arXiv:1611.08574v1 [cs.DS] 25 Nov 2016

An Efficient Streaming Algorithm for the Submodular Cover Problem

Ashkan Norouzi-Fard[*1], Abbas Bazzi[†1], Marwa El Halabi[‡2], Ilija Bogunovic[§2], Ya-Ping Hsieh[¶2], and Volkan Cevher[‖2]

[1]Theory of Computation Laboratory 2 (THL2), EPFL
[2]Laboratory for Information and Inference Systems (LIONS), EPFL

November 28, 2016

Abstract

We initiate the study of the classical Submodular Cover (SC) problem in the data streaming model which we refer to as the Streaming Submodular Cover (SSC). We show that any single pass streaming algorithm using sublinear memory in the size of the stream will fail to provide any non-trivial approximation guarantees for SSC. Hence, we consider a relaxed version of SSC, where we only seek to find a *partial* cover.

We design the first *Efficient bicriteria Submodular Cover Streaming* (ESC-Streaming) algorithm for this problem, and provide theoretical guarantees for its performance supported by numerical evidence. Our algorithm finds solutions that are competitive with the near-optimal offline greedy algorithm despite requiring only a *single* pass over the data stream. In our numerical experiments, we evaluate the performance of ESC-Streaming on active set selection and large-scale graph cover problems.

Keywords: Submodular Cover, Streaming Algorithms.

[*]email: ashkan.norouzifard@epfl.ch
[†]email: abbas.bazzi@epfl.ch. The first two authors contributed equally to this work.
[‡]email:marwa.elhalabi@epfl.ch
[§]email: ilija.bogunovic@epfl.ch
[¶]email: ya-ping.hsieh@epfl.ch
[‖]email: volkan.cevher@epfl.ch

1 Introduction

We consider the *Streaming Submodular Cover (SSC)* problem, where we seek to find the smallest subset that achieves a certain utility, as measured by a monotone submodular function. The data is assumed to arrive in an arbitrary order and the goal is to minimize the number of passes over the whole dataset while using a memory that is as small as possible.

The motivation behind studying SSC is that many real-world applications can be modeled as cover problems, where we need to select a small subset of data points such that they maximize a particular utility criterion. Often, the quality criterion can be captured by a utility function that satisfies submodularity [27, 16, 15], an intuitive notion of diminishing returns. Despite the fact that the standard *Submodular Cover* (SC) problem is extensively studied and very well-understood, all the proposed algorithms in the literature heavily rely on having access to whole ground set during their execution. However, in many real-world applications, this assumption does not hold. For instance, when the dataset is being generated on the fly or is too large to fit in memory, having access to the whole ground set may not be feasible. Similarly, depending on the application, we may have some restrictions on how we can access the data. Namely, it could be that random access to the data is simply not possible, or we might be restricted to only accessing a small fraction of it. In all such scenarios, the optimization needs to be done on the fly.

The SC problem is first considered by Wolsey [28], who shows that a simple greedy algorithm yields a logarithmic factor approximation. This algorithm performs well in practice and usually returns solutions that are near-optimal. Moreover, improving on its theoretical approximation guarantee is not possible under some natural complexity theoretic assumptions [12, 10]. However, such an offline greedy approach is impractical for the SSC, since it requires an infeasible number of passes over the stream.

1.1 Our Contribution

In this work, we rigorously show that achieving any *non-trivial* approximation of SSC with a single pass over the data stream, while using a *reasonable* amount of memory, is *not* possible. More generally, we establish an uncontidional lower bound on the trade-off between the memory and approximation ratio for any p-pass streaming algorithm solving the SSC problem.

Hence, we consider instead a relaxed version of SSC, where we only seek to achieve a fraction $(1-\varepsilon)$ of the specified utility. We develop the first *Efficient bicriteria Submodular Cover Streaming* (ESC-Streaming) algorithm. ESC-Streaming is simple, easy to implement, and memory as well time efficient. It returns solutions that are competitive with the near-optimal offline greedy algorithm. It requires only a single pass over the data in arbitrary order, and provides for any $\varepsilon > 0$, a $2/\varepsilon$-approximation to the optimal solution, while achieving a $(1-\varepsilon)$ fraction of the specified utility. In our experiments, we test the performance of ESC-Streaming on active set selection in materials science and graph cover problems. In the latter, we consider a graph dataset that consists of more than 787 million nodes and 47.6 billion edges.

1.2 Related work

Submodular optimization has attracted a lot of interest in machine learning, data mining, and theoretical computer science. Faced with streaming and massive data, the traditional (offline) greedy approaches fail. One popular approach to deal with the challenge of the data deluge is to adopt streaming or distributed perspective. Several submodular optimization problems have been studied so far under these two settings [25, 11, 9, 2, 20, 8, 18, 7, 14, 1]. In the streaming setting,

the goal is to find nearly optimal solutions, with a minimal number of passes over the data stream, memory requirement, and computational cost (measured in terms of oracle queries).

A related streaming problem to SSC was investigated by Badanidiyuru et al. [2], where the authors studied the streaming Submodular Maximization (SM) problem *subject to a cardinality constraint*. In their setting, given a budget k, the goal is to pick at most k elements that achieve the largest possible utility. Whereas for the SC problem, given a utility Q, the goal is to find the minimum number of elements that can achieve it. In the offline setting of cadinality constrained SM, the greedy algorithm returns a solution that is $(1 - 1/e)$ away from the optimal value [21], which is known to be the best solution that one can obtain efficiently [22]. In the streaming setting, Badanidiyuru et al. [2] designed an elegant single pass $(1/2 - \varepsilon)$-approximation algorithm that requires only $O((k \log k)/\varepsilon)$ memory. More general constraints for SM have also been studied in the streaming setting, e.g., in [8].

Moreover, the *Streaming Set Cover* problem, which is a special case of the SSC problem is extensively studied [25, 11, 9, 7, 14, 1]. In this special case, the elements in the data stream are m subsets of a universe $\mathcal{X}$ of size n, and the goal is to find the minimum number of sets k^* that can cover all the elements in the universe $\mathcal{X}$. The study of the *Streaming Set Cover* problem is mainly focused on the *semi-streaming* model, where the memory is restricted to $\widetilde{O}(n)$[1]. This regime is first investigated by Saha and Getoor [25], who designed a $O(\log n)$-pass, $O(\log n)$-approximation algorithm that uses $\widetilde{O}(n)$ space. Emek and Rosén [11] show that if one restricts the streaming algorithm to perform only one pass over the data stream, then the best possible approximation guarantee is $O(\sqrt{n})$. This lower bound holds even for randomized algorithms. They also designed a deterministic greedy algorithm that matches this approximation guarantee. By relaxing the single pass constraint, Chakrabarti and Wirth [7] designed a p-pass semi-streaming $(p+1)n^{1/(p+1)}$-approximation algorithm, and proved that this is essentially tight up to a factor of $(p+1)^3$.

