since disjoint pairs u, v in the set $\{f(u|v)\}_{u,v}$ may be independently computed. $f(u|V \setminus u)$ may be precomputed once in linear time.

3.3 Analysis of Submodular Sparsification

According to Lemma 2, a small w_{uv} leads to a small objective loss when v is removed and u retained. Instead of solving non-monotone submodular maximization in Eq. (8), SS randomly selects probes $u \in U$ to rule out elements v from V. The following lemma uses the directed triangle inequality in Lemma 3 to study which us, if sampled, can lead to a relatively small w_{uv} and thus a small w_{Uv} in Algorithm 1. Proofs of all the following results can be found in [30].

Lemma 4. *Let $u_v^* \in \operatorname{argmin}_{u \in V^*} w_{uv}$ be the tail node of an edge with the minimal weight over all edges from elements in V^* to head v. Then, for any item $v \in V$, $\forall u \in P(u_v^*) \cap Q(u_v^*)$ where*

$$\begin{aligned} P(u_v^*) &= \{u \in V : f(u + u_v^*) \le f(v + u_v^*)\}, \\ Q(u_v^*) &= \{u \in V : f(u) + f(u|V \backslash u) \ge \\ &\qquad f(u_v^*) + f(u_v^*|V \backslash u_v^*)\}. \end{aligned} \tag{10}$$

we have that $w_{uv} \le 2w_{u_v^* v}$.

Lemma 4 states that for any item v, if $P(u_v^*) \cap Q(u_v^*) \neq \emptyset$ and at least one $u \in P(u_v^*) \cap Q(u_v^*)$ is sampled in Algorithm 1, then w_{uv}, the maximal loss in $f(S)$ caused by dropping v, is sufficiently small, so v can be safely removed. The below discusses how to sample us and drop vs.

Proposition 2. *For an element $u^* \in V^*$ and $c > 1$, define its $|V|/(cK)$-NN ball $B(u^*, |V|/(cK))$ as the set of $|V|/(cK)$ elements in V with the smallest $f(u + u^*)$, and let $V_{u^*} = \{v \in V : u_v^* = u^*\}$ denote the set of elements ruled out by u^*. If one $u \in B(u^*, |V|/(cK)) \cap Q(u^*)$ is sampled into U in some iteration of Algorithm 1, then all the elements in V_{u^*} outside the ball fulfill the following:*

$$\forall v \in V_{u^*} \backslash B\Big(u^*, |V|/(cK)\Big), \;\; w_{uv} \le 2w_{u^* v}. \tag{11}$$

Based on Proposition 2, we can derive the maximal number of removed elements v whose importance represented by w_{Uv} cannot be upper bounded.

Proposition 3. *For each $u^* \in V^*$, if one $u \in B(u^*, |V|/(cK)) \cap Q(u^*)$ is sampled into U and added to V' in some iteration of Algorithm 1, then*

$$|\{x \in V : w_{Ux} \ge 2w_{V^* x}\}| \le |V|/c. \tag{12}$$

The following proposition explains why Algorithm 1 reduces ground set V exponentially by a ratio of $1 - 1/\sqrt{c}$. It also shows that all the pruned elements v satisfy $w_{Uv} \le 2w_{V^* v}$, which indicates that ruling out them from V will lead to at most a $2w_{V^* v}$ loss in objective $f(S)$.

Proposition 4. *Before line 11 of Algorithm 1, the following holds.*

$$|\{v \in V : w_{Uv} \le 2w_{V^* v}\}| \ge \left(1 - 1/\sqrt{c}\right)|V|. \tag{13}$$

Therefore, it is safe to remove the $1 - 1/\sqrt{c}$ fraction of items from V with the smallest w_{Uv}, since their importance w_{Uv} can be upper bounded. Proposition 4 results in the following Lemma.

Lemma 5. *For each $u^* \in V^*$, if at least one $u \in B(u^*, |V|/(cK)) \cap Q(u^*)$ is sampled and added into U, $\forall v \in V \backslash V'$ where V' is the output of Algorithm 1, we have $w_{V' v} \le 2w_{V^* v}$.*

Now we study the failure probability, i.e., the probability that the condition in Lemma 5 is not true.

