Fair and Representative Subset Selection from Data Streams

Yanhao Wang
University of Helsinki
Helsinki, Finland
yanhao.wang@helsinki.fi

Francesco Fabbri*
Pompeu Fabra University & Eurecat
Barcelona, Spain
francesco.fabbri@upf.edu

Michael Mathioudakis
University of Helsinki
Helsinki, Finland
michael.mathioudakis@helsinki.fi

ABSTRACT

We study the problem of extracting a small subset of representative items from a large data stream. In many data mining and machine learning applications such as social network analysis and recommender systems, this problem can be formulated as maximizing a monotone submodular function subject to a cardinality constraint k. In this work, we consider the setting where data items in the stream belong to one of several disjoint groups and investigate the optimization problem with an additional *fairness* constraint that limits selection to a given number of items from each group. We then propose efficient algorithms for the fairness-aware variant of the streaming submodular maximization problem. In particular, we first give a $(\frac{1}{2} - \varepsilon)$-approximation algorithm that requires $O(\frac{1}{\varepsilon} \log \frac{k}{\varepsilon})$ passes over the stream for any constant $\varepsilon > 0$. Moreover, we give a single-pass streaming algorithm that has the same approximation ratio of $(\frac{1}{2} - \varepsilon)$ when unlimited buffer sizes and post-processing time are permitted, and discuss how to adapt it to more practical settings where the buffer sizes are bounded. Finally, we demonstrate the efficiency and effectiveness of our proposed algorithms on two real-world applications, namely *maximum coverage on large graphs* and *personalized recommendation*.

CCS CONCEPTS

• **Information systems → Data stream mining**.

KEYWORDS

algorithmic fairness, approximation algorithm, data summarization, streaming algorithm, submodular maximization

ACM Reference Format:
Yanhao Wang, Francesco Fabbri, and Michael Mathioudakis. 2021. Fair and Representative Subset Selection from Data Streams. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3442381.3449799

1 INTRODUCTION

A crucial task in modern data-driven applications, ranging from influence maximization [23, 35] and recommender systems [32, 34], to nonparametric learning [4, 17] and data summarization [12, 33], is to extract a few representatives from a large dataset. In all aforementioned applications, this task can be formulated as selecting a subset of items to maximize a utility function that quantifies the "representativeness" (or "utility") of the selected subset. Oftentimes, the objective function satisfies *submodularity*, a property of "diminishing returns" such that adding an item to a smaller set always leads to a greater increase in utility than adding it to a larger set. Consequently, maximizing submodular set functions subject to a cardinality constraint (i.e., the size of the selected subset is limited to an integer k) is general enough to model many practical problems in data mining and machine learning. In this work, we adopt the same formulation for representative item selection.

The classic approach to the cardinality-constrained submodular maximization problem is the Greedy algorithm proposed by Nemhauser et al. [31], which achieves an approximation factor of $(1 - \frac{1}{e})$ that is NP-hard to improve [13]. In many real-world scenarios, however, the data become too large to fit in memory or arrive incrementally at a high rate. In such cases, the Greedy algorithm becomes very inefficient because it requires k repeated sequential scans over the whole dataset. Therefore, streaming algorithms for submodular maximization problems have received much attention recently [2, 4, 17, 21, 32]. Typically, these streaming algorithms require only one or a few passes over the dataset, store a small portion of items in memory, and compute a solution more efficiently than the Greedy algorithm at the expense of slightly lower quality.

Despite the extensive studies on streaming submodular maximization, unfortunately, it seems that none of the existing methods consider the *fairness* issue of the subsets extracted from data streams. In fact, recent studies [5, 10, 11, 20] reveal that data summaries automatically generated by algorithms might be biased with respect to sensitive attributes such as gender, race, or ethnicity, and the biases in summaries could be passed to data-driven decision-making processes in education, recruitment, banking, and judiciary systems. Thus, it is necessary to introduce *fairness* constraints into submodular maximization problems so that the selected subset can fairly represent each sensitive attribute in the dataset. Towards this end, we consider that a data stream V comprises l disjoint groups $V_1, V_2, \ldots, V_l$ defined by some sensitive attribute. For example, the groups may correspond to a demographic attribute such as *gender* or *race*. We define the fairness constraint by assigning a cardinality constraint k_i to *each group* V_i and ensuring that $\sum_{i=1}^{l} k_i = k$. Then, our goal is to maximize the submodular objective function under the constraint that the selected subset must contain k_i items from V_i. The fairness constraint as defined above can incorporate different concepts of *fairness* by assigning different values of $k_1, k_2, \ldots, k_l$. For example, one can extract a subset that approximately represents the proportion of each group in the dataset by setting $k_i = \frac{|V_i|}{|V|} \cdot k$. As another example, one can enforce a balanced representation of each group by setting $k_i = \frac{k}{l}$.

*This research was done while Francesco Fabbri worked at the University of Helsinki.

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449799

Theoretically, the above-defined fairness constraint is a case of *partition matroid* constraints [1, 19, 24], and thus the optimization problem can be reduced to maximizing submodular set functions with matroid constraints. It is not surprising that all existing algorithms for submodular maximization with cardinality constraints cannot be directly used for this problem anymore, because their solutions may not satisfy the fairness constraint (i.e., the group-specific cardinality constraints). Nevertheless, a seminal work of Fisher et al. [15] indicates that the Greedy algorithm with minor modifications is $\frac{1}{2}$-approximate for this problem. But it still suffers from efficiency issues in the streaming setting. In addition, the state-of-the-art streaming algorithms [6, 8, 14] for submodular maximization with matroid constraints are only $\frac{1}{4}$-approximate and do not provide solutions of the same quality as the Greedy algorithm efficiently in practice.

In this paper, we investigate the problem of *streaming submodular maximization with fairness constraints*. Our main contributions are summarized as follows.

- We first formally define the *fair submodular maximization* (FSM) problem and show its NP-hardness. We also describe the $\frac{1}{2}$-approximation Greedy algorithm for the FSM problem and discuss why it cannot work efficiently in data streams. (Section 3)
- We propose a multi-pass streaming algorithm MP-FSM for the FSM problem. Theoretically, MP-FSM requires $O(\frac{1}{\varepsilon}\log\frac{k}{\varepsilon})$ passes over the dataset, stores $O(k)$ items in memory, and has an approximation ratio of $(\frac{1}{2}-\varepsilon)$ for any constant $\varepsilon > 0$. (Section 4.1)
- We further propose a single-pass streaming algorithm SP-FSM for the FSM problem, which requires only one pass over the data stream and offers the same approximation ratio as MP-FSM when an unbounded buffer size is permitted. We also discuss how to adapt SP-FSM heuristically to limit the buffer size to $O(k)$. (Sections 4.2 & 4.3)
- Finally, we evaluate the performance of our proposed algorithms against the state-of-the-art methods in two real-world application scenarios, namely *maximum coverage on large graphs* and *personalized recommendation*. The empirical results on several real-world and synthetic datasets demonstrate the efficiency, effectiveness, and scalability of our proposed algorithms. (Section 5)

2 RELATED WORK

There has been a large body of work on submodular optimization for its wide applications in various real-world problems, including influence maximization [23, 37], facility location [27, 28], nonparametric learning [4, 17], and group recommendation [34]. We refer interested readers to [25] for a survey.

The line of research that is the most relevant to this work is *streaming algorithms for submodular maximization*. The seminal work of Fisher, Nemhauser, and Wolsey [15, 31] showed that the Greedy algorithm, which iteratively added an item that maximally increased the utility with k passes over the dataset, gave approximation ratios of $(1-\frac{1}{e})$ and $\frac{1}{2}$ for maximizing monotone submodular functions with cardinality and matroid constraints, respectively. Then, a series of recent studies [4, 17, 21, 26] proposed multi- or single-pass streaming algorithms for maximizing monotone submodular functions subject to cardinality constraints with the same approximation ratio of $(\frac{1}{2}-\varepsilon)$. Furthermore, Norouzi-Fard et al. [32] showed that any single-pass streaming algorithm must use $\Omega(\frac{n}{k})$ memory to achieve an approximation ratio of over $\frac{1}{2}$. They also proposed streaming algorithms with approximation factors better than $\frac{1}{2}$ by assuming that items arrived in random order or running in multiple passes. Alaluf et al. [2] proposed a 0.2779-approximation streaming algorithm for maximizing non-monotone submodular functions with cardinality constraints. Moreover, streaming submodular maximization is also studied in different models, e.g., the sliding-window model [12, 38] where only recent items within a time window are available for selection, the time-decay model [40] where the weights of old items decrease over time, and the deletion-robust model [22, 29, 30] where existing items might be removed from the stream. However, all above steaming algorithms are specific for the cardinality constraint and cannot be directly used for the fairness constraint in this paper. We note that Kazemi et al. [22] also introduce *fairness* into submodular maximization problems. However, they consider removing sensitive items from the dataset for ensuring fairness, which is different from the problem we study in this paper.