***Partial* streaming submodular optimization.** The *Streaming Set Cover* has also been studied from a bicriteria perspective, where one settles for solutions that only cover a $(1 - \varepsilon)$-fraction of the universe. Building on the work of [11], the authors in [7] designed a semi-streaming p-pass streaming algorithm that achieves a $(1 - \varepsilon, \delta(n, \varepsilon))$-approximation, where $\delta(n, \varepsilon) = \min\{8p\varepsilon^{1/p}, (8p + 1)n^{1/(p+1)}\}$. They also provided a lower bound that matches their approximation ratio up to a factor of $\Theta(p^3)$.

Distributed submodular optimization. Mirzasoleiman et al. [20] consider the SC problem in the distributed setting, where they design an efficient algorithm whose solution is close to that of the offline greedy algorithm. Moreover, they study the trade-off between the communication cost and the number of rounds to obtain such a solution.

To the best of our knowledge, no other works have studied the general SSC problem. We propose the first efficient algorithm ESC-Streaming that approximately solves this problem with tight guarantees.

2 Problem Statement

Preliminaries. We assume that we are given a utility function $f : 2^V \mapsto \mathbb{R}^+$ that measures the quality of a given subset $S \subseteq V$, where $V = \{e_1, \cdots, e_m\}$ is the ground set. The marginal gain

[1]The $\widetilde{O}$ notation is used to hide poly-log factors, i.e., $\widetilde{O}(n) := O(n \text{ poly}\{\log n, \log m\})$

associated with any given element $e \in V$ with respect to some set $S \subseteq V$, is defined as follows

$$\Delta_f(e|S) := \Delta(e|S) = f(S \cup \{e\}) - f(S).$$

In this work, we focus on *normalized*, *monotone*, *submodular* utility functions f, where f is referred to be:

1. **submodular** if for all S, T, such that $S \subseteq T$, and for all $e \in V \setminus T$, $\Delta(e|S) \geqslant \Delta(e|T)$.
2. **monotone** if for all S, T such that $S \subseteq T \subseteq V$, we have $f(S) \leqslant f(T)$.
3. **normalized** if $f(\emptyset) = 0$.

In the standard *Submodular Cover* (SC) problem, the goal is to find the smallest subset $S \subseteq V$ that satisfies a certain utility Q, i.e.,

$$\min_{S \subseteq V} |S| \text{ s.t. } f(S) \geqslant Q. \tag{SC}$$

Hardness results. The SC problem is known to be NP-Hard. A simple greedy strategy [28] that in each round selects the element with the highest marginal gain until Q is reached, returns a solution of size at most $H(\max_e f(\{e\}))k^*$, where k^* is the size of the optimal solution set S^*.[2] Moreover, Feige [12] proved that this is the best possible approximation guarantee unless NP $\subseteq$ DTIME $\left(n^{O(\log\log n)}\right)$. This was recently improved to an NP-hardness result by Dinur and Steurer [10].

Streaming Submodular Cover (SSC). In the streaming setting, the main challenge is to solve the SC problem while maintaining a small memory and without performing a large number of passes over the data stream. We use m to denote the size of the data stream. Our first result states that any single pass streaming algorithm with an approximation ratio better than $m/2$, must use at least $\Omega(m)$ memory. Hence, for large datasets, if we restrict ourselves to a single pass streaming algorithm with sublinear memory $o(m)$, we cannot obtain *any* non-trivial approximation of the SSC problem (cf., Theorem 4.1 in Section 4). To obtain non-trivial and feasible guarantees, we need to relax the coverage constraint in SC. Thus, we instead solve the *Streaming Bicriteria Submodular Cover* (SBSC) defined as follows:

Definition 2.1. Given $\varepsilon \in (0, 1)$ and $\delta \geqslant 1$, an algorithm is said to be a $(1 - \varepsilon, \delta)$-bicriteria approximation algorithm for the SBSC problem if for any Submodular Cover instance with utility Q and optimal set size k^*, the algorithm returns a solution S such that

$$f(S) \geqslant (1 - \varepsilon)Q \quad \text{and} \quad |S| \leqslant \delta k^* . \tag{2.1}$$

3 An efficient streaming submodular cover algorithm

ESC-Streaming algorithm. The first phase of our algorithm is described in Algorithm 1. The algorithm receives as input a parameter M representing the size of the allowed memory. The discussion of the role of this parameter is postponed to Section 4. The algorithm keeps $t + 1 = \log(M/2) + 1$ *representative* sets. Each representative set S_j ($j = 0, .., t$) has size at most 2^j, and has a corresponding threshold value $Q/2^j$. Once a new element e arrives in the stream, it is added to all

[2] Here, $H(x)$ is the x-th harmonic number and is bounded by $H(x) \leqslant 1 + \ln x$.

the representative sets that are not fully populated, and for which the element's marginal gain is above the corresponding threshold, i.e., $\Delta(e|S_j) \geqslant \frac{Q}{2^j}$. This phase of the algorithm requires only one pass over the data stream. The running time of the first phase of the algorithm is $O(\log(M))$ for every element of the stream, since the per-element computational cost is $O(\log(M))$ oracle calls.

In the second phase (i.e., Algorithm 2), given a feasible $\tilde{\varepsilon}$, the algorithm finds the smallest set S_i among the stored sets, such that $f(S_i) \geqslant (1-\tilde{\varepsilon})Q$. For any query, the running time of the second phase is $O(\log\log(M))$. Note that after one-pass over the stream, we have no limitation on the number of the queries that we can answer, i.e., we do not need another pass over the stream. Moreover, this phase does not require any oracle calls, and its total memory usage is at most M.

Algorithm 1 ESC-Streaming Algorithm - Picking representative set

$t = \log(M/2)$

$S_0 = S_1 = ... = S_t = \emptyset$
for $i = 1, \cdots, m$ **do**
  Let e be the next element in the stream
  for $j = 0, \cdots, t$ **do**
    if $\Delta(e|S_j) \geqslant \frac{Q}{2^j}$ and $|S_j| \leqslant 2^j$ **then**
      $S_j \leftarrow S_j \cup e$
    end if
  end for
end for

Algorithm 2 ESC-Streaming Algorithm - Responding to the queries

Given value $\tilde{\varepsilon}$, perform the following steps

Run a binary search on the $S_0, ..., S_t$
Return the smallest set S_i such that $f(S_i) \geqslant (1-\tilde{\varepsilon})Q$
If no such set exists, Return "Assumption Violated"

In the following section, we analyze ESC-Streaming and prove that it is an $(1-\tilde{\varepsilon}, 2/\tilde{\varepsilon})$-bicriteria approximation algorithm for SSC. Formally we prove the following:

Theorem 3.1. *For any given instance of SSC problem, and any values $M, \tilde{\varepsilon}$, such that $k^*/\tilde{\varepsilon} \leqslant M$, where k^* is size optimal solution to SSC, ESC-Streaming algorithm returns a $(1-\tilde{\varepsilon}, 2/\tilde{\varepsilon})$-approximation solution.*

4 Theoretical Bounds

Lower Bound. We start by establishing a lower bound on the tradeoff between the memory requirement and the approximation ratio of any p-pass streaming algorithm solving the SSC problem.