Proposition 5. *If for each $u^* \in V^*$, $\Pr[u \in Q(u^*) | u \in B(u^*, |V|/(cK))] \ge q$ for an item u uniformly sampled from V, and if $r = O(cK) = pcK$, then the probability that no $u \in B(u^*, |V|/(cK)) \cap Q(u^*)$ is sampled and added into U for at least one $u^* \in V^*$ in at least one iteration of Algorithm 1 is at most $n^{1-qp} \log_{\sqrt{c}} n$.*

Although $\Pr[u \in Q(u^*) | u \in B(u^*, |V|/(cK))]$ is a data dependent term that is hard to analyze, there exist several choices for its lower bound q, which are more interpretable and increases when reducing c. The details are given in [30]. In §3.4, we provide an importance re-sampling method that can effectively increase q in practice.

By using Lemma 5 and Proposition 5, we replace ϵ in the proof of Theorem 1 with 2ϵ, which yields:

Theorem 2. *Under the assumptions in Proposition 5, the size of the output V' of Algorithm 1 is $|V'| =$*

[3]The base of all logarithms is 2 if not otherwise specified.

[4]Which means f is defined based on an underlying weighted graph whose weight is usually given by pairwise similarity. Examples include facility location and saturated coverage [19].

$(cp/\log\sqrt{c})K\log^2 n$. *With high probability, i.e.,* $1 - n^{1-qp}\log_{\sqrt{c}} n$*, we have that* $\forall v \in V\backslash V'$*,* $w_{V'v} \le 2w_{V^*v}$*, and thus the greedy algorithm on* V' *outputs a solution* S' *such that*

$$f(S') \ge \left(1 - e^{-1}\right)\left(f(S^*) - 2k\epsilon\right), \tag{14}$$

where S^* *is the optimal solution to Eq. (2), and* k *is the budget in Eq. (2).*

Remarks: Critically, via ϵ, the above analysis shows a tradeoff between: 1) the approximation bound, 2) the size of V' (the memory load), and 3) the computational cost. The approximation bound Eq. (14) can be improved if ϵ in Eq. (8) is small, but a smaller ϵ leads to larger $K = |V^*|$ (size of the optimal solution to Eq. (8)). This results in a larger reduced set V' of size $(cp/\log\sqrt{c})K\log^2 n$; and a larger V' produced by Algorithm 1 means more computation per step. It also shows a tradeoff between the success probability and $|V'|$ (which is proportional to the memory and the computational cost) via c and p: if c or p is large, the success probability $1 - n^{1-qp}\log_{\sqrt{c}} n$ increases, but $|V'|$ also increases. Given ϵ that measures the loss from approximate optimality (the $1 - 1/e$ guarantee), $K \in [1, |V|]$ measures the ϵ-reducibility of V. In Theorem 2, although K and q is hard to know, we can adjust c and p (by adjusting $c = pcK$) to achieve a small $|V'|$ and high success probability. SS fails when $K = |V|$ because $|V'| \ge K$. On real datasets we observe $|V'| \ll |V|$ even when ϵ is small, thus suggesting a large zone of practical success for SS.

SS can also reduce the ground set for non-monotone submodular maximization monotone under general constraints (e.g., knapsack or matroid) by applying it before any algorithm runs. All previous analysis still holds in general except Theorem 1 and Theorem 2, whose proofs rely on a cardinality constraint and monotonicity. They can be easily modified, however, by applying Eq. (18) to the proof procedure of the other algorithm's bound. The fundamental reason is that the properties (Lemmas 1-3) of weight w_{uv} on the submodularity graph $G(V, E)$ depend *only* on submodularity and non-negativity of f.

3.4 Additional Improvements

In practice, several techniques can be further applied to Algorithm 1 to improve either its effectiveness or efficiency. Firstly, the pruning technique based on $f(u|V\backslash u)$ proposed in [28] can be applied to V before running Algorithm 1 to rule out additional elements and save computation.

