Fairness in Streaming Submodular Maximization over a Matroid Constraint

Marwa El Halabi Samsung - SAIT AI Lab, Montreal
Federico Fusco Sapienza University of Rome
Ashkan Norouzi-Fard Google Research
Jakab Tardos Google
Jakub Tarnawski Microsoft Research

Monotone Submodular Function

Ground Set

$$f : 2^w \rightarrow R^+$$

Monotone Submodular Function

Ground Set

$$f : 2^{w} \rightarrow R^{+}$$

$$0 \leq f(e \mid X)$$

Any set X and element e

Monotone Submodular Function

Ground Set

$$f : 2^{w} \rightarrow R^{+}$$

Diminishing Returns

$$f(e \mid X \cup Y) \leq f(e \mid X)$$

Any sets X, Y and element e

Monotone Submodular Function

Influence Maximization

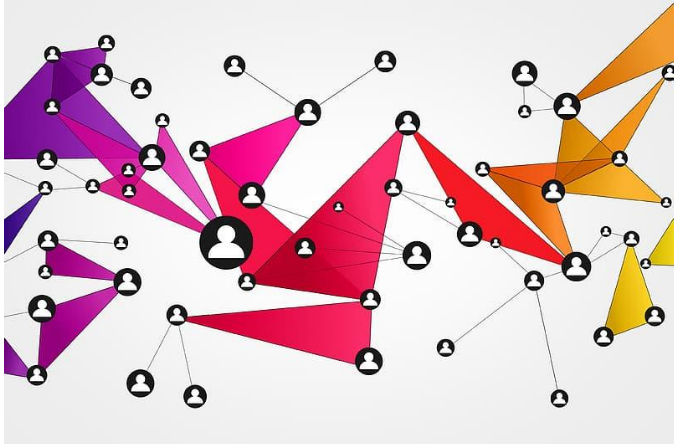


Image Summarization

Maximizing f under Cardinality Constraint

$$\max_{|S| \leq k} f(S)$$

Cardinality Constraint

Maximizing f under Matroid Constraint

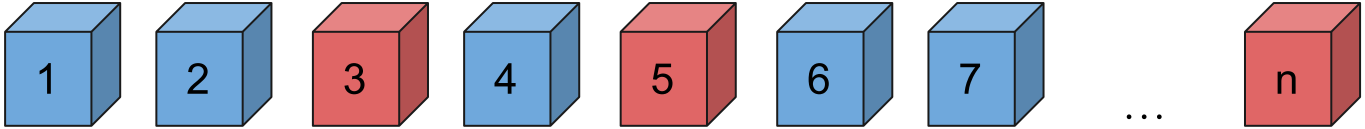
$$\max_{S \in \mathcal{M}} f(S)$$

Matroid Constraint

Fair Streaming Setting

- Elements arrive on a stream.
- We have limited memory.
- Each element has a color.
- We are given lower and upper bound constraint for each color.
    - The minimum and maximum number of elements that we can pick from each color.

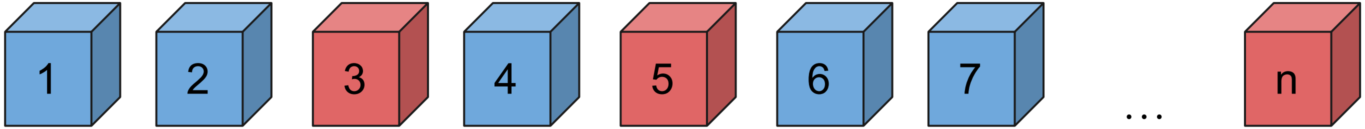
Fair Streaming Setting



Find a solution such that

1. Number of blue elements in range [1, 2]
2. Number of red elements is in range [0, 3]
3. The solution belongs to a matroid

Fair Streaming Setting



Find a solution such that

1. Number of blue elements in range [1, 2]
2. Number of red elements is in range [0, 3]
3. The solution belongs to a matroid

The bounds are not constants

Our Results

A tight $\frac{1}{2}$-approximation algorithm with exponential memory

Our Results

A tight $\frac{1}{2}$-approximation algorithm with exponential memory

Theorem 1.1. *For any constant* $\eta \in (0, 1/2)$, *there exists a one-pass streaming* $(1/2 - \eta)$*-approximation algorithm for FMMSM that uses* $2^{O(k^2 + k \log C)} \cdot \log \Delta$ *memory, where* $\Delta = \frac{\max_{e \in V} f(e)}{\min_{\{e \in V \mid f(e) > 0\}} f(e)}$.