Theorem 4.1. *For any number of passes p and any stream size m, a p-pass streaming algorithm that, with probability at least $2/3$, approximates the submodular cover problem to a factor smaller than $\frac{m^{\frac{1}{p}}}{p+1}$, must use a memory of size at least $\Omega\left(\frac{m^{\frac{1}{p}}}{p(p+1)^2}\right)$.*

We defer the proof of this theorem to Section 5. Note that for $p = 1$, Theorem 4.1 states that any one-pass streaming algorithm with an approximation ratio better than $m/2$ requires at

least $\Omega(m)$ memory. Hence, for large datasets, Theorem 4.1 rules out *any* approximation of the streaming submodular cover problem, if we restrict ourselves to a one-pass streaming algorithm with sublinear memory $o(m)$. This result motivates the study of the *Streaming Bicriterion Submodular Cover* (SBSC) problem as in Definition 2.1.

Main result and discussion. Refining the analysis of the greedy algorithm for SC [28], where we stop once we have achieved a utility of $(1-\varepsilon)Q$, yields that the number of elements that we pick is at most $k^* \ln(1/\varepsilon)$. This yields a *tight* $(1-\varepsilon, \ln(1/\varepsilon))$-bicriteria approximation algorithm for the *Bicriteria Submoludar Cover* (BSC) problem. One can turn this bicriteria algorithm into an $(1-\varepsilon, \ln(1/\varepsilon))$-bicriteria algorithm for SBSC, at the cost of doing $k^* \ln(1/\varepsilon)$ passes over the data stream which may be infeasible for some applications. Moreover, this requires $mk^* \ln\left(\frac{1}{\varepsilon}\right)$ oracle calls, which may be infeasible for large datasets.

To circumvent these issues, it is natural to parametrize our algorithm by a user defined memory budget M that the streaming algorithm is allowed to use. Assuming, for some $0 < \varepsilon \leqslant e^{-1}$, that the $(1-\varepsilon, \ln(1/\varepsilon))$-bicriteria solution given by the *offline* greedy algorithm's fits in a memory of $M/2$ for the BSC variant of the problem, then our algorithm (ESC-Streaming) is guaranteed to return a $(1 - 1/\ln(1/\varepsilon), 2\ln(1/\varepsilon))$-bicriteria solution for the SBSC problem, while using at most M memory. Hence in only one pass over the data stream, ESC-Streaming returns solutions guaranteed to cover, for small values of ε, almost the same fraction of the utility as the greedy solution, loosing only a factor of two in the worst case solution size. Moreover, the number of oracle calls needed by ESC-Streaming is only $m \log M$, which for $M = 2k^* \ln(1/\varepsilon)$ is bounded by

$$m \log M = \underbrace{m \log(2k^* \ln(1/\varepsilon))}_{\text{oracle calls by ESC-Streaming algorithm}} \ll \underbrace{mk^* \ln\left(\frac{1}{\varepsilon}\right)}_{\text{oracle calls by greedy}},$$

which is more than a factor $k^*/\log(k^*)$ smaller than the greedy algorithm. This enables ESC-Streaming algorithm to perform much faster than the offline greedy algorithm. Another feature of ESC-Streaming is that it performs a single pass over the data stream, and after this unique pass, we are able to query a $(1 - 1/\ln(1/\varepsilon'), 2\ln(1/\varepsilon'))$-bicriteria solution for any $\varepsilon \leqslant \varepsilon' \leqslant e^{-1}$, without any additonal oracle calls. Whenever the above inequality does not hold, ESC-Streaming returns "Assumption Violated". More precisely, we prove the following theorem:

Theorem 4.2. *For any given instance of SSC problem, and any values M, ε, such that $2k^* \ln 1/\varepsilon \leqslant M$, where k^* is the optimal solution size, ESC-Streaming algorithm returns a $(1 - 1/(\ln 1/\varepsilon), 2\ln 1/\varepsilon)$-approximation solution.*

Proof. Choose j such that $\frac{2^j}{2\ln 1/\varepsilon} \leqslant k^* < \frac{2^{j+1}}{2\ln 1/\varepsilon}$. We first show that the number of elements in S_j is at most $2k^* \ln 1/\varepsilon$ and $f(S_j) \geqslant (1 - \frac{1}{\ln 1/\varepsilon})Q$. We know that $|S_j| \leqslant 2^j$, and hence

$$|S_j| \leqslant 2^j \leqslant 2k^* \ln 1/\varepsilon$$

If $|S_j| = 2^j$ then we get $f(S_j) \geqslant Q$. Now assume that $|S_j| < 2^j$ and let S^* be the optimum solution, so $|S^*| = k^*$ and $f(S^*) \geqslant Q$. By monotonicity, we get that $f(S^* \cup S_j) \geqslant Q$. Let $S^* \backslash S_j = \{w_1, ...w_d\}$. We know that the marginal gain that w_i gives to a subset of S_j is less than $\frac{Q}{2^j}$ for all $1 \leqslant i \leqslant d$. By submodularity, we get that $\Delta(w_i|S_j) < \frac{Q}{2^j}$ for all $1 \leqslant i \leqslant d$. Therefore,

$$f(S^* \cup S_j) - f(S_j) \leqslant \sum_{i=1}^{d} \Delta(w_i|S_j) < d\frac{Q}{2^j} \leqslant d\frac{Q}{k^* \ln 1/\varepsilon} \leqslant \frac{Q}{\ln 1/\varepsilon},$$

which gives

$$f(S^* \cup S_j) - f(S_j) \leqslant \frac{Q}{\ln 1/\varepsilon} \to f(S_j) \geqslant Q\left(1 - \frac{1}{\ln 1/\varepsilon}\right).$$