The second improvement would apply importance resampling to the uniformly sampled probes in U after Line 5 of Algorithm 1. According to Proposition 5, uniform sampling leads to large $\Pr\left[u \in B\left(u^*, |V|/(cK)\right)\right]$, while re-sampling u with large $f(u) + f(u|V\backslash u)$ increases the probability $\Pr\left[u \in Q(u^*_v)|u \in B\left(u^*, |V|/(cK)\right)\right]$ and thus its lower bound q, which results in a larger success probability $1 - n^{1-qp}\log_{\sqrt{c}} n$. Intuitively, large $f(u)$ suggests u may be important, while large $f(u|V\backslash u)$ indicates its importance is undiminished by other elements in V.

The third strategy is to further reduce V' by exploring its redundancy. In particular, after Algorithm 1, the bi-directional greedy algorithm [6] can be used to solve Eq. (8) defined on the reduced ground set V'. Since V' is much smaller than V, the cost may be acceptable.

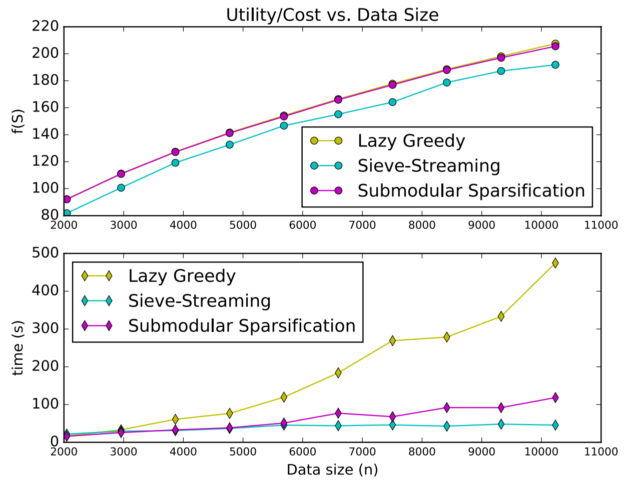

Figure 1: Utility $f(S)$ and time cost vs. size of data n

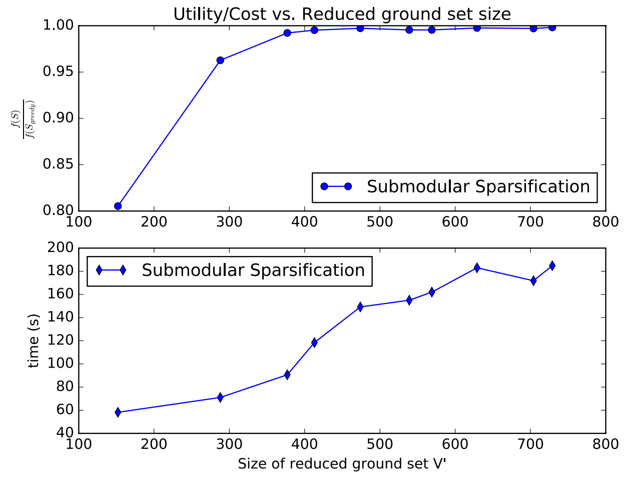

Figure 2: Relative utility $f(S)/f(S_{greedy})$ and time cost associated with different sizes of reduced set V', which correspond to 10 different values of r varying between $[2, 20]$ with step size 2.

4 Experiments

In this section, on several news and video datasets, we compare the summary achieved by running the greedy algorithm on the reduced set V' of SS with summaries achieved by other algorithms on the original set V. We use the feature based submodular function [29] $f(S) = \sum_{u\in\mathcal{U}}\sqrt{c_u(S)}$ as our objective, where $\mathcal{U}$ is a set of features, and $c_u(S) = \sum_{v\in S}\omega_{v,u}$ is a modular score ($\omega_{v,u}$ is the affinity of element v to feature u). This function typically achieves good performance on summarization tasks.