Memory Usage

What if we want to use less memory?

Our Results

It is not possible to use efficient memory even if we make multiple passes

Our Results

It is not possible to use efficient memory even if we make multiple passes

Theorem 1.2. *Any (randomized) $o(\sqrt{\log C})$-pass streaming algorithm that determines the existence of a feasible solution for FMMSM with probability at least $2/3$ requires $\max(k, C)^{2-o(1)}$ memory.*

Our Results

If we violate the lower bounds we can get a high solution with quadratic memory usage in two passes over the stream

Our Results

If we violate the lower bounds we can get a high solution with quadratic memory usage in two passes over the stream

Theorem 1.3. *There exists a two-pass streaming algorithm for FMMSM that runs in polynomial time, uses $O(k \cdot C)$ memory, and outputs a set S such that (i) S is independent, (ii) it holds that $\lfloor \ell_c/2 \rfloor \leq |V_c \cap S| \leq u_c$ for any color $c = 1, \ldots, C$, and (iii) $f(S) \geq \mathrm{OPT}/11.656$.*

Our Results

Even with more violations, it is not possible to get efficient algorithms.

Our Results

Even with more violations, it is not possible to get efficient algorithms.

Theorem 1.4. *There is no one-pass semi-streaming algorithm that determines the existence of a feasible solution for FMMSM with probability at least* $2/3$*, even if it is allowed to violate the fairness lower bounds by a factor of* 2 *and completely ignore the fairness upper bounds.*

Overview of Our Algorithm

First pass: Find any feasible solution

Overview of Our Algorithm

First pass: Find any feasible solution

1. Find a solution in matroid for each color independently.
2. Find a feasible solution by combining these solutions.

Overview of Our Algorithm

First pass: Find any feasible solution

1. Find a solution in matroid for each color independently.
2. Find a feasible solution by combining these solutions.

Algorithm 1 FAIR-RESERVOIR

1: $I_c \leftarrow \emptyset$ for all $c = 1, ..., C$
2: **for** each element e on the stream **do**
3: Let c be the color of e
4: **If** $I_c + e \in \mathcal{I}$ **then** $I_c \leftarrow I_c + e$
5: Consider the partition matroid $\mathcal{I}_C$ on V defined in (1)
6: $S \leftarrow$ a max-cardinality subset of $\bigcup_c I_c$ in $\mathcal{I} \cap \mathcal{I}_C$ (Lemma 2.2)
7: **Return** S

Overview of Our Algorithm

Second pass: Improve the quality of the solution

Overview of Our Algorithm

Second pass: Improve the quality of the solution

1. Divide the solution into two so that the lower bounds are violated by at most a factor two.
2. Extend these two sets by adding good elements to them without violating upper bounds and matroid constraint.
3. Return the best solution.

Overview of Our Algorithm

Second pass: Improve the quality of the solution

1. Divide the solution into two so that the lower bounds are violated by at most a factor two.
2. Extend these two sets by adding good elements to them without violating upper bounds and matroid constraint.
3. Return the best solution.

How can we do this?

→ Matroid intersection

Overview of Our Algorithm

1. Divide the solution into two so that the lower bounds are violated by at most a factor two.
2. Extend these two sets by adding good elements to them without violating upper bounds and matroid constraint.
3. Return the best solution.

Algorithm 2 FAIR-STREAMING

Input: Set S from FAIR-RESERVOIR and routine $\mathcal{A}$
$S_1 \leftarrow \emptyset$, $S_2 \leftarrow \emptyset$
for e in S **do**
  Let c be the color of e
  if $|S_1 \cap V_c| < |S_2 \cap V_c|$ **then**
    $S_1 \leftarrow S_1 + e$
  else
    $S_2 \leftarrow S_2 + e$
Define matroids $\mathcal{I}^C, \mathcal{I}_1, \mathcal{I}_2$ as in Equations (2) and (3)
Run two copies of $\mathcal{A}$, one for matroids $\mathcal{I}^C, \mathcal{I}_1$ and one for matroids $\mathcal{I}^C, \mathcal{I}_2$, and let S'_1 and S'_2 be their outputs
for $i = 1, 2$ **do**
  for e in S_i **do**
    Let c be the color of e
    If $|S'_i \cap V_c| < u_c$ **then** $S'_i \leftarrow S'_i + e$
Return $S' = \arg\max(f(S'_1), f(S'_2))$

Open Directions

1. Other constraints
    - Knapsack constraint

2. Single pass algorithm with efficient memory

3. Stronger impossibility results