Hence the set S_j is indeed a good candidate solution. Therefore, the algorithm does not return any solution, only if the assumption is not satisfied, i.e., $k^* \ln 1/\varepsilon > M$. Otherwise, if the algorithm returns a solution S_{j_1}, such that $j_1 < j$, then S_{j_1} must satisfy $f(S_{j_1}) \geqslant (1 - \frac{1}{\ln 1/\varepsilon})Q$, and it contains fewer elements than 2^j, then it also satisfies size and utility guarantees. □

Remarks. Note that in Algorithm 1, we can replace the constant 2 by another choice of the constant $1 < \alpha \leqslant 2$. The representative sets are changed accordingly to α^j, and $t = \log_\alpha(M/\alpha)$. Varying α provides a trade-off between memory and solution size guarantee. More precisely, for any $1 < \alpha \leqslant 2$, ESC-Streaming achieves a $(1 - \frac{1}{\ln 1/\varepsilon}, \alpha \ln 1/\varepsilon)$-approximation guarantee, for instances of SSC where $\alpha k^* \ln 1/\varepsilon \leqslant M$. However, the improvement in the size approximation guarantee, comes at the cost of increased memory usage $\frac{M-1}{\alpha-1}$, and increased number of oracle calls $m(\log_\alpha(M/\alpha) + 1)$.

Notice that in the statement of Theorem 4.2, the approximation guarantee of ESC-Streaming is given with respect to a memory only large enough to fit the offline greedy algorithm's solution. However, if we allow our memory M to be as large as k^*/ε, then Theorem 3.1 follows immediately for $\tilde{\varepsilon} = 1/\ln(1/\varepsilon)$.

5 Lower Bound

In this section, we prove an unconditional lower bound on any p-pass streaming algorithm that approximates the streaming submodular cover problem, as stated in Theorem 4.1. A common way to prove lower bounds for streaming algorithms is through communication complexity. For our purposes, the suitable communication problem to start with is the Multi-Player Pointer Jumping Problem.

In what follows, we start by defining the Multi-Player Pointer Jumping Problem and stating its known communication complexity hardness results in Section 5.1. We then present in Section 5.2 a reduction from the streaming submodular cover problem to the aforementioned communication problem. The proof of Theorem 4.1 then follows in Section 5.3.

5.1 Multi-Player Pointer Jumping Problem

In the Multi-Player Pointer Jumping Problem, we are given a rooted tree T of depth $\ell \geqslant 1$, where the nodes in the tree are divided into $k = \ell + 1$ layers according to their distance from the root, with the convention that the root is at layer k, and the leaves are at layer 1. For $1 \leqslant i \leqslant k$, we refer to the set of nodes in layer i as $\mathcal{V}_i$. In this problem, denoted $\text{MPJ}_{T,k}$ for a given tree T, we have k players $P_1, \ldots, P_k$ where each player i is *in charge* of the vertices in layer i. An input π for $\text{MPJ}_{T,k}$ is divided as follows:

- For each $2 \leqslant i \leqslant k$, and for each $v \in \mathcal{V}_i$, player P_i gets a pointer indicating one outgoing edge from v to one of its children, say u, in layer $i-1$, and we write in this context that $\pi(v) = u$.
- For each $v \in \mathcal{V}_1$, player P_1 gets a bit $b(v) := b_\pi(v) \in \{0, 1\}$.

Note that given π, the tree T_π restricted to the edges indicated in π contains a unique root-to-leaf path, leading to a unique leaf $v_\pi \in \mathcal{V}_1$, and hence the output of this problem given an input π is $\text{MPJ}_{T,k}(\pi) = b(v_\pi) \in \{0, 1\}$.

This problem is interesting from a communication protocol perspective, where the k players get the input π as defined earlier, and broadcast messages in r rounds. In each round, players $P_1, P_2, \ldots, P_k$ each send a message, in that order. In this setting, the message in the last round is a single bit, corresponding to their guess about $\text{MPJ}_{T,k}(\pi)$. The goal in this problem then is to figure out this bit using the minimum amount of communication per round. At each round, we think of the players as writing their messages in the order from P_1 to P_k on a shared blackboard. Formally speaking, an $[r, C, \varepsilon]$-protocol for $\text{MPJ}_{T,k}$ is defined as

- The game is played in r rounds.
- The total number of bits communicated per round is at most C.
- The protocol's output is equal to $\text{MPJ}_{T,k}(\pi)$ with probability at least $1 - \varepsilon$.

We quantify the *complexity* of such communication problems by studying their r-round randomised communication complexity $R^r(\text{MPJ}_{T,k})$ of $\text{MPJ}_{T,k}$ as follows:

$$R^r(\text{MPJ}_{T,k}) = \min\{C : \text{there exists a } [r, C, 1/3]\text{-protocol for } \text{MPJ}_{T,k}\}$$

A result in communication complexity shows that if the number of players is k, then the problem requires a large number of communicated bits if we restrct ourselves to r-rounds protocol for $r < k$. Formally, the authors of [6] prove the following:

Theorem 5.1. *Let T be complete t-ary tree of depth $\ell \geqslant 1$ (and consequently $k = \ell + 1 \geqslant 2$ layers). Then*

$$R^{k-1}(\text{MPJ}_{T,k}) = \Omega\left(\frac{t}{k^2}\right)$$

The $\text{MPJ}_{T,k}$ problem will serve as the starting hard communication problem that we use to prove our streaming lower bounds for the submodular cover problem. In what follows, we will prove a lower bound on the required memory of any p-pass streaming algorithm with good approximation guarantee for the submodular cover problem. To do that, let m be the number of sets that arrive in the stream, and p be the number of passes that the streaming algorithm is allowed to perform. We will show that if a p-pass streaming algorithm can provide an approximation guarantee smaller than $m^{\frac{1}{p}}/p$ for the the set cover problem using a memory less than $\Omega(m^{\frac{1}{p}}/p^3)$, then we can solve the Multi-Player Pointer Jumping Problem on any complete t-ary tree with $p + 1$ player in p rounds while communicating less than $\Omega(t/p^2)$ bits per round, which yields a contradiction to Theorem 5.1. We formalise this in what follows.

5.2 Reduction to Submodular Cover

Let $T := T(t, k)$ be a complete t-ary tree with k layers over the vertex set $\mathcal{V}$, where each layer contains vertices $\mathcal{V}_i \subseteq \mathcal{V}$ for $1 \leqslant i \leqslant k$. Note that

$$|\mathcal{V}_i| = t^{k-i} \qquad \forall i \in \{1, 2, \ldots, k\}$$

$$|\mathcal{V}| = \sum_{i=1}^{k} |\mathcal{V}_i| = \sum_{i=1}^{k} t^{k-i} = \frac{t^k - 1}{t - 1}$$

Without loss of generality, assume the children of every node $v \in \bigcup_{i=2}^{k} \mathcal{V}_i$ are ordered, i.e., for every $2 \leqslant i \leqslant k$, for every $v \in \mathcal{V}_i$, and for every $1 \leqslant j \leqslant t$, $c_j(v) \in \mathcal{V}_{i-1}$ denotes the j^{th} child of v in layer $i - 1$.