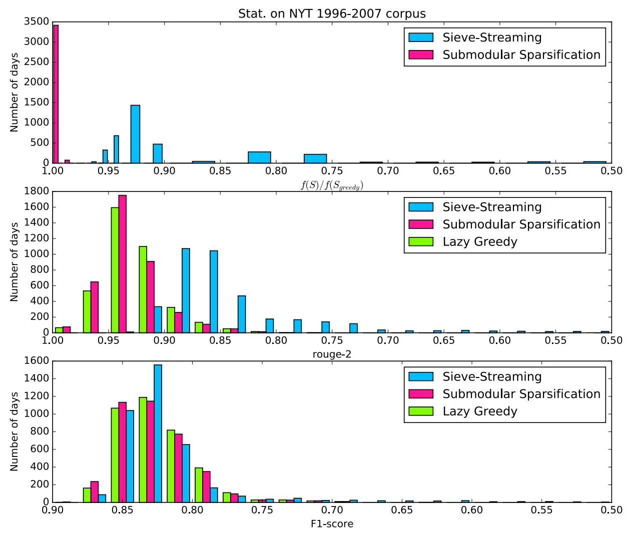

Figure 3: Statistics of relative utility $f(S)/f(S_{greedy})$, ROUGE-2 score and F1-score on daily news summarization results of 3823 days' news from New York Times corpus between 1996-2007.

Our baseline algorithms are the lazy greedy approach [20] (which has identical output as greedy but is faster) and the "sieve-streaming" [1] approach for streaming submodular maximization, which has low memory requirements as it takes one pass over the data. We set $r = 8$ and $1 - 1/\sqrt{c} = 1 - \sqrt{2}/4 \approx 64.6\%$ in Algorithm 1.

4.1 Empirical Study on News

An empirical study is conducted on a ground set containing sentences from all NYT articles on a randomly selected date from the NYTs annotated corpus 1996-2007 (https://catalog.ldc.upenn.edu/LDC2008T19). Figure 1 shows how $f(S)$ and time cost varies when we change n. The budget size k of the summary set to the number of sentences in a human generated summary. The number of trials in sieve-streaming is 50, leading to memory requirement of $50k$. The utility curve of SS overlaps that of lazy greedy, while its time cost is much less and increases more slowly than that of lazy greedy. Sieve-streaming performs much worse than SS in terms of utility (a 10% smaller utility usually leads to substantial decline on summarization performance, please refer to Figure 3), and its time cost is only slightly less (this is because it quickly saturates by selecting k elements after passing a few ($\ll |V|$) elements). Figure 2 shows how relative utility $f(S)/f(S_{greedy})$ (S_{greedy} is the greedy solution) and SS time cost vary with the size of the reduced set V'. SS quickly reaches a $f(S) = 0.97f(S_{greedy})$ once the size exceeds 300, while its computational cost increases slowly.

4.2 News Summarization

We conduct summarization experiments on two large news corpora, the NYTs annotated corpus 1996-2007, and the DUC 2001 corpus (http://www-nlpir.nist.gov/

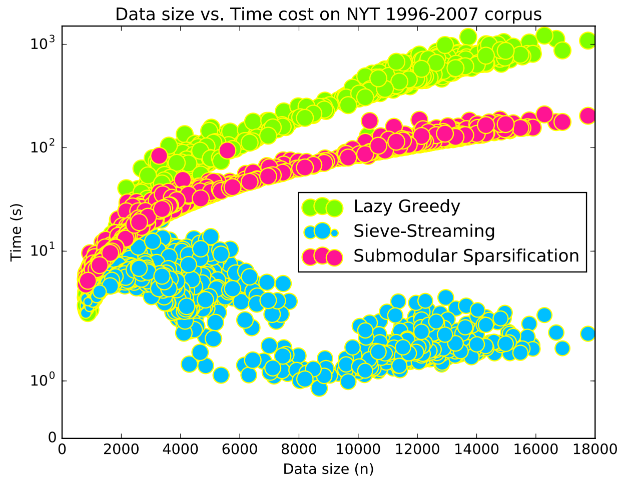

Figure 4: Size of data n vs. time cost on daily news summarization results of 3823 days' news from New York Times corpus between 1996-2007. The area of each circle is proportional to the relative utility $f(S)/f(S_{greedy})$.