Chakrabarti and Kale [6] proposed a $\frac{1}{4p}$-approximation single-pass streaming algorithms for maximizing monotone submodular functions with the intersections of p matroid constraints. Chekuri et al. [8] generalized the algorithm in [6] to the case of non-monotone submodular functions. Both algorithms achieve a $\frac{1}{4}$-approximation ratio for the FSM problem. Chan et al. [7] improved the approximation ratio for partition matroids to 0.3178 via randomization and relaxation. Feldman et al. [14] introduced a subsampling method to speed up the algorithm of [8] while still achieving an approximation ratio of $\frac{1}{4p}$ in expectation. Huang et al. [18] proposed an $O(\frac{1}{\varepsilon})$-pass $\frac{1}{2+\varepsilon}$-approximation algorithm for monotone submodular maximization with matroid constraints. We implement the aforementioned algorithms from [6, 8, 14, 18] as baselines in our experiments. We do not implement the algorithm in [7] since it is not scalable to large datasets.

Another line of research related to this work is *fair data summarization*. Fair k-center for data summarization was studied in [9, 19, 24]. Celis et al. [5] proposed a determinantal point process (DPP) based sampling method for fair data summarization. Dash et al. [11] considered the fairness issue on summarizing user-generated textual content. Although these studies adopt similar definitions of fairness constraints to ours, their proposed methods cannot be applied to the FSM problem since the objective functions of the problems they study are not submodular.

3 PROBLEM DEFINITION

We consider the problem of selecting a subset of representative items from a dataset V of size n. Our goal is to maximize a non-negative set function $f : 2^V \to \mathbb{R}^+$, where, for any subset $S \subseteq V$, $f(S)$ quantifies the utility of S, i.e., how well S represents V according to some objective. In many data summarization problems (e.g., [4, 12, 17, 28]), the utility function satisfies an intuitive *diminishing returns* property called *submodularity*. To describe it formally, we define the *marginal gain* $\Delta_f(v|S) \coloneqq f(S \cup \{v\}) - f(S)$ as the

Algorithm 1: GREEDY

Input : Dataset V, groups $V_1, \ldots, V_l \subseteq V$, total size constraint $k \in \mathbb{Z}^+$, group size constraints $k_1, \ldots, k_l \in \mathbb{Z}^+$

Output : Solution S for the FSM problem on V

Initialize the solution $S \leftarrow \emptyset$;
for $j \leftarrow 1, \ldots, k$ **do**
  for $i \leftarrow 1, \ldots, l$ **do**
    if $|S \cap V_i| < k_i$ **then**
      $v_i^* \leftarrow \arg\max_{v \in V_i} \Delta_f(v|S)$;
    else
      $v_i^* \leftarrow$ NULL;
  $v^* \leftarrow \arg\max_{i \in [l]: v_i^* \neq \text{NULL}} \Delta_f(v_i^*|S)$;
  $S \leftarrow S \cup \{v^*\}$, $V \leftarrow V \setminus \{v^*\}$;
return S;

increase in utility when an item v is added to a set S. A set function f is *submodular* iff $\Delta_f(v|A) \geq \Delta_f(v|B)$ for any $A \subseteq B \subseteq V$ and $v \in V \setminus B$. This means that adding an item e to a set A leads to at least as much utility gain as adding v to a superset B of A. Additionally, a submodular function f is *monotone* iff $\Delta_f(v|S) \geq 0$ for any $S \subseteq V$ and $v \in V \setminus S$, i.e., adding any new item v never decreases the utility of S. In this work, we assume that the function f is both monotone and submodular. Moreover, following most existing works [4, 8, 12, 14, 17, 21, 27, 32], we assume that the utility $f(S)$ of any set $S \subseteq V$ is given by a value oracle – i.e., the value of $f(S)$ is retrieved in constant time.

Let us consider the following canonical optimization problem: given a monotone submodular set function f and a dataset V, find a subset of size k from V that maximizes the function f, i.e.,

$$\max_{S \subseteq V} f(S) \quad \text{s.t.} \quad |S| = k \tag{1}$$

The problem in Eq. 1 is referred to as the cardinality-constrained submodular maximization (CSM) problem and proven to be NP-hard [13] for many classes of submodular functions. And the well-known greedy algorithm of Nemhauser et al. [31] achieves a $(1 - \frac{1}{e})$-approximation for this problem.

Now we introduce the *fairness* issue into the CSM problem. Let $[l] = \{1, \ldots, l\}$. Suppose that the dataset V is partitioned into l (disjoint) groups, each of which corresponds to a sensitive class, and V_i is the set of items from the i-th group in V with $\bigcup_{i=1}^{l} V_i = V$. Then, for each group, we demand that the solution S must contain k_i items from V_i, and $\sum_{i=1}^{l} k_i = k$. Formally, the fair submodular maximization (FSM) problem is defined as follows:

$$S^* = \arg\max_{S \subseteq V} f(S) \quad \text{s.t.} \quad |S \cap V_i| = k_i, \forall i \in [l] \tag{2}$$

where S^* and $\texttt{OPT} = f(S^*)$ denote the optimal solution and its utility. The values of $k_1, \ldots, k_l \in \mathbb{Z}^+$ are given as input to the problem (here, we assume $k_i > 0$ since we can simply ignore all items in V_i if $k_i = 0$) and determined according to the notion of fairness. For example, one can use $k_i = \frac{n_i}{n} \cdot k$ where $n_i = |V_i|$ to obtain a *proportional representation*. As another example, an *equal representation* can be acquired by setting $k_i = \frac{k}{l}$ for all $i \in [l]$.

The FSM problem in Eq. 2 is still NP-hard because the CSM problem in Eq. 1 is its special case when $l = 1$. Nevertheless, a modified GREEDY algorithm first proposed in [15] provides a $\frac{1}{2}$-approximate solution for the FSM problem, since the fairness constraint we consider is a case of the *partition matroid* constraint. The procedure of GREEDY is described in Algorithm 1. Starting from $S = \emptyset$, it iteratively adds an item v^* with the maximum utility gain $\Delta_f(v^*|S)$ to the current solution S. To guarantee that solution S satisfies the fairness constraint, it excludes from consideration all items of V_i once there are k_i items from V_i in S, i.e., $|S \cap V_i| = k_i$. The solution S after k iterations is returned for the FSM problem. The running time of GREEDY is $O(nk)$ because it runs k passes through the dataset and evaluates the value of f at most n times per pass for identifying v_i^*. Therefore, GREEDY becomes very inefficient when the dataset size is large; even worse, GREEDY cannot work in the single-pass streaming setting if the dataset does not fit in the memory. In what follows, we will investigate the FSM problem in streaming settings.

4 OUR ALGORITHMS

In this section, we present our proposed algorithms for the fair submodular maximization (FSM) problem in data streams. Firstly, we propose a multi-pass streaming algorithm called MP-FSM. For any constant $\varepsilon \in (0, 1)$, MP-FSM requires $O\big(\frac{1}{\varepsilon} \log \frac{k}{\varepsilon}\big)$ passes over the dataset, stores $O(k)$ items in memory, and provides a $\frac{1}{2}(1 - \varepsilon)$-approximate solution for the FSM problem. Secondly, we propose a single-pass streaming algorithm called SP-FSM on the top of MP-FSM. SP-FSM has an approximation ratio of $(\frac{1}{2} - \varepsilon)$ and sublinear update time per item. But it might keep $O(n)$ items in a buffer for post-processing in the worst case, and thus its space complexity is $O(n)$. Therefore, we further discuss how to restrict the buffer size of SP-FSM when the memory space is limited and how the approximation ratio of SP-FSM is affected accordingly.

4.1 Multi-Pass Streaming Algorithm

In this subsection, we present our multi-pass streaming algorithm called MP-FSM for the FSM problem. In general, MP-FSM adopts a threshold-based approach similar to existing streaming algorithms for the CSM problem [4, 21, 26, 32]. The high-level idea of the threshold-based approach is to process items in a data stream sequentially with a threshold τ: for each item v received from the stream, it will accept v into a solution S if $\Delta_f(v|S)$ reaches τ and discard v otherwise. But differently from most thresholding algorithms [4, 21, 26] for the CSM problem, which run in only one pass and use a fixed threshold for each candidate solution, MP-FSM scans the dataset in multiple passes using a decreasing threshold to determine whether to include an item in each pass so that the solution has a constant approximation ratio while satisfying the fairness constraint.