Given a complete t-ary tree T with k layers, we construct a structure $\mathcal{T}_{t,k,\ell}$ for $\ell \geqslant t$, such that $\mathcal{T}$ has the same tree structure as T, but we additionally have sets associated with each $v \in \mathcal{V}$. To do that, let $\mathcal{X}$ be a universe of elements such that $|\mathcal{X}| = t^{k-1}\ell$. We partition $\mathcal{X}$ into t^{k-1} equal sized (i.e., each of size ℓ) disjoint sets, and assign each such set $\tilde{S}$ to one of the t^{k-1} leafs of T (i.e., the vertices in $\mathcal{V}_1$). For every vertex $v \in \mathcal{V}_1$, we denote its corresponding set by $\tilde{S}_v$. This process defines the sets assigned to leaf vertices. We now recursively define the sets associated with the remaining vertices as follows:

– For every $i = 2, \ldots, k$, and for every $v \in \mathcal{V}_i$, the set $\tilde{S}_v$ associated with the vertex v is the union of the sets corresponding to its children, i.e.,

$$\tilde{S}_v = \bigcup_{j=1}^{t} \tilde{S}_{c_j(v)}$$

Note that with this construction, if we denote by r the root of the tree (i.e., the unique vertex in $\mathcal{V}_k$), then $\tilde{S}_r = \mathcal{X}$. In fact, one can easily verify that for any $i \in \{1, 2, \ldots, k\}$, and any $v \in \mathcal{V}_i$, our construction satisfies that

$$|\tilde{S}_v| = \ell t^{i-1} \tag{5.1}$$

Before we proceed, we record the following easy observation.

Observation 1. For any 2 vertices $v, u \in \mathcal{V}$ such that v is neither the ancestor nor the descendent of v, we have that $\tilde{S}_v \cap \tilde{S}_u = \emptyset$.

Now given an input π for $\mathrm{MPJ}_{T,k}$ over $T := T(t,k)$, we construct an instance $\mathcal{I}(\pi)$ of $\mathrm{SMC}_{t,k,\ell}$ using $\mathcal{T}^{t,k,\ell}$ as follows, where each player out of the k players $P_1, P_2, \ldots, P_k$ gets a collection of sets. Namely,

– Player P_k gets the following sets; let r be the root of the tree, and let $u = \pi(r)$ be the chosen child of v according to π, then P_k gets the set $S_r = \tilde{S}_r \backslash \tilde{S}_u$, and a singleton set for each element $e \in \tilde{S}_u$. It follows from (5.1) that P_k gets $\ell t^{k-2} + 1$ sets.

– For each $2 \leqslant i \leqslant k-1$, player P_i gets the following sets: For each $v \in \mathcal{V}_i$, let $u = \pi(v)$ be the chosen child of v according to π. Then Player P_i gets for each such vertex v the set $S_v = \tilde{S}_v \backslash \tilde{S}_u$. Hence, for $2 \leqslant i \leqslant k-1$, player P_i gets in total t^{k-i} sets.

– Player P_1 gets the following sets: For each $v \in \mathcal{V}_1$, player P_1 gets the set $S_v = \tilde{S}_v$ if $b(v) = 1$, and nothing from this vertex if $b(v) = 0$. Hence P_1 gets at most t^{k-1} sets.

Let $\mathcal{S}_i$ be the family of sets that player P_i gets, and $\mathcal{S}$ be the family of all the sets in our instance, then $m := |\mathcal{S}|$ is at most

$$m \leqslant \underbrace{t^{k-1}}_{P_1} + \underbrace{\sum_{i=2}^{k-1} t^{k-i}}_{P_2, \ldots, P_{k-1}} + \underbrace{\ell t^{k-2} + 1}_{P_k} = \frac{t^k - 1 + \ell t^{k-2}(t-1)}{t-1} \text{ sets.}$$

The submodular function f that we consider for $\mathrm{SMC}_{t,k,\ell}$ is the cover function defined as follows:

$$f(S) = |S| \qquad \forall S \in \mathcal{S}$$

$$f(\tilde{\mathcal{S}}) = |\bigcup_{S \in \tilde{\mathcal{S}}} S| \qquad \forall \tilde{\mathcal{S}} \subseteq \mathcal{S}$$

Then the goal in this submodular cover problem is to find the minimum cardinality family of sets $\mathcal{S}^* \subset \mathcal{S}$ such that

$$f(\mathcal{S}^*) \geqslant \ell t^{k-1} := Q$$

Note that in our construction, we can always achieve the desired value Q, no matter whether $\text{MPJ}_{T,k}(\pi)$ is 0 or 1, as the sets assigned to player P_k are enough to cover $\mathcal{X}$. In other words, one can easily check that $f(\mathcal{S}_k) = \ell t^{k-1} = Q$.

In order to see the intuition behind this construction, recall that in the communication problem $\text{MPJ}_{T,k}$, we were interested in the value of $\text{MPJ}_{T,k}(\pi) = b(v_\pi)$ given an input π. We will show in the following claim how to relate the value of $b(v_\pi)$ to the size of the submodular cover in the instance $\mathcal{I}(\pi)$ of $\text{SMC}_{t,k,\ell}$ that we have constructed.

Claim 1. Let $\text{MPJ}_{T,k}$ be an instance of the Multi-Player Pointer Jumping Problem, and let $\mathcal{I}(\pi)$ be the resulting submodular cover instance of $\text{SMC}_{t,k,\ell}$. Then the following holds:

1. If $b(v_\pi) = 1$, then there exists a family of sets $\mathcal{S}$ such that $|\mathcal{S}| = k$, and $f(\mathcal{S}) = Q$.
2. If $b(v_\pi) = 0$, then any family of sets $\mathcal{S}$ of size less than ℓ will have $f(\mathcal{S}) < Q$.

Proof. Let $v_k, v_{k-1}, \ldots, v_1$ be the unique root-to-leaf path resulting from π.