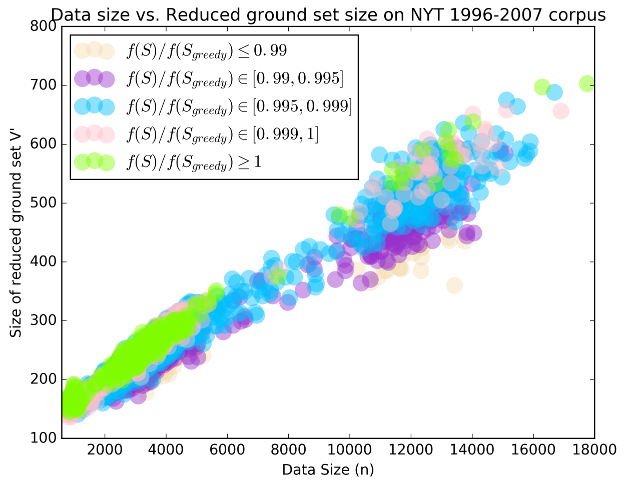

Figure 5: Scatter plot of relative utility $f(S)/f(S_{greedy})$ achieved by submodular sparsification on the 3823 days' news with the corresponding size of ground set V and the size of reduced set V'. Each point corresponds to one day.

projects/duc). The first dataset includes articles published in the NYTs over 3823 days from 1996-2007. We collect the sentences in articles associated with human generated summaries as the ground set V (with sizes varying from 2000 to 20000), and extract their TFIDF features to build $f(S)$. We concatenate the sentences from all human generated summaries for the same date as a reference summary. We compare the machine generated summaries produced by different methods with the reference summary by ROUGE-2 [18] (recall on 2-grams) and ROUGE-2 F1-score (F1-measure based on recall and precision on 2-grams).

We also compare their relative utility. As before, sieve-streaming has memory set at $50k$. The statistics over 3823 days are shown in Figure 3. SS has a relative utility of ≥ 0.99 on most days, while sieve-streaming is mostly in the $[0.92, 0.93]$ region. Both the ROUGE-2 and F1 score