We present the detailed procedure of MP-FSM in Algorithm 2. In the first pass, it finds the item v_{max} with the maximum utility $\delta_{max} = f(\{v_{max}\})$ among all items in the dataset V. The purpose of finding v_{max} is to determine the range of thresholds to be used in subsequent passes. Meanwhile, it keeps a random sample R_i of k_i items uniformly from V_i for each $i \in [l]$, which will be used for post-processing to guarantee that the solution satisfies the fairness constraint. Then, it initializes a solution S containing only v_{max}

Algorithm 2: MP-FSM

Input : Dataset V, groups $V_1, \ldots, V_l \subseteq V$, total size constraint $k \in \mathbb{Z}^+$, group size constraints $k_1, \ldots, k_l \in \mathbb{Z}^+$, parameter $\varepsilon \in (0, 1)$

Output: Solution S for the FSM problem on V

/* Pass 1: Get v_{max} and reservoir sampling */

$v_{max} \leftarrow \arg\max_{v \in V} f(\{v\})$ and $\delta_{max} \leftarrow f(\{v_{max}\})$;

Keep a random sample R_i of k_i items uniformly from V_i for each $i \in [l]$ via reservoir sampling [36];

/* Pass 2 to p: Compute solution S */

$S \leftarrow \{v_{max}\}$ and $\tau \leftarrow (1-\varepsilon) \cdot \delta_{max}$;

while $\tau > \frac{\varepsilon}{k} \cdot \delta_{max}$ **do**

  foreach *item* $v \in V \setminus S$ **do**

    if $v \in V_i$ *and* $|S \cap V_i| < k_i$ *and* $\Delta_f(v|S) \geq \tau$ **then**

      $S \leftarrow S \cup \{v\}$;

  if $|S| = k$ **then**

    break;

  else

    $\tau \leftarrow (1-\varepsilon) \cdot \tau$;

/* Post processing: Ensure fairness */

while $\exists i \in [l] : |S \cap V_i| < k_i$ **do**

  Add items in R_i to S until $|S \cap V_i| = k_i$;

return S;

and a threshold $\tau = (1-\varepsilon) \cdot \delta_{max}$ for the second pass. After that, it scans the dataset V sequentially in multiple passes. In each pass, it decreases the threshold τ by $(1-\varepsilon)$ times and adds an item $v \in V_i$ to the current solution S if the marginal gain of v w.r.t. S reaches τ and there are fewer than k_i items in S from V_i. When the solution S has contained k items or the threshold τ has been decreased to be lower than $\frac{\varepsilon}{k} \cdot \delta_{max}$, no more passes are needed. Finally, if the solution S does not satisfy the fairness constraint, it will add items from random samples to S for ensuring its validity.

Next, we provide some theoretical analyses for the MP-FSM algorithm. First, we give the approximation ratio of MP-FSM in Theorem 4.1. And then, the complexity of MP-FSM is analyzed in Theorem 4.2.

Theorem 4.1. *For any parameter $\varepsilon \in (0, 1)$, MP-FSM in Algorithm 2 is a $\frac{1}{2}(1-\varepsilon)$-approximation algorithm for the FSM problem.*

Proof. Let O be the optimal solution for the FSM problem on dataset V and $O_i = O \cap V_i$ be the intersection of O and V_i for each $i \in [l]$. We consider that MP-FSM runs in p passes and $S^{(j)}$ $(1 \leq j \leq p)$ is the partial solution of MP-FSM after j passes. For any subset O_i of O and the solution $S^{(p)}$ after p passes, we have either (1) $|S^{(p)} \cap V_i| = k_i$ or (2) $|S^{(p)} \cap V_i| < k_i$. If $|S^{(p)} \cap V_i| = k_i$, there are two cases for each item $o \in O_i$: (1.1) $o \in S^{(p)}$ and (1.2) $o \notin S^{(p)}$. In Case (1.1), we have $\Delta_f(o|S^{(p)}) = 0$. In Case (1.2), we compare o with an item s from V_i added to the solution during the j-th pass. Since both o and s cannot be added in the $(j-1)$-th pass and $|S^{(j-1)} \cap V_i| < k_i$, it is safe to say that the marginal gains of o and s w.r.t. $S^{(j-1)}$ do not reach the threshold $\tau^{(j-1)}$ of the $(j-1)$-th pass. As s is added in the j-th pass, we have $\Delta_f(s|S') \geq \tau^{(j)}$ where $S' \subseteq S^{(j)}$ is the partial solution before s is added. Therefore, we have the following sequence of inequalities:

$$\Delta_f(o|S^{(p)}) \leq \Delta_f(o|S^{(j-1)}) < \tau^{(j-1)} = \frac{\tau^{(j)}}{1-\varepsilon} \leq \frac{\Delta_f(s|S')}{1-\varepsilon} \tag{3}$$

Then, if $|S^{(p)} \cap V_i| < k_i$, there are also two cases for $o \in O_i$: (2.1) $o \in S^{(p)}$ and (2.2) $o \notin S^{(p)}$. Case (2.1) is exactly the same as Case (1.1). In Case (2.2), we have:

$$\Delta_f(o|S^{(p)}) < \tau^{(p)} \leq \frac{\varepsilon}{k(1-\varepsilon)} \cdot \delta_{max} \tag{4}$$

where $\tau^{(p)}$ is the threshold of the p-th pass.

Next, we divide O into two disjoint subsets O' and O'' as follows: $O' = \bigcup_{i'} O_{i'}$ where $|S^{(p)} \cap V_{i'}| = k_{i'}$, i.e., all items from groups satisfying Case (1), and $O'' = O \setminus O'$, i.e., all items from groups satisfying Case (2). We define an injection $\pi : O' \to S^{(p)}$ that maps each item in O' to an item in $S^{(p)}$ as follows: If $o \in S^{(p)}$, then $\pi(o) = o$; otherwise, $\pi(o)$ will be an arbitrary item $s \in S^{(p)}$ from the same group as o and $s \notin O$. Based on the result of Eq. 3, we can get the following inequalities for O':

$$\sum_{o \in O'} \Delta_f(o|S^{(p)}) \leq \frac{\sum_{\pi(o) \in S^{(p)}} \Delta_f(\pi(o)|S')}{1-\varepsilon} \leq \frac{f(S^{(p)})}{1-\varepsilon} \tag{5}$$

Here, S' denotes the partial solution before $\pi(o)$ is added and the second inequality is acquired from the fact that $f(S^{(p)}) = \sum_{s \in S^{(p)}} \Delta_f(s|S')$. Then, based on the result of Eq. 4, we have the following inequalities for O'':

$$\sum_{o \in O''} \Delta_f(o|S^{(p)}) \leq \frac{\varepsilon |O''|}{k(1-\varepsilon)} \cdot \delta_{max} \leq \frac{\varepsilon}{1-\varepsilon} \cdot f(S^{(p)}) \tag{6}$$

because $|O''| < k$ and $\delta_{max} \leq f(S^{(p)})$. Finally, we have the following sequence of inequalities from Eq. 5 and 6:

$$\begin{aligned} f(O \cup S^{(p)}) - f(S^{(p)}) &= \sum_{o \in O'} \Delta_f(o|S^{(p)}) + \sum_{o \in O''} \Delta_f(o|S^{(p)}) \\ &\leq \frac{1}{1-\varepsilon} \cdot f(S^{(p)}) + \frac{\varepsilon}{1-\varepsilon} \cdot f(S^{(p)}) = \frac{1+\varepsilon}{1-\varepsilon} \cdot f(S^{(p)}) \end{aligned}$$

Since $\mathtt{OPT} = f(O) \leq f(O \cup S^{(p)})$, we get $\mathtt{OPT} \leq (1 + \frac{1+\varepsilon}{1-\varepsilon}) \cdot f(S^{(p)}) \leq \frac{2}{1-\varepsilon} \cdot f(S^{(p)})$. Finally, we conclude the proof from the fact that $f(S) \geq f(S^{(p)}) \geq \frac{1}{2}(1-\varepsilon)\mathtt{OPT}$. □

Theorem 4.2. *MP-FSM in Algorithm 2 requires $O\left(\frac{1}{\varepsilon} \log \frac{k}{\varepsilon}\right)$ passes over the dataset V, stores at most $O(k)$ items, and has $O\left(\frac{n}{\varepsilon} \log \frac{k}{\varepsilon}\right)$ time complexity.*

Proof. First of all, since the threshold τ is decreased by $(1-\varepsilon)$ times after one pass, $\tau^{(2)} = (1-\varepsilon) \cdot \delta_{max}$, and $\tau^{(p)} \geq \frac{\varepsilon}{k} \cdot \delta_{max}$, we get $(1-\varepsilon)^{p-1} \geq \frac{\varepsilon}{k}$. Taking the logarithm on both sides of the last inequality and the Taylor expansion of $\log(1-\varepsilon)$, we have $p - 1 \leq \frac{1}{\log(1-\varepsilon)} \cdot \log \frac{\varepsilon}{k} \leq \frac{1}{\varepsilon} \log \frac{k}{\varepsilon}$ and thus the number p of passes in MP-FSM is $O\left(\frac{1}{\varepsilon} \log \frac{k}{\varepsilon}\right)$. Furthermore, MP-FSM only stores items in the solution and random samples for post-processing, both of which contain at most k items. Hence, MP-FSM stores at most $O(k)$ items. Finally, because MP-FSM evaluates the value of function f at most n times per pass, the total number of function evaluations in MP-FSM is $O\left(\frac{n}{\varepsilon} \log \frac{k}{\varepsilon}\right)$. □

4.2 Single-Pass Streaming Algorithm

In this subsection, we present our single-pass streaming algorithm called SP-FSM for the FSM problem. Generally, SP-FSM is based on a threshold-based approach, similar to MP-FSM. However, several adaptations are required so that SP-FSM can provide an approximate solution in only one pass over the dataset. First of all, because v_{max} and δ_{max} are unknown in advance, SP-FSM should keep track of them from received items, dynamically decide a sequence of thresholds based on the observed δ_{max}, and maintain a candidate solution for each threshold (instead of keeping only one solution over multiple passes in MP-FSM). Furthermore, as only one pass is permitted, an item will be unrecoverable once it is discarded. To provide a theoretical guarantee for the quality of solutions in adversarial settings, SP-FSM keeps a buffer to store items that are neither included into solutions nor safely discarded. Finally, whenever a solution is requested during the stream, SP-FSM will reconsider the buffered items for post-processing by attempting to add them greedily to candidate solutions. We will show that SP-FSM has an approximation ratio of $(\frac{1}{2}-\varepsilon)$ with a judicious choice of parameters when the buffer size is unlimited.