We start with the first case, i.e., $b(v_\pi) = 1$. In this case, consider the sets $S_{v_k}, S_{v_{k-1}}, \ldots, S_{v_1}$. It follows from the definition of the sets according to π that

$$\begin{array}{lcll} S_{v_k} & = & \tilde{S}_{v_k} \setminus \tilde{S}_{v_{k-1}} & = \mathcal{X} \setminus \tilde{S}_{v_{k-1}} \\ S_{v_{k-1}} & = & \tilde{S}_{v_{k-1}} \setminus \tilde{S}_{v_{k-2}} & \\ \vdots & = & \vdots & \\ S_{v_2} & = & \tilde{S}_{v_2} \setminus \tilde{S}_{v_1} & \end{array}$$

where $\tilde{S}_{v_1} = S_{v_1}$ since $b(v_1) = b(v_\pi) = 1$ in π. It then follows that

$$\bigcup_{i=1}^{k} S_{v_i} = \mathcal{X}$$

and hence for $\mathcal{S} = \left\{S_{v_k}, S_{v_{k-1}}, \ldots, S_{v_1}\right\}$, we get $f(\mathcal{S}) = Q$.

Now for the second case, also consider $v_k, v_{k-1}, \ldots, v_1$ the unique root-to-leaf path resulting from π. In this case, since $b(v_\pi) = b(v_1) = 0$, $S_{v_1} = \emptyset$. Note however that $\tilde{S}_{v_1}$ contained ℓ elements. We claim that for any non-singleton set $S \in \mathcal{S}$, $S \cap \tilde{S}_{v_1} = \emptyset$. To see this, observe that for any vertex v, $S_v \subseteq \tilde{S}_v$, and hence we never add to S_v an element that was not already in $\tilde{S}_v$. Thus, Observation 1 yields that for any $S_u \in \mathcal{S}$ such that u is not an ancestor of v_1, $S_u \cap \tilde{S}_{v_1} = \emptyset$. It remains to show that $S_{v_i} \cap \tilde{S}_{v_1} = \emptyset$ for all $2 \leqslant i \leqslant k$, i.e., for the set associated with the vertices on the unique root-to-leaf path. But this easily follows from the *child-exclusion nature* of our construction , that yields that $\tilde{S}_{v_1}$ does is disjoint from any set on the path leading from the root to v_1.

This says that in order to cover the elements of $\tilde{S}_{v_1}$, we need to include all the required singletons (recall that these singletons must be in $\mathcal{S}_k$), and hence since $|\tilde{S}_{v_1}| = \ell$, we need to include at least ℓ singleton sets, which yields the second part of the claim. □

In our argument for the second case, we assumed that we can cover $\mathcal{X} \backslash S_{v_1}$ *for free*. One can refine the analysis to show that in this case, any family of sets $\tilde{\mathcal{S}}$ that achieves $f(\tilde{S}) \geqslant Q$ must have a size of at least $\ell + k - 1$. For our purposes, we think of $\ell >> k$, and hence a gap between k and ℓ is enough for the main result of this section.

5.3 Lower bound for the Submodular Cover Problem

We are now ready to prove the main theorem of this section.

Theorem 5.2. *For any number of passes p and any stream size m, a p-pass streaming algorithm that, with probability at least 2/3, approximates the submodular cover problem to a factor smaller than $\frac{m^{\frac{1}{p}}}{p+1}$ must use at least $\Omega\left(\frac{m^{\frac{1}{p}}}{p(p+1)^2}\right)$.*

Proof. Let $\text{MPJ}_{T,p+1}$ be an instance of the Multi-Player Pointer Jumping Problem over a complete t-ary tree T with $p+1$ levels, and let π be the input to the problem spread amongst the $p+1$ players $P_1, P_2, \ldots, P_{p+1}$ as discussed in Section 5.1. We now construct the instance $\mathcal{I}(\pi)$ of $\text{SMC}_{t,p+1,\ell}$ for some integer $\ell \geqslant t$ over m sets as described in Section 5.2. Recall that

$$m \leqslant \frac{t^k - 1 + \ell t^{k-2}(t-1)}{t-1}$$

and $k = p + 1$ in our case.

It follows from Claim 1 that distinguishing between the case when $b(v_\pi) = 1$ and $b(b_\pi) = 0$ is equivalent to distinguishing whether the minimal set cover $\mathcal{S}^*$ of $\text{SMC}_{t,p+1,\ell}$ is of size at most $p+1$, or at least ℓ.

Consider a p-pass streaming algorithm Alg that approximates $\text{SMC}_{t,p+1,\ell}$ using memory M, to a factor smaller than $\frac{\ell}{p+1}$, where the stream consist of the sets of P_1 followed by those of P_2 and so on. We will use Alg to design the following an $[p, (p+1)M, 1/3]$-protocol Prtcl for $\text{MPJ}_{T,k}$ as follows:

- In each round $i = 1, \ldots, p$, we emulate the i^{th} pass of Alg on the stream; When Alg processes the last set corresponding to P_1 in the stream, the content of the memory is broadcasted to all the players. Then we do the same after Alg finishes P_2's chunk on the stream, and so on up to P_{p+1}, in that order.

Since Alg approximates the size of $\mathcal{S}^*$ for $\mathcal{I}(\pi)$ to a factor smaller than $\frac{\ell}{p+1}$ with probability at least 2/3, then Prtcl outputs $\text{MPJ}_{T,k}(\pi)$ with probability at least 2/3. Recall that the game $\text{MPJ}_{T,p+1}$ is played among $p+1$ players, and we know from Theorem 5.1 that $R^p(\text{MPJ}_{T,p+1}) = \Omega\left(\frac{t}{(p+1)^2}\right)$, hence M must be at least $M = \Omega\left(\frac{t}{p(p+1)^2}\right)$.

It remains to relate the approximation ratio and the required memory size, to the size of the stream m. Recall that:

$$m \leqslant \frac{t^k - 1 + \ell t^{k-2}(t-1)}{t-1}$$

For $\ell = t$, we get that $m \leqslant \frac{2t^k - t^{k-1}}{t-1} \leqslant 2t^{k-1} = 2t^p$, and equivalently $\ell = t \approx m^{1/p}$. Thus we get that any p-pass streaming algorithm that, with probability at least 2/3, approximates the submodular cover problem to a factor smaller than $\frac{m^{\frac{1}{p}}}{p+1}$ must use at least $\Omega\left(\frac{m^{\frac{1}{p}}}{p(p+1)^2}\right)$ memory. □

6 Example Applications

Many real-world problems, such as data summarization [27], image segmentation [16], influence maximization in social networks [15], can be formulated as a submodular cover problem and can benefit from the streaming setting. In this section, we discuss two such concrete applications.

6.1 Active set selection

To scale kernel methods (such as kernel ridge regression, Gaussian processes, etc.) to large data sets, we often rely on active set selection methods [23]. For example, a significant problem with Gaussian process prediction is that it scales as $O(n^3)$. Storing the kernel matrix K and solving the associated linear system is prohibitive when n is large. One way to overcome this is to select a small subset of data while maintaining a certain diversity. A popular approach for active set selection is Informative Vector Machine (IVM) [26], where the goal is to select a set S that maximizes the utility function

$$f(S) = \frac{1}{2} \log \det(I + \sigma^{-2} K_{S,S}), . \tag{6.1}$$

Here, $K_{S,S}$ is the submatrix of K, corresponding to rows/columns indexed by S, and $\sigma > 0$ is a regularization parameter. This utility function is monotone submodular, as shown in [17].