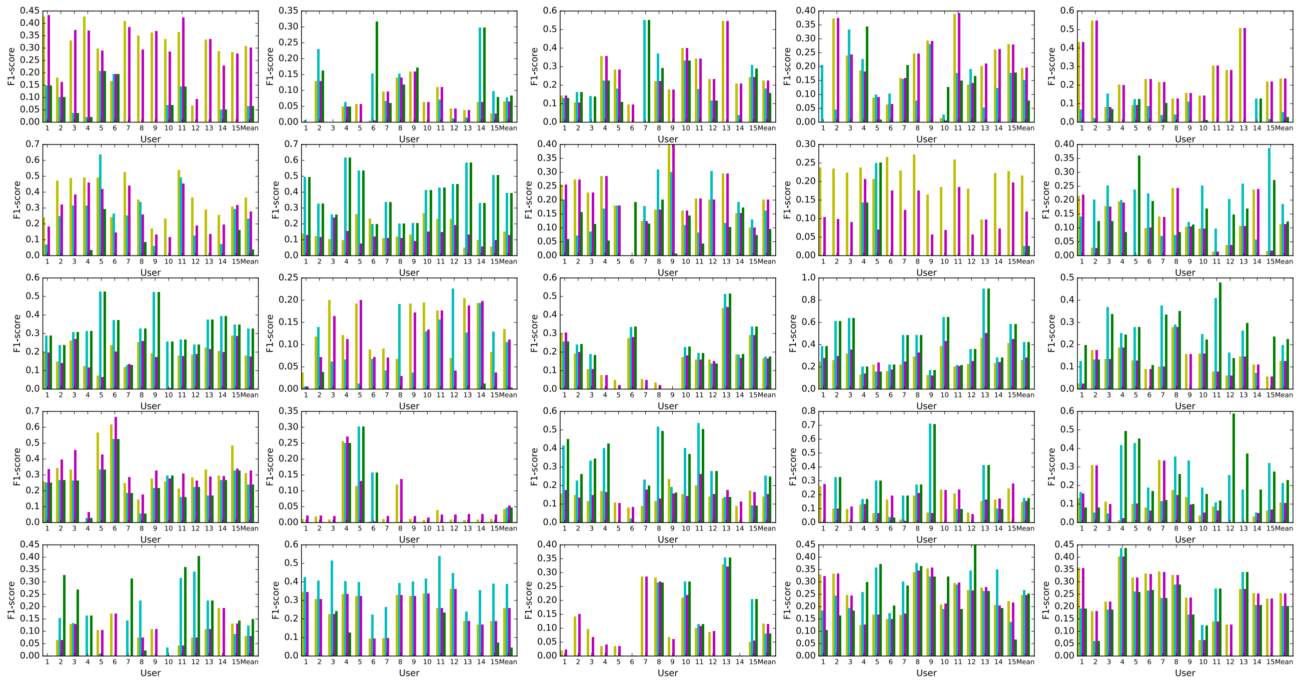

Figure 6: F1-score of the summaries generated by greedy (yellow bar), sieve-streaming (cyan bar), SS (magenta bar) and the first 15% frames (green bar) comparing to reference summaries from 15 users on 25 videos from SumMe dataset. Each plot associates with a video.

of SS are better than sieve-streaming, and even outperform greedy a bit. This may be because SS removes many of the elements on which greedy might become trapped in some local sub-optimal region.

Figure 4 shows the number n of sentences per day and the corresponding time cost of each algorithm. The area of each circle is proportional to relative utility. We use a log scale time axis for a wider dynamic range. SS reduces computation over lazy greedy especially when n is large. Sieve-streaming's time cost decreases when $n \geq 6000$, but its relative utility is reduced due to the aforementioned early saturating. Figure 5 shows the distribution of relative utility achieved by SS with different data sizes n and reduced ground set sizes over 3823 different days. The relative utility of SS is ≥ 0.99 on most days, and even ≥ 1 when $n \leq 6000$. This indicates that summarization on the reduced set V' achieved by SS can even occasionally outperform that on the original ground set V.

4.3 Video Summarization

We apply lazy greedy, sieve-streaming, and SS to 25 videos from dataset SumMe [13] (`http://www.vision.ee.ethz.ch/~gyglim/vsum/`). Each video has 1000 $\sim$ 10000 frames as given in Table 2 [30]. The results are given in [30]. The greedy algorithm on the SS-reduced ground set consistently approaches or outperforms lazy greedy on recall and F1-score, while the time cost is much smaller and a large fraction of frames may be removed.

We resize each frame to a 180×360 image, and extract features from two standard image descriptors, i.e., a pyramid of HoG (pHoG) [5] to delineate local and global shape, and GIST [26] to capture global scene. The 2728 pHoG features are achieved over a four-level pyramid using 8 bins with angle of 360 degrees. The 256 GIST features are obtained by using 4×4 blocks and 8 orientation per scale. We concatenate them to form a 2984-dimensional feature vector for each frame to build $f(\cdot)$. Each algorithm selects 15% of all frames as summary set, i.e., $k = 0.15|V|$. Sieve-streaming holds a memory of $10k$ frames.

We compare the summaries generated by the three algorithms with the ones produced by the ground truth and 15 users. Each user was asked to select a subset of frames as summary, and ground truth score of each frame is given by voting from all 15 users. For each video, we compare the generated summary with the reference summary composed of the top p frames with the largest ground truth scores for different p, and the user summary from different users.

In particular, we report F1-score and recall for comparison to ground truth score generated summaries in Figure 9 and Figure 10 [30]. We report F1-score and recall for comparison to user summaries in Figure 6 and Figure 11 [30]. In each plot for each video, we also report the average F1-score and average recall over all 15 users.