The detailed procedure of SP-FSM is presented in Algorithm 3. Here, δ_{max} keeps the maximum utility of any single item among all items received so far, LB maintains the lower bound of OPT estimated from candidate solutions, B stores the buffered items, and R_i is a set of k_i items sampled uniformly from all received items in V_i. In addition, two parameters α and β affect the number of candidate solutions and the number of buffered items, respectively. For larger values of α, the gaps between neighboring thresholds are bigger and thus the numbers of candidate solutions are fewer; for larger values of β, the conditions for adding an item to the buffer are more rigorous and naturally the buffer sizes are smaller. The procedure for stream processing of SP-FSM is given in Line 2–14. For each item $v \in V_i$ received from V, it first updates the value of δ_{max} and the sample R_i w.r.t. v. Then, it maintains a sequence T of thresholds picked from a geometric progression $\{(1+\alpha)^j | j \in \mathbb{Z}\}$ and a candidate solution S_τ for each $\tau \in T$. Specifically, the upper bound of the threshold is set to δ_{max} since $S_\tau = \emptyset$ for any $\tau > \delta_{max}$; the lower bound is set to $\frac{\max\{\delta_{max}, \mathsf{LB}\}}{2k}$ because any candidate with a threshold lower than $\frac{\mathsf{OPT}}{2k}$ is safe to be discarded (as shown in our theoretical analysis later) and $\max\{\delta_{max}, \mathsf{LB}\}$ is the lower bound of OPT. After maintaining the thresholds and their corresponding candidates, SP-FSM evaluates the marginal gain $\Delta_f(v|S_\tau)$ of v for each candidate S_τ with threshold $\tau \in T$. Similar to MP-FSM, it will add v to S_τ if $\Delta_f(v|S_\tau)$ reaches τ and $|S_\tau \cap V_i| < k_i$. Additionally, it will add v to the buffer B if $\Delta_f(v|S_\tau)$ is at least $\frac{\beta \cdot \mathsf{LB}}{k}$ but less than τ. Finally, LB is updated to the utility of the best solution found so far. The procedure for post-processing of SP-FSM is shown in Lines 15–17. It first finds out the smallest $\tau \in T$ such that $|S_\tau \cap V_i| < k_i$ for each $i \in [l]$ as τ'; if such τ does not exist, i.e., there exists some i such that $|S_\tau \cap V_i| = k_i$ for every S_τ, the largest $\tau \in T$ is used as τ'. For each $\tau \le \tau'$ in T, it runs Greedy in Algorithm 1 to reevaluate the items in B and R_i and add them to S_τ until $|S_\tau| = k$. Lastly, the candidate solution with the maximum utility after post-processing is returned as the final solution.

Next, we will provide the theoretical analyses for the SP-FSM algorithm. First, in Lemma 4.3, we analyze the special cases when the solution returned after stream processing (without post-processing) can achieve a good approximation ratio.

Algorithm 3: SP-FSM

Input : Data stream V, groups $V_1, \ldots, V_l \subseteq V$, total size constraint $k \in \mathbb{Z}^+$, group size constraints $k_1, \ldots, k_l \in \mathbb{Z}^+$, parameters $\alpha, \beta \in (0,1)$

Output : Solution S for the FSM problem on V

1 $\delta_{max} \leftarrow 0$, $\mathsf{LB} \leftarrow 0$, $B \leftarrow \emptyset$, and $R_i \leftarrow \emptyset$ for each $i \in [l]$;
/* Stream processing */
2 **foreach** *item* $v \in V_i$ *received from* V **do**
3 $\delta_{max} \leftarrow \max\{\delta_{max}, f(\{v\})\}$;
4 Update R_i w.r.t. v using reservoir sampling [36];
5 $T \leftarrow \{(1+\alpha)^j | j \in \mathbb{Z}, \frac{\max\{\delta_{max}, \mathsf{LB}\}}{2k} \le (1+\alpha)^j \le \delta_{max}\}$;
6 Discard S_τ for all $\tau \notin T$;
7 Initialize $S_\tau \leftarrow \emptyset$ for each τ newly added to T;
8 **foreach** $\tau \in T$ **do**
9 **if** $|S_\tau \cap V_i| < k_i$ **then**
10 **if** $\Delta_f(v|S_\tau) \ge \tau$ **then**
11 $S_\tau \leftarrow S_\tau \cup \{v\}$;
12 **else if** $\Delta_f(v|S_\tau) \ge \frac{\beta \cdot \mathsf{LB}}{k}$ **then**
13 $B \leftarrow B \cup \{v\}$;
14 $\mathsf{LB} \leftarrow \max_{\tau \in T} f(S_\tau)$;
/* Post processing */
15 Let τ' be the smallest $\tau \in T$ such that $|S_\tau \cap V_i| < k_i$ for each $i \in [l]$ or the largest $\tau \in T$ if there exists some i such that $|S_\tau \cap V_i| = k_i$ for every S_τ;
16 **foreach** $\tau \le \tau'$ *in* T **do**
17 Run Greedy in Algorithm 1 to add items from buffer B and samples R_i for all $i \in [l]$ to S_τ until $|S_\tau| = k$;
18 **return** $S \leftarrow \arg\max_{\tau \in T} f(S_\tau)$;

Lemma 4.3. *Assume that* $\frac{\mathsf{OPT}}{2k} \le \tau \le \frac{(1+\alpha)\cdot\mathsf{OPT}}{2k}$. *If either* $|S_\tau| = k$ *or* $|S_\tau \cap V_i| < k_i$ *for all* $i \in [l]$, *then* $f(S_\tau) \ge \frac{1-\alpha}{2} \cdot \mathsf{OPT}$.

Proof. First of all, when $|S_\tau| = k$, it holds that $f(S_\tau) \ge k\tau \ge k \cdot \frac{\mathsf{OPT}}{2k} = \frac{1}{2} \cdot \mathsf{OPT} \ge \frac{1-\alpha}{2} \cdot \mathsf{OPT}$. Then, when $|S_\tau \cap V_i| < k_i$ for all $i \in [l]$, we have $\Delta_f(v|S_\tau) < \tau$ for any $v \in V \setminus S_\tau$. Let O be the optimal solution for the FSM problem on V. We can acquire that

$$f(O \cup S_\tau) - f(S_\tau) \le \sum_{o \in O \setminus S_\tau} \Delta_f(o|S_\tau) < k\tau \le k \cdot \frac{(1+\alpha)\cdot\mathsf{OPT}}{2k} = (1+\alpha)\cdot\frac{\mathsf{OPT}}{2}$$

Therefore, we have $f(S_\tau) \ge f(O \cup S_\tau) - (1+\alpha)\cdot\frac{\mathsf{OPT}}{2} \ge \mathsf{OPT} - (1+\alpha)\cdot\frac{\mathsf{OPT}}{2} = \frac{1-\alpha}{2}\cdot\mathsf{OPT}$. We conclude the proof by considering both cases collectively. □

Lemma 4.3 is useful because one of the thresholds $\tau \in T$ of SP-FSM (Line 5 of Algorithm 3) must satisfy the first condition $\frac{\mathsf{OPT}}{2k} \le \tau \le \frac{(1+\alpha)\cdot\mathsf{OPT}}{2k}$ of the lemma. This is because T is a geometric progression with a scale factor of $(1+\alpha)$ and spans the range $[\frac{\max\{\delta_{max}, \mathsf{LB}\}}{2k}, \delta_{max}]$, with $\max\{\delta_{max}, \mathsf{LB}\} \le \mathsf{OPT} \le k \cdot \delta_{max}$.

This implies that, if the remaining conditions of Lemma 4.3 were satisfied as well, the solution of SP-FSM after stream processing would have the strong approximation guarantee given by Lemma 4.3. Intuitively, this would be the case when the utility distribution of items was generally "balanced" among groups, so that either all or none of the group budgets would be exhausted by the end of stream processing. However, in case that the utilities are highly imbalanced among groups, the approximation ratio would become significantly lower. On the one hand, SP-FSM might miss high-utility items in some groups from the stream because the threshold is too low and the solution has been filled by earlier items with lower utilities in these groups. On the other hand, SP-FSM might not include enough items from the other groups because the threshold is too high for them. Note that, for $\frac{\texttt{OPT}}{2k} \leq \tau \leq \frac{(1+\alpha)\cdot\texttt{OPT}}{2k}$, Lemma 4.3 allows the approximation factor of S_τ to drop to $\frac{\min_{i\in[l]} k_i\tau}{\texttt{OPT}} \geq \min_{i\in[l]} \frac{k_i}{2k} \geq \frac{1}{2k}$ when some group budgets are exhausted but the others are not.