6.2 Graph set cover

In a lot of applications, e.g., influence maximization in social networks [15], community detection in graphs [13], etc., we are interested in selecting a small subset of vertices from a massive graph that "cover" in some sense a large fraction of the graph.
In particular, in section 7, we consider two fundamental set cover problems: Dominating Set and Vertex Cover problems. Given a graph $G(V, E)$ with vertex set V and edge set E, let $\rho(S)$ denote the neighbours of the vertices in S in the graph, and $\delta(S)$ the edges in the graph connect to a vertex in S. The dominating set is the problem of selecting the smallest set that covers the vertex set V, i.e., the corresponding utility is $f(S) = |\rho(S) \cup S|$. The vertex cover is the problem of selecting the smallest set that covers the edge set E, i.e., the corresponding utility is $f(S) = |\delta(S)|$. Both utilities are monotone submodular functions.

7 Experimental Results

We address the following questions in our experiments:

1. How does ESC-Streaming perform in comparison to the offline greedy algorithm, in terms of solution size and speed?
2. How does α influence the trade-off between solution size and speed ?
3. How does ESC-Streaming scale to massive data sets?

We evaluate the performance of ESC-Streaming on real-world data sets with two applications: active set selection and graph set cover problems, described in section 6. For active set selection, we choose a dataset having a size that permits the comparison with the offline greedy algorithm.

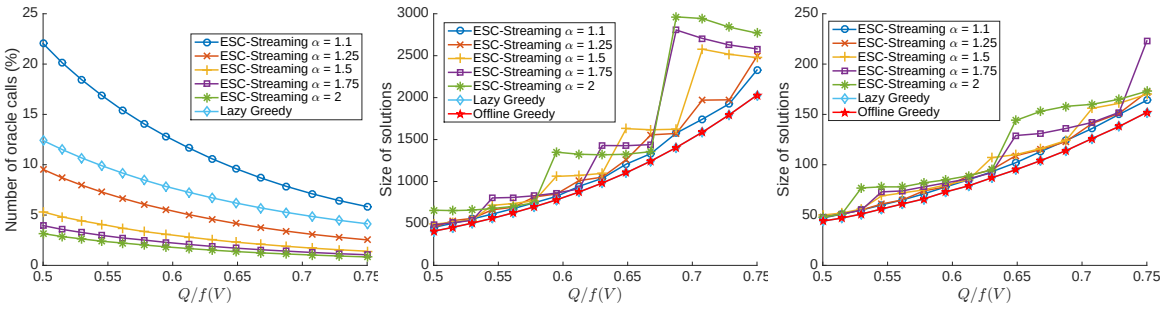

Figure 1: Active set selection of molecules: (Left) Percentage of oracle calls made relative to offline greedy, (Middle) Size of selected sets for $\varepsilon = 0.01$, (Right) Size of selected sets for $\varepsilon = 0.5$.

For graph cover, we run ESC-Streaming on a large graph of 787 million nodes and 47.6 billion edges.

We measure the computational cost in terms of the number of oracle calls, which is independent of the concrete implementation and platform.

7.1 Active Set Selection for Quantum Mechanics

In quantum chemistry, computing certain properties, such as atomization energy of molecules, can be computationally challenging [24]. In this setting, it is of interest to choose a small and diverse training set, from which one can predict the atomization energy (e.g., by using kernel ridge regression) of other molecules.

In this setting, we apply ESC-Streaming on the log-det function defined in Section 6.1 where we use the Gaussian kernel $K_{ij} = \exp(-\frac{\|x_i - x_j\|_2^2}{2h^2})$, and we set the hyperparameters as in [24]: $\sigma = 1, h = 724$. The dataset consists of 7k small organic molecules, each represented by a 276 dimensional vector. We set $M = 2^{15}$ and vary Q from $\frac{f(V)}{2}$ to $\frac{3f(V)}{4}$, and α from 1.1 to 2.

We compare against offline greedy, and its accelerated version with lazy updates (Lazy Greedy)[19]. For all algorithms, we provide a vector of different values of $\tilde{\varepsilon}$ as input, and terminate once the utility $(1 - \tilde{\varepsilon})Q$, corresponding to the smallest $\tilde{\varepsilon}$, is achieved. Below we report the performance for the smallest and largest tested $\tilde{\varepsilon} = 0.01$ and $\tilde{\varepsilon} = 0.5$, respectively.

In Figure 7.1, we show the performance of ESC-Streaming with respect to the offline greedy and lazy greedy, in terms of size of solutions picked and number of oracle calls made. The computational costs of all algorithms are normalized to those of offline greedy.

It can be seen that standard ESC-Streaming, with $\alpha = 2$, always chooses a set at most twice (largest ratio is 2.1089) as large as offline greedy, using at most 3.15% and 25.5% of the number of oracle calls made, respectively, by offline greedy and lazy greedy. As expected, varying the parameter α leads to smaller solutions at the cost of more oracle calls: $\alpha = 1.1$ leads to solutions roughly of the same size as the solutions found by the offline greedy. Note also that choosing larger values α leads to jumps in the solution sets sizes (c.f., 7.1). In particular, varying the required utility Q, even by a small amount, may not be possible to achieve by the current solution's size (α^j) and would require moving to a set larger by at least an α factor (α^{j+1}).

Finally, we remark that even for this small dataset, offline greedy, for largest tested Q, required 1.2×10^7 oracle calls, and took almost 2 days to run on the same machine.

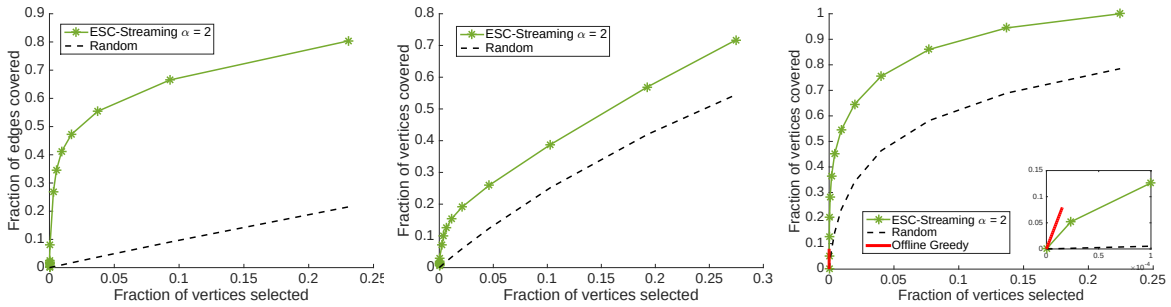

Figure 2: (Left) Vertex cover on "uk-2014" (Middle) Dominating set on "uk-2014" (Right) Dominating set on "Friendster"

7.2 Cover problems on Massive graphs

To assess the scalability of ESC-Streaming, we apply it to the "uk-2014" graph, a large snapshot of the .uk domain taken at the end of 2014 [5, 4, 3]. It consists of 787,801,471 nodes and 47,614,527,250 edges. This graph is sparse, with average degree 60.440, and hence requires large cover solutions. Storing this dataset (i.e., the adjacency list of the graph) on the hard-drive requires more than 190GB of memory.