SS consistently approaches or outperforms lazy greedy, while the time cost is much smaller according to Table 2 [30]. Although on a few videos sieve-streaming achieves the best F1-score, in these cases its generated summaries are trivially dominated by the first 15% frames as shown in Figure 6.

References

[1] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *SIGKDD*, pages 671–680, 2014.

[2] Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *SODA*, pages 1497–1514, 2014.

[3] Rafael Barbosa, Alina Ene, Huy Nguyen, and Justin Ward. The power of randomization: Distributed submodular maximization on massive datasets. In *ICML*, pages 1236–1244, 2015.

[4] Mohammadhossein Bateni, Mohammadtaghi Hajiaghayi, and Morteza Zadimoghaddam. Submodular secretary problem and extensions. *ACM Trans. Algorithms*, 9(4):32:1–32:23, 2013.

[5] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Representing shape with a spatial pyramid kernel. In *ACM International Conference on Image and Video Retrieval*, pages 401–408, 2007.

[6] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *FOCS*, pages 649–658, 2012.

[7] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *SODA*, pages 1433–1452, 2014.

[8] Niv Buchbinder, Moran Feldman, and Roy Schwartz. Online submodular maximization with preemption. In *SODA*, pages 1202–1216, 2015.

[9] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. *arXiv:1504.08024*, 2015.

[10] Dan Feldman, Amos Fiat, Micha Sharir, and Danny Segev. Bi-criteria linear-time approximations for generalized k-mean/median/center. In *Proceedings of the Twenty-third Annual Symposium on Computational Geometry*, pages 19–26, 2007.

[11] Satoru Fujishige. *Submodular functions and optimization*. Annals of discrete mathematics. Elsevier, 2005.

[12] Ryan Gomes and Andreas Krause. Budgeted nonparametric learning from data streams. In *ICML*, 2010.

[13] Michael Gygli, Helmut Grabner, Hayko Riemenschneider, and Luc Van Gool. Creating summaries from user videos. In *ECCV*, 2014.

[14] Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S. Mirrokni. Composable core-sets for diversity and coverage maximization. In *PODS*, pages 100–108, 2014.

[15] Rishabh Iyer, Stefanie Jegelka, and Jeff Bilmes. Curvature and optimal algorithms for learning and minimizing submodular functions. In *NIPS*, 2013.

[16] Rishabh Iyer, Stefanie Jegelka, and Jeff A. Bilmes. Fast semidifferential-based submodular function optimization. In *ICML*, 2013.

[17] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *SIGKDD*, pages 420–429, 2007.

[18] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, 2004.

[19] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *ACL*, pages 510–520, 2011.

[20] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, volume 7 of *Lecture Notes in Control and Information Sciences*, chapter 27, pages 234–243. 1978.

[21] Vahab Mirrokni and Morteza Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *STOC*, pages 153–162, 2015.

[22] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. In *AAAI*, pages 1812–1818, 2015.

[23] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *NIPS*, pages 2049–2057, 2013.

[24] Baharan Mirzasoleiman, Morteza Zadimoghaddam, and Amin Karbasi. Fast distributed submodular cover: Public-private data summarization. In *NIPS*, pages 3594–3602. 2016.

[25] 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.

[26] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[27] Xinghao Pan, Stefanie Jegelka, Joseph E Gonzalez, Joseph K Bradley, and Michael I Jordan. Parallel double greedy submodular maximization. In *NIPS*, pages 118–126, 2014.

[28] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Fast multi-stage submodular maximization. In *ICML*, 2014.

[29] Kai Wei, Yuzong Liu, Katrin Kirchhoff, Chris D. Bartels, and Jeff A. Bilmes. Submodular subset selection for large-scale speech training data. In *IEEE International Conference on Acoustics, Speech and Signal Processing, (ICASSP) 2014*, pages 3311–3315, 2014.

[30] Tianyi Zhou, Hua Ouyang, Jeff Bilmes, Yi Chang, and Carlos Guestrin. Supplementary material for "scaling submodular maximization via pruned submodularity graphs". In *AISTATS*, 2017.