Therefore, we further include the buffer and post-processing procedures in SP-FSM so that it still achieves a constant approximation independent of k for an arbitrary group size constraint. In Lemma 4.4, we analyze the approximation ratio of the solution returned by SP-FSM after post-processing.

Lemma 4.4. *Let τ' be chosen according to Line 15 of Algorithm 3. It holds that $f(S_{\tau'}) \geq \frac{1-\beta}{2+\alpha} \cdot \texttt{OPT}$ after post-processing.*

Proof. We consider two cases separately: (1) $|S_{\tau'} \cap V_i| < k_i$ for each $i \in [l]$ or (2) τ' is the maximum in T. In Case (1), we divide the items in the optimal solution O into three disjoint subsets: $O_1 = O \cap S_{\tau'}$, i.e., items included in $S_{\tau'}$ during stream and post processing; $O_2 = O \cap (B \setminus S_{\tau'})$, i.e., items stored in the buffer but not added to $S_{\tau'}$; $O_3 = O \cap (V \setminus (B \cup S_{\tau'}))$, i.e., items discarded during stream processing. For each $o \in O_2$, we can always find an item $s \in S_{\tau'}$ from the same group as o such that $\Delta_f(s|S') \geq \Delta_f(o|S') \geq \Delta_f(o|S_{\tau'})$ where $S' \subseteq S_{\tau'}$ is the partial solution when s is added. This is because Greedy always picks the item with the maximum marginal gain within each group. In addition, for each $o \in O_3$, we have $\Delta_f(o|S_{\tau'}) \leq \frac{\beta\cdot\texttt{LB}}{k} \leq \frac{\beta\cdot\texttt{OPT}}{k}$. Therefore, we have

$$\begin{aligned} f(O \cup S_{\tau'}) - f(S_{\tau'}) &\leq \sum_{o\in O\setminus S_{\tau'}} \Delta_f(o|S_{\tau'}) \\ &= \sum_{o\in O_2} \Delta_f(o|S_{\tau'}) + \sum_{o\in O_3} \Delta_f(o|S_{\tau'}) \\ &\leq \sum_{s\in S_{\tau'}} \Delta_f(s|S') + \beta\cdot\texttt{OPT} \\ &= f(S_{\tau'}) + \beta\cdot\texttt{OPT} \end{aligned}$$

where S' is the partial solution when s is added to $S_{\tau'}$. And we conclude that $f(S_{\tau'}) \geq \frac{1-\beta}{2}\cdot\texttt{OPT}$ from the above inequalities. In Case (2), we have τ' is the maximum in T and thus $\tau' \in [\frac{\delta_{max}}{1+\alpha}, \delta_{max}]$. We divide O into O_1, O_2, O_3 in the same way as Case (1). It is easy to see that the results for O_1 and O_3 are exactly the same as Case (1). The only difference is that there may exist some items in O_2 rejected by $S_{\tau'}$ because their groups have been filled in $S_{\tau'}$. For any $o \in O_2$, we have $\Delta_f(o|S_{\tau'}) \leq \delta_{max} \leq (1+\alpha)\cdot\tau' \leq (1+\alpha)\cdot\Delta_f(s|S')$ where s is from the same group as o and S' is the partial solution when s is added. Accordingly, we can get $\texttt{OPT} - f(S_{\tau'}) \leq (1+\alpha)\cdot f(S_{\tau'}) + \beta\cdot\texttt{OPT}$ and thus $f(S_{\tau'}) \geq \frac{1-\beta}{2+\alpha}\cdot\texttt{OPT}$ in both cases. □

Next, we give the approximation ratio and complexity of SP-FSM in Theorems 4.5 and 4.6, respectively.

Theorem 4.5. *Assuming that $\alpha, \beta = O(\varepsilon)$, SP-FSM in Algorithm 3 is a $(\frac{1}{2} - \varepsilon)$-approximation algorithm for the FSM problem.*

Proof. According to the results of Lemmas 4.3 and 4.4, we have $f(S) \geq \frac{1-\beta}{2+\alpha}\cdot\texttt{OPT}$ for the solution S returned by Algorithm 3. By assuming $\alpha, \beta = O(\varepsilon)$, we conclude the proof. □

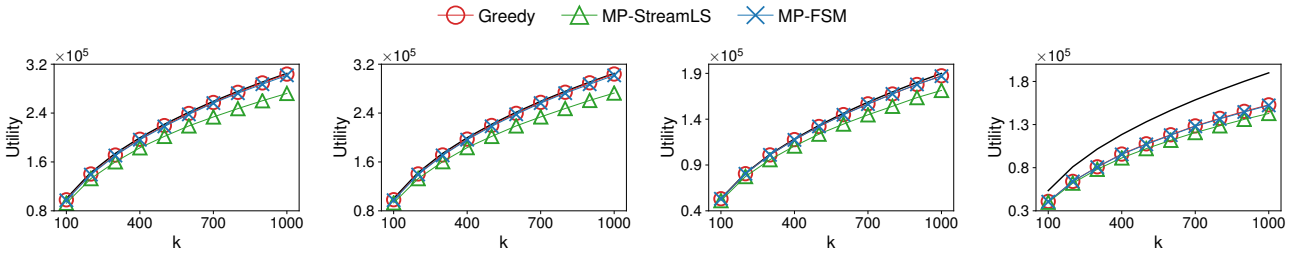
Theorem 4.6. *Assuming that $\alpha, \beta = O(\varepsilon)$, SP-FSM in Algorithm 3 requires one pass over the data stream V, stores at most $O\big(\frac{k\log k}{\varepsilon} + |B|\big)$ items, has $O\big(\frac{\log k}{\varepsilon}\big)$ update time per item for stream processing, and takes $O\big(\frac{k\log k}{\varepsilon}\cdot(|B|+k)\big)$ time for post-processing.*

Proof. The number $|T|$ of thresholds maintained at any time satisfies that $(1+\alpha)^{|T|} \leq 2k$. Using the Taylor expansion of $\log(1+\alpha)$, we have $|T| \leq \frac{\log 2k}{\log(1+\alpha)} \leq \frac{\log 2k}{\alpha \log 2} = O\big(\frac{\log k}{\alpha}\big)$. Therefore, the number of function evaluations per item is $O\big(\frac{\log k}{\varepsilon}\big)$. Since each candidate solution contains at most k items, the total number of items stored in SP-FSM is $O\big(\frac{k\log k}{\varepsilon} + |B|\big)$. For each candidate solution S_τ, the post-processing procedure runs in $(k - |S_\tau|)$ iterations and processes at most $(|B| + k)$ items at each iteration. Therefore, it takes $O\big(\frac{k\log k}{\varepsilon}\cdot(|B|+k)\big)$ time for post-processing. □

4.3 SP-FSM with Bounded Buffer Size

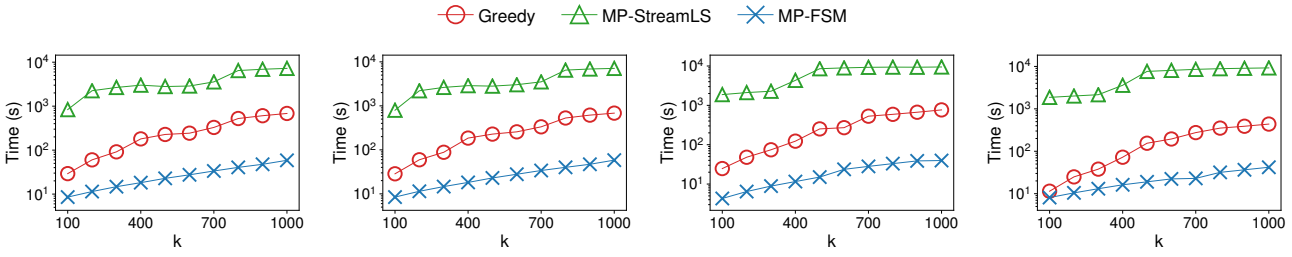
From the above results, we can see that SP-FSM may store $O(n)$ items in the buffer and take $O\big(\frac{nk\log k}{\varepsilon}\big)$ time for post-processing in the worst case. In practice, a streaming algorithm is often required to process massive data streams with limited time and memory (sublinear to or independent of n). And it is not favorable for SP-FSM to store an unlimited number of items in the buffer B. Therefore, we propose a simple strategy for SP-FSM to manage the buffered items so that the buffer size is always bounded at the expense of lower approximation ratios in adversary settings.