We solve both the Dominating Set and Vertex Cover problems, whose utility functions are defined in Section 6. For the Dominating Set problem, we set $M = 520$ MB, $\alpha = 2$ and $Q = 0.7|V|$. We run the first phase of ESC-Streaming (c.f., Algorithm 1), then query for different values of $\tilde{\varepsilon}$ between 0 to 1, using Algorithm 2. Similarly, for the Vertex Cover problem, we set $M = 320$ MB, $\alpha = 2$ and $Q = 0.8|E|$. Figure 7.2 shows the performance of ESC-Streaming on both the dominating set and vertex cover problems, in terms of utility achieved, i.e., number vertices/edges covered, for all the feasible $\tilde{\varepsilon}$ values, with respect to the size of the subset of vertices picked.

As a baseline, we compare against a random selection procedure, that picks a random permutation of the vertices and then select any vertex with a non-zero marginal, until it reaches the same partial cover achieved by ESC-Streaming. Note that the offline greedy, even with lazy evaluations, is not applicable here since it does not terminate in a reasonable time, so we omit it from the comparison. Similarly, we do not compare against the Emek–Rosén's algorithm [11], due to its large memory requirement of $n \log m$, which in this case is roughly 20 times bigger than the memory used by ESC-Streaming.

We do significantly better than a random selection, especially on the Vertex Cover problem, which for sparse graphs is more challenging than the Dominating Set problem.

Since running the greedy algorithm on "uk-2014" graph goes beyond our computing infrastructure, we include another instance of the Dominating set cover problem on a smaller graph "Friendster", an online gaming network [29], to compare with offline greedy algorithm. This graph has 65.6 million nodes, and 1.8 billion edges. The memory required by ESC-Streaming is less than 30MB for $\alpha = 2$. We let offline greedy run for 2 days, and gathered data for 2000 greedy iterations. Figure 7.2 (Right) shows that our performance almost matches the greedy solutions we managed to compute.

8 Conclusion

In this paper, we consider the SC problem in the streaming setting, where we select the least number of elements that can achieve a certain utility, measured by a submodular function. We prove that there cannot exist any single pass streaming algorithm that can achieve a non-trivial approximation of SSC, using sublinear memory, if the utility have to be met exactly. Consequently, we develop an efficient approximation algorithm, ESC-Streaming, which finds solution sets, slightly larger than the optimal solution, that partially cover the desired utility. We rigorously analyzed the approximation guarantees of ESC-Streaming, and compared these guarantees against the offline greedy algorithm. We demonstrate the performance of ESC-Streaming on real-world problems. We believe that our algorithm is an important step towards solving streaming and large scale submodular cover problems, which lie at the heart of many modern machine learning applications.

Acknowledgements

We would like to thank Michael Kapralov and Ola Svensson for useful discussions. This work was supported in part by the European Commission under ERC Future Proof, SNF 200021-146750, SNF CRSII2-147633, NCCR Marvel, and ERC Starting Grant 335288-OptApprox.

References

[1] Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. *arXiv preprint arXiv:1603.05715*, 2016. 1, 2

[2] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 671–680. ACM, 2014. 1, 2

[3] Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. BUbiNG: Massive crawling for the masses. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web*, pages 227–228. International World Wide Web Conferences Steering Committee, 2014. 13

[4] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th international conference on World Wide Web*, pages 587–596. ACM Press, 2011. 13

[5] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press. 13

[6] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 641–650. ACM, 2008. 7

[7] Amit Chakrabarti and Tony Wirth. Incidence geometries and the pass complexity of semi-streaming set cover. *arXiv preprint arXiv:1507.04645*, 2015. 1, 2

[8] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *Automata, Languages, and Programming*, pages 318–330. Springer, 2015. 1, 2

[9] Erik D Demaine, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. On streaming and communication complexity of the set cover problem. In *Distributed Computing*, pages 484–498. Springer, 2014. 1, 2

[10] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 624–633, New York, NY, USA, 2014. ACM. 1, 3

[11] Yuval Emek and Adi Rosén. Semi-streaming set cover. In *Automata, Languages, and Programming*, pages 453–464. Springer, 2014. 1, 2, 13

[12] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998. 1, 3

[13] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010. 11

[14] Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. Towards tight bounds for the streaming set cover problem. *arXiv preprint arXiv:1509.00118*, 2015. 1, 2

[15] David Kempe, Jon Kleinberg, and Éva 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. ACM, 2003. 1, 11

[16] Gunhee Kim, Eric P Xing, Li Fei-Fei, and Takeo Kanade. Distributed cosegmentation via submodular optimization on anisotropic diffusion. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 169–176. IEEE, 2011. 1, 11

[17] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 3:19, 2012. 11

[18] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. *ACM Transactions on Parallel Computing*, 2(3):14, 2015. 1

[19] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, pages 234–243. Springer, 1978. 12

[20] Baharan Mirzasoleiman, Amin Karbasi, Ashwinkumar Badanidiyuru, and Andreas Krause. Distributed submodular cover: Succinctly summarizing massive data. In *Advances in Neural Information Processing Systems*, pages 2863–2871, 2015. 1, 2

[21] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978. 2

[22] George L Nemhauser and Leonard A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of operations research*, 3(3):177–188, 1978. 2

[23] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. 11

[24] Matthias Rupp. Machine learning for quantum mechanics in a nutshell. *International Journal of Quantum Chemistry*, 115(16):1058–1073, 2015. 12

[25] Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *SDM*, volume 9, pages 697–708. SIAM, 2009. 1, 2

[26] Matthias Seeger. Greedy forward selection in the informative vector machine. Technical report, Technical report, University of California at Berkeley, 2004. 11

[27] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in Neural Information Processing Systems*, pages 1413–1421, 2014. 1, 11

[28] Laurence A Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982. 1, 3, 5

[29] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12, pages 3:1–3:8, New York, NY, USA, 2012. ACM. 13