We consider that the maximum buffer size is restricted to $k' = O(k)$ and extra items should be dropped from B once its size exceeds k'. The following rules are considered for buffer management. Firstly, since LB increases over time, it is safe to drop at any time during stream processing any item already in the buffer whose marginal gain is lower than $\frac{\beta\cdot\texttt{LB}}{k}$ for the current value of LB, without affecting the theoretical guarantee. Secondly, to avoid duplications, if an item is added to some candidate solution but needs to be buffered for another, it is not necessary to add this item to the buffer because the algorithm has already stored this item. In this case, items in both candidates and the buffer should be used for post-processing. Thirdly, as the buffer is used for storing high-utility items for post-processing, the items with larger marginal gains should have higher priorities to be stored. If the buffer size still exceeds k' after (safely) dropping items using the first two rules, it is required to sort the items in B in a descending order of marginal gain $\delta(v) = \max_{\tau\in T} \Delta_f(v, S_\tau)$ and drop the item v with the lowest $\delta(v)$ until $|B| = k'$. Fourthly, considering the fairness constraint, it



(a) POKEC (Gender, PR) (b) POKEC (Gender, ER) (c) POKEC (Age, PR) (d) POKEC (Age, ER)

Figure 1: Solution utilities of multi-pass algorithms on POKEC. The results of Greedy without any fairness constraint are plotted as black lines to illustrate "the prices of fairness".



(a) POKEC (Gender, PR) (b) POKEC (Gender, ER) (c) POKEC (Age, PR) (d) POKEC (Age, ER)

Figure 2: Running time of multi-pass algorithms on POKEC.

will not drop any item v from V_i anymore if $|B \cap V_i| \leq k_i$ even if $\delta(v)$ is among the lowest marginal gains. In this case, it will drop the item with the lowest $\delta(v)$ from V_i with $|B \cap V_i| > k_i$ instead.

The first two rules above have no effect on the theoretical guarantee on the approximation ratio of SP-FSM. The latter two rules will lower the approximation ratio of SP-FSM in some cases. Let v' be the item with the largest $\delta(v)$ among all items dropped due to Rule (3) or (4). The approximation ratio of SP-FSM will drop to $\frac{1-\beta'}{2}$ where $\beta' = \frac{k \cdot \delta(v')}{\mathsf{LB}}$. Once $\beta' \geq 1 - \frac{1}{k}$, the approximation ratio will become $\frac{1}{2k}$ in the worst case. Nevertheless, according to our experimental results in Section 5, SP-FSM provides high-quality solutions empirically with very small buffer sizes (i.e., $k' = 2k$).

5 EXPERIMENTS

The goal of our experiments is three-fold. First, we aim to quantify "the prices of fairness and streaming", i.e., the losses in solution utilities caused by introducing the fairness constraint and restricting data access to a single pass over the stream. Second, we aim to demonstrate the improvements of MP-FSM upon existing algorithms in the multi-pass streaming setting. Third, we aim to illustrate that SP-FSM (with unlimited and bounded buffer sizes) outperforms existing single-pass streaming algorithms.

Towards this end, we perform extensive experiments on two applications, namely *maximum coverage on large graphs* and *personalized recommendation*, for evaluation. We compare MP-FSM with the following two multi-pass streaming algorithms:

- Greedy: the classic $\frac{1}{2}$-approximation k-pass greedy algorithm proposed by Fisher et al. [15].
- MP-StreamLS: a $\frac{1}{2+\varepsilon}$-approximation $O(\frac{1}{\varepsilon})$-pass streaming algorithm in [18].

Moreover, we compare SP-FSM with the following two single-pass streaming algorithms:

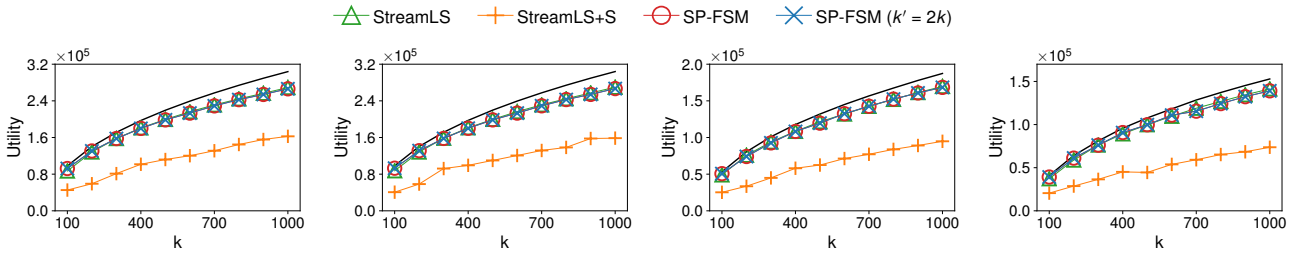
- StreamLS: a $\frac{1}{4}$-approximation streaming algorithm in [6, 8].
- StreamLS+S: an improved version of StreamLS with subsampling in [14]. The subsampling rate q is set to 0.1.

All algorithms were implemented in Python 3.6, and the experiments were conducted on a server running Ubuntu 16.04 with an Intel Broadwell 2.40GHz CPU and 29GB memory. Our implementation is publicly available on GitHub[1]. For each of the experiments, we invoked our algorithms with the following parameter values: MP-FSM with $\varepsilon = 0.2$ and SP-FSM with $\alpha, \beta = 0.5$ and $k' = 2k$ in all cases where the buffer size is bounded. Note that in all the following figures, we refer to SP-FSM with an unlimited buffer size as **SP-FSM** and SP-FSM with $k' = 2k$ as **SP-FSM ($k' = 2k$)**.

5.1 Maximum Coverage on Large Graphs

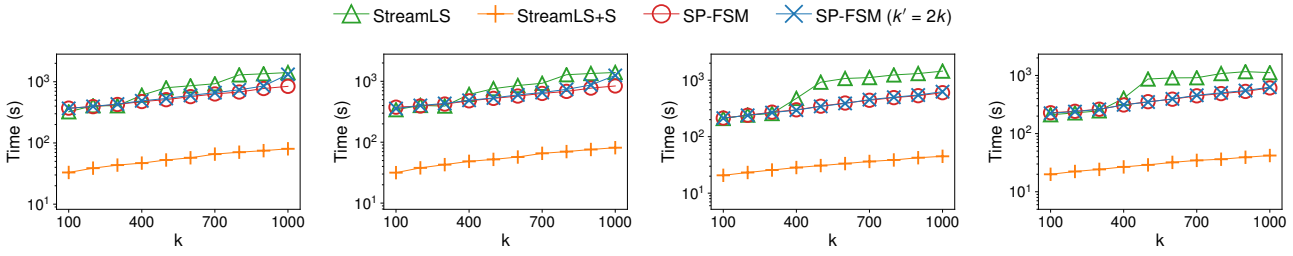
Maximum coverage is a classic submodular optimization task on graphs with many real-world applications such as community detection [16], influence maximization [37], and web monitoring [33]. The goal of this task is to select a small subset of nodes that covers a large portion of nodes in a graph. Formally, given a graph $G = (V, E)$ where $n = |V|$ is the number of nodes and $m = |E|$ is the number of edges, the goal is to find a size-k subset S of V that maximizes the nodes in the neighborhood of S, i.e., $f(S) = |\bigcup_{v \in S} N(v)|$

[1] https://github.com/FraFabbri/fair-subset-datastream



(a) POKEC (Gender, PR) (b) POKEC (Gender, ER) (c) POKEC (Age, PR) (d) POKEC (Age, ER)

Figure 3: Solution utilities of single-pass algorithms on POKEC. The results of Greedy are plotted as black lines to illustrate "the prices of streaming data access".



(a) POKEC (Gender, PR) (b) POKEC (Gender, ER) (c) POKEC (Age, PR) (d) POKEC (Age, ER)

Figure 4: Running time of single-pass algorithms on POKEC.

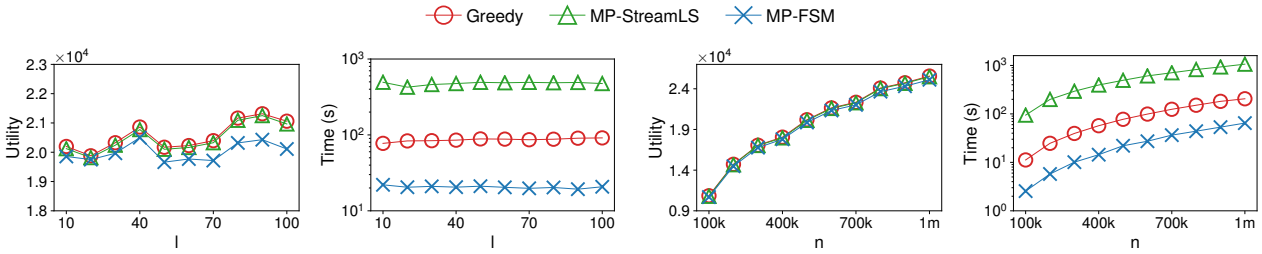
where $N(v)$ is the set of nodes connected to v. It is easy to verify that f is nonnegative, monotone, and submodular.

We perform the experiments for maximum coverage on two graph datasets as follows: (1) **POKEC** is a real-world dataset published on SNAP[2]. It is a directed graph with 1,632,803 nodes and 30,622,564 edges representing the follower/followee relationships among users in Pokec. Each node is associated with a user profile with demographic information. The nodes are partitioned into $l = 2$ groups by gender or $l = 7$ groups by age in our experiments. (2) **SYN** is a set of synthetic graphs generated by the Barabási-Albert model [3] with equal number of nodes and edges, i.e., $n = m$. To test the effect of graph size, we generate different graphs by ranging n from 100k to 1m. The nodes are randomly partitioned into l groups and the group sizes follow a Zipf's distribution with parameter $s = 2$. By default, we set the number l of groups to 10. To test the effect of l, we fix $n = 500$k and vary l from 10 to 100.

In the first set of experiments, we evaluate the performance of Greedy, MP-StreamLS, and MP-FSM in the multi-pass streaming setting. We range the total cardinality constraint $k = \sum_i k_i$ from 100 to 1,000 and use both *proportional representation* (PR) and *equal representation* (ER) to assign the group-specific cardinality constraint k_i for each $i \in [l]$. The solution utilities and running time on POKEC are presented in Figures 1 and 2, respectively. "The price of fairness" – i.e., the loss in utility caused by the fairness constraint, is marginal for PR in both cases of gender and age groups, and ER in the case of gender groups, as two gender groups are roughly balanced (e.g., 51% female vs. 49% male) on POKEC. However, for highly imbalanced groups (e.g., age groups on POKEC), enforcing equal representation leads to significant losses in utilities (see Figure 1(d)). MP-FSM outperforms Greedy and MP-StreamLS in terms of both running time and solution utility in almost all cases. It runs up to 19 and 567 times faster than Greedy and MP-StreamLS, respectively. Meanwhile, its solution utilities are always nearly equal to (at least 99% of) those of Greedy and consistently (up to 10%) higher than those of MP-StreamLS.

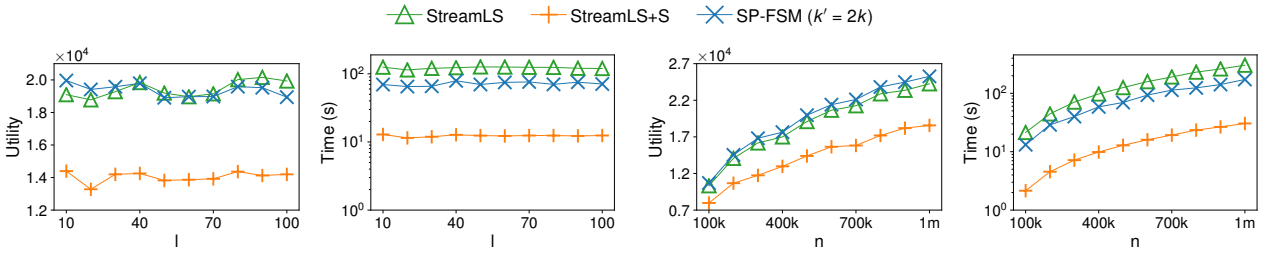
In the second set of experiments, we evaluate the performance of StreamLS, StreamLS+S, and SP-FSM with unlimited and bounded (i.e., $k' = 2k$) buffer sizes in the single-pass streaming setting. We also vary k from 100 to 1,000 and use both PR and ER for fairness constraints. The experimental results on POKEC are illustrated in Figures 3 and 4. Firstly, the utilities of the solutions provided by StreamLS and SP-FSM are typically around 10% lower than the utilities of the solutions of Greedy. This can be seen as "the price of streaming data access" – i.e., the loss in utility for restricting data access only to a single pass over the stream. Secondly, the solution quality of SP-FSM is generally equivalent to or better than that of StreamLS. Meanwhile, the efficiency of SP-FSM is consistently higher than that of StreamLS, particularly so for larger values of k. Thirdly, the performance of SP-FSM is hardly affected by the buffer size: The solution quality and running time of SP-FSM are nearly identical when setting the buffer size to be unlimited or $2k$. This confirms the effectiveness of the buffer management strategies we propose. Fourthly, the subsampling technique used in StreamLS+S does not perform well in our scenario: although it obviously improves the efficiency upon StreamLS, its solution quality becomes significantly inferior to any other algorithm.

[2]https://snap.stanford.edu/data/soc-Pokec.html



(a) Varying l (b) Varying n

Figure 5: Performance of multi-pass algorithms on SYN with varying dataset size n and number of groups l.



(a) Varying l (b) Varying n

Figure 6: Performance of single-pass algorithms on SYN with varying dataset size n and number of groups l.

In the third set of experiments, we test the scalability of different algorithms with varying the number l of groups and the dataset size n on SYN when k is fixed to 500. Because the results for PR and ER are similar to each other, we only present the results for PR. The performance of multi-pass streaming algorithms is shown in Figure 5. The solution utilities of different algorithms keep steady w.r.t. l while growing with increasing n as expected. Meanwhile, the solution quality of Greedy, MP-Stream-LS, and MP-FSM is close to each other with varying l and n. The difference in utilities are within 5% in all cases. Furthermore, the running time of all algorithms generally keeps steady for different values of l and grows near linearly with increasing n. At the same time, MP-FSM runs nearly 10 and 100 times faster than Greedy and MP-Stream-LS, respectively, for different values of l and n. The performance of single-pass streaming algorithms on SYN is shown in Figure 6. Since SP-FSM shows nearly identical performance for different buffer sizes, we only present the results of SP-FSM ($k' = 2k$) here. Generally, we observe the same trends as the multi-pass case with varying l and n. For different values of l and n, the solution quality of SP-FSM and StreamLS is close to each other, but SP-FSM runs much faster than StreamLS. With the benefit of subsampling, StreamLS+S has much higher efficiency than SP-FSM and StreamLS. Nevertheless, its solution quality is obviously worse than them.

In summary, for maximum coverage on large graphs, our experimental results demonstrate that our proposed algorithms MP-FSM and SP-FSM manage to pay small "prices" for the restrictions of the settings (i.e., fairness constraint and streaming data access). And compared with the state-of-the-art algorithms, they exhibit an excellent combination of performance in terms of running time and solution quality.

5.2 Personalized Recommendation

The personalized recommendation problem has been used for benchmarking submodular maximization algorithms in [30, 32]. Its goal is to select a subset S of k items that is both relevant to a given user u and well represents all items in the collection V. Formally, each query user u and each item v in V are denoted by feature vectors in $\mathbb{R}^d$. The relevance between a user and an item is computed by the inner product of their feature vectors. The objective function f is defined as follows:

$$f(S) = \lambda \cdot \sum_{v' \in V} \max_{v \in S} \langle v', v \rangle + (1 - \lambda) \cdot \sum_{v \in S} \langle u, v \rangle$$

and, again, f is known to be nonnegative, monotone and submodular [30]. The first term measures how well a subset S represents the collection V; the second term denotes the relevance of S to user u; and the parameter λ trades off between both terms. We set $\lambda = 0.75$ following [30, 32] in our experiments.

We perform the experiments for personalized recommendation on the **MovieLens** dataset[3]. It contains 3,883 items (movies) and 6,040 users with one million user ratings for movies. We denote each item or user as a 50-dimensional vector by performing Nonnegative Matrix Factorization (NMF) [39] on the user-item rating matrix. The items are partitioned into $l = 10$ groups according to *genre*. Since the results for PR and ER are similar to each other, we omit the results for ER in these experiments.

We present the performance of multi-pass streaming algorithms by ranging k from 10 to 100 in Figure 7. Since the number l of groups is relatively large compared to k, the utility losses caused by fairness constraints are more significant than those in *maximum coverage*.

[3]https://grouplens.org/datasets/movielens/

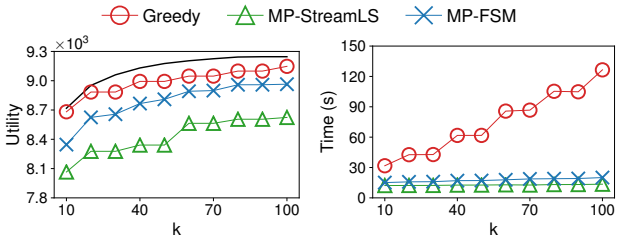


Figure 7: Performance of multi-pass algorithms on MovieLens. The utilities of Greedy without any fairness constraint are plotted as a black line.

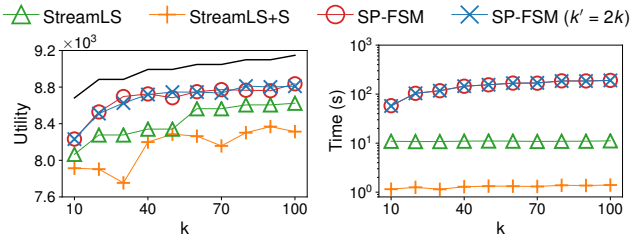


Figure 8: Performance of single-pass algorithms on MovieLens. The utilities of Greedy are plotted as a black line.

Among all multi-pass streaming algorithms, Greedy runs the slowest but achieves the best solution quality. Moreover, MP-FSM shows higher efficiency than Greedy, especially when k becomes larger. Meanwhile, it provides solutions of at least 96% utilities of the solutions of Greedy. Although MP-StreamLS runs faster than MP-FSM and Greedy because of fewer updates in solutions, its solution quality becomes worse as well. We describe the performance of single-pass streaming algorithms by ranging k from 10 to 100 in Figure 8. Similar to the case of *maximum coverage*, the solution utilities of SP-FSM (with unlimited and bounded buffer sizes) are around 10% lower than those of Greedy because only a single pass over the stream is permitted. Nevertheless, SP-FSM provides solutions of higher quality than StreamLS at the expense of longer running time. Finally, StreamLS+S still brings great improvements in efficiency but leads to obvious losses in solution quality.

In summary, for personalized recommendation, our experimental results demonstrate that our proposed algorithms MP-FSM and SP-FSM have good performance compared with the state-of-the-art algorithms: they provide solutions of higher quality than the local search based streaming algorithms (i.e., MP-StreamLS, StreamLS, and StreamLS+S) at the expense of lower efficiency.

6 CONCLUSION

In this paper, we studied the problem of extracting fair and representative items from data streams. We formulated the problem as maximizing monotone submodular functions subject to partition matroid constraints. We first proposed a $(\frac{1}{2} - \varepsilon)$-approximation multi-pass streaming algorithm called MP-FSM for the problem. Then, we designed a single-pass streaming algorithm called SP-FSM for the problem. SP-FSM had the same approximation ratio of $(\frac{1}{2} - \varepsilon)$ as MP-FSM when an unlimited buffer size is permitted, which improved the best-known approximation ratio of $\frac{1}{4}$ in the literature. We further considered the practical implementation of SP-FSM when the buffer sizes are bounded. Finally, extensive experimental results on two real-world applications confirmed the efficiency, effectiveness, and scalability of our proposed algorithms.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments to improve this paper. Yanhao Wang and Michael Mathioudakis have been supported by the MLDB project of Academy of Finland (decision number: 322046). Francesco Fabbri has been supported by the Helsinki Institute for Information Technology (HIIT).

REFERENCES

[1] Zeinab Abbassi, Vahab S. Mirrokni, and Mayur Thakur. 2013. Diversity maximization under matroid constraints. In *KDD*. 32–40.
[2] Naor Alaluf, Alina Ene, Moran Feldman, Huy L. Nguyen, and Andrew Suh. 2020. Optimal Streaming Algorithms for Submodular Maximization with Cardinality Constraints. In *ICALP*. 6:1–6:19.
[3] Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Rev. Mod. Phys.* 74 (2002), 47–97. Issue 1.
[4] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. 2014. Streaming Submodular Maximization: Massive Data Summarization on the Fly. In *KDD*. 671–680.
[5] L. Elisa Celis, Vijay Keswani, Damian Straszak, Amit Deshpande, Tarun Kathuria, and Nisheeth K. Vishnoi. 2018. Fair and Diverse DPP-Based Data Summarization. In *ICML*. 715–724.
[6] Amit Chakrabarti and Sagar Kale. 2015. Submodular maximization meets streaming: matchings, matroids, and more. *Math. Program.* 154, 1-2 (2015), 225–247.
[7] T.-H. Hubert Chan, Zhiyi Huang, Shaofeng H.-C. Jiang, Ning Kang, and Zhihao Gavin Tang. 2017. Online Submodular Maximization with Free Disposal: Randomization Beats ¼ for Partition Matroids. In *SODA*. 1204–1223.
[8] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. 2015. Streaming Algorithms for Submodular Function Maximization. In *ICALP*. 318–330.
[9] Ashish Chiplunkar, Sagar Kale, and Sivaramakrishnan Natarajan Ramamoorthy. 2020. How to Solve Fair k-Center in Massive Data Models. In *ICML*. 6887–6896.
[10] Alexandra Chouldechova and Aaron Roth. 2020. A snapshot of the frontiers of fairness in machine learning. *Commun. ACM* 63, 5 (2020), 82–89.
[11] Abhisek Dash, Anurag Shandilya, Arindam Biswas, Kripabandhu Ghosh, Saptarshi Ghosh, and Abhijnan Chakraborty. 2019. Summarizing User-generated Textual Content: Motivation and Methods for Fairness in Algorithmic Summaries. *Proc. ACM Hum. Comput. Interact.* 3, CSCW (2019), 172:1–172:28.
[12] Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. 2017. Submodular Optimization Over Sliding Windows. In *WWW*. 421–430.
[13] Uriel Feige. 1998. A Threshold of ln n for Approximating Set Cover. *J. ACM* 45, 4 (1998), 634–652.
[14] Moran Feldman, Amin Karbasi, and Ehsan Kazemi. 2018. Do Less, Get More: Streaming Submodular Maximization with Subsampling. In *NeurIPS*. 730–740.
[15] Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. 1978. An analysis of approximations for maximizing submodular set functions—II. In *Polyhedral Combinatorics*, Michel L. Balinski and Alan J. Hoffman (Eds.). Springer Berlin Heidelberg, 73–87.
[16] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. 2014. Overlapping community detection in labeled graphs. *Data Min. Knowl. Discov.* 28, 5-6 (2014), 1586–1610.
[17] Ryan Gomes and Andreas Krause. 2010. Budgeted Nonparametric Learning from Data Streams. In *ICML*. 391âĂŞ398.
[18] Chien-Chung Huang, Theophile Thiery, and Justin Ward. 2020. Improved Multi-Pass Streaming Algorithms for Submodular Maximization with Matroid Constraints. In *APPROX/RANDOM*. 62:1–62:19.
[19] Matthew Jones, Huy Lê Nguyên, and Thy Nguyen. 2020. Fair k-Centers via Maximum Matching. In *ICML*. 7460–7469.
[20] Matthew Kay, Cynthia Matuszek, and Sean A. Munson. 2015. Unequal Representation and Gender Stereotypes in Image Search Results for Occupations. In *CHI*. 3819–3828.
[21] Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. 2019. Submodular Streaming in All Its Glory: Tight Approximation, Minimum Memory and Low Adaptive Complexity. In *ICML*. 3311–3320.
[22] Ehsan Kazemi, Morteza Zadimoghaddam, and Amin Karbasi. 2018. Scalable Deletion-Robust Submodular Maximization: Data Summarization with Privacy and Fairness Constraints. In *ICML*. 2549–2558.
[23] David Kempe, Jon M. Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *KDD*. 137–146.

[24] Matthäus Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. 2019. Fair k-Center Clustering for Data Summarization. In *ICML*. 3448–3457.
[25] Andreas Krause and Daniel Golovin. 2014. Submodular Function Maximization. In *Tractability: Practical Approaches to Hard Problems*, Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli (Eds.). Cambridge University Press, 71–104.
[26] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. 2015. Fast Greedy Algorithms in MapReduce and Streaming. *ACM Trans. Parallel Comput.* 2, 3 (2015), 14:1–14:22.
[27] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. Van Briesen, and Natalie S. Glance. 2007. Cost-effective outbreak detection in networks. In *KDD*. 420–429.
[28] Erik M. Lindgren, Shanshan Wu, and Alexandros G. Dimakis. 2016. Leveraging Sparsity for Efficient Submodular Data Summarization. In *NIPS*. 3414–3422.
[29] Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. 2017. Deletion-Robust Submodular Maximization: Data Summarization with "the Right to be Forgotten". In *ICML*. 2449–2458.
[30] Slobodan Mitrovic, Ilija Bogunovic, Ashkan Norouzi-Fard, Jakub Tarnawski, and Volkan Cevher. 2017. Streaming Robust Submodular Maximization: A Partitioned Thresholding Approach. In *NIPS*. 4557–4566.
[31] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Math. Program.* 14, 1 (1978), 265–294.
[32] Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. 2018. Beyond 1/2-Approximation for Submodular Maximization on Massive Data Streams. In *ICML*. 3826–3835.
[33] Barna Saha and Lise Getoor. 2009. On Maximum Coverage in the Streaming Model & Application to Multi-topic Blog-Watch. In *SDM*. 697–708.
[34] Dimitris Serbos, Shuyao Qi, Nikos Mamoulis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2017. Fairness in Package-to-Group Recommendations. In *WWW*. 371–379.
[35] Ana-Andreea Stoica, Jessy Xinyi Han, and Augustin Chaintreau. 2020. Seeding Network Influence in Biased Networks and the Benefits of Diversity. In *WWW*. 2089–2098.
[36] Jeffrey Scott Vitter. 1985. Random Sampling with a Reservoir. *ACM Trans. Math. Softw.* 11, 1 (1985), 37–57.
[37] Yanhao Wang, Qi Fan, Yuchen Li, and Kian-Lee Tan. 2017. Real-Time Influence Maximization on Dynamic Social Streams. *Proc. VLDB Endow.* 10, 7 (2017), 805–816.
[38] Yanhao Wang, Yuchen Li, and Kian-Lee Tan. 2019. Efficient Representative Subset Selection over Sliding Windows. *IEEE Trans. Knowl. Data Eng.* 31, 7 (2019), 1327–1340.
[39] Yu-Xiong Wang and Yu-Jin Zhang. 2013. Nonnegative Matrix Factorization: A Comprehensive Review. *IEEE Trans. Knowl. Data Eng.* 25, 6 (2013), 1336–1353.
[40] Junzhou Zhao, Shuo Shang, Pinghui Wang, John C. S. Lui, and Xiangliang Zhang. 2019. Submodular Optimization over Streams with Inhomogeneous Decays. In *AAAI*. 5861–5868.