Fast Streaming Algorithms for k-Submodular Maximization under a Knapsack Constraint

Canh V. Pham
ORLab, Faculty of Computer Science
Phenikaa University, Hanoi, Vietnam
canh.phamvan@phenikaa-uni.edu.vn

Dung K.T. Ha
Faculty of Information Technology,
VNU University of Engineering and Technology, Hanoi, Vietnam
20028008@vnu.edu.vn

Huan X. Hoang
Faculty of Information Technology,
Halong University, Quang Ninh, Vietnam
and VNU University of Engineering and Technology
Ha Noi, Vietnam
huanxhoang@vnu.edu.vn

Tan D. Tran
People's Security Academy, Hanoi, Vietnam
tantd.hvan@gmail.com

***Abstract*—This paper proposes two fast streaming algorithms for the problem of k-submodular maximization over the ground set of n elements under the knapsack constraint which is important and popular in combinatorial optimization and machine learning. Our algorithms are the first ones that provide constant-approximation ratios within $O(nk)$ query complexity. The first algorithm is a single-pass streaming algorithm that returns a $1/10$-approximation solution, the second one is a multi-pass streaming algorithm and improves the approximation ratio to nearly $1/4$. Although these ratios are simply near to the state-of-the-art algorithms yet the number of queries can diminish by a large factor. We further investigate the performance of our algorithms by directing several experiments on instances of the issue: Influence Maximization and Sensor Placement. The outcomes confirm that our algorithms not only methodology in the quality arrangement of the cutting edge techniques including streaming and non-streaming algorithms yet in addition significantly reduce the number of queries.**

***Index Terms*—Approximation algorithm, k-submodular maximization, knapsack constraint, streaming algorithm.**

I. Introduction

The issue of maximizing k-submodular functions under constraints has drawn in a ton of considerations recently due to the wide reach application in various domains such as influence maximization in social networks [1–4], sensor placement [1–3], feature selection [5] and information coverage maximization [3], etc. Given a finite ground set V and an integer number k, we define $[k] = \{1, 2, \ldots, k\}$ and $(k+1)^V = \{(V_1, V_2, \ldots, V_k) | V_i \subseteq V, \forall i \in [k], V_i \cap V_j = \emptyset, \forall i \neq j\}$ be a family of k disjoint sets, called the k-**set**. A function $f : (k+1)^V \mapsto \mathbb{R}_+$ is k-**submodular** iff for any $\mathbf{x} = (X_1, X_2, \ldots, X_k)$ and $\mathbf{y} = (Y_1, Y_2, \ldots, Y_k) \in (k+1)^V$, we have:

$$f(\mathbf{x}) + f(\mathbf{y}) \geq f(\mathbf{x} \sqcap \mathbf{y}) + f(\mathbf{x} \sqcup \mathbf{y}) \tag{1}$$

where

$$\mathbf{x} \sqcap \mathbf{y} = (X_1 \cap Y_1, \ldots, X_k \cap Y_k)$$

and

$$\mathbf{x} \sqcup \mathbf{y} = (Z_1, \ldots, Z_k), \text{ where } Z_i = X_i \cup Y_i \setminus (\bigcup_{j \neq i} X_j \cup Y_j)$$

The problem was first studied with unconstrained [6, 7] and further examined under a few typical constraints like cardinality [1–4], knapsack [8, 9] and matroid [10, 11]. In such a constraint list, the knapsack constraint is one of the most natural and general since it captures limitations on the budget, time, or size of the elements in numerous genuine applications.

Under the knapsack constraint, each element e assigns a positive cost $c(e)$. Given a limited budget B, **the k-Submodular Maximization under a Knapsack constraint (kSMK) problem** asks to find a k-set $\mathbf{s} = (S_1, S_2, \ldots, S_k)$ with total cost $c(\mathbf{s}) = \sum_{i \in [k]} \sum_{e \in S_i} c(e) \leq B$ so that $f(\mathbf{s})$ is maximized. The kSMK was first concentrated by Wang *et al.* [9, 12] in which the authors proposed a $1/2 - 1/(2e)$-approximation algorithm by utilizing the Greedy approach. Nonetheless, the algorithm needs an expensive query complexity of $O(n^4k^3)$ that makes it impractical for modern instance sizes. Streaming algorithms are an effective method to find solutions under the state of the limitation of computer memory while dealing with large-scale issues. A streaming algorithm scans the data once or a few times in which it receives every element in the ground set sequentially and keeps just a small number of the element in memory at any point of time. Therefore it not only optimizes stored memory, running time, or a number of queries but also produces a guaranteed solution. However, existing literature has largely focused on developing streaming algorithms for submodular function optimizations and just a couple of works are focusing on k-submodular function maximization under constraints due to the intrinsic difference between submodularity and k-submodularity [4, 8, 13]. To our knowledge, the authors in [8]

first proposed two streaming algorithms for kSMK that obtain a constant-approximation ratio within $O(kn\log(n))$ queries.

In specific applications, the consistent increment of input data and objective function f has a costly computational expense. For example, about Influence Maximization in a social network, calculating exactly the influence spread of a given set of users is #P-hard [14, 15] and the number of users and links in social networks has exhibited exponential growth recent years [16, 17]. Therefore, it is crucial to devise practical streaming algorithms that accomplish a guaranteed solution as well as algorithm lessen the number of queries.

A. Our contribution

In this paper we propose two streaming algorithms that achieve constant approximation ratios yet just require $O(kn)$ query complexity. To our knowledge, that is the most reduced computational cost of any constant ratio approximation algorithm and plays an significant role in finding near-optimal solutions for applications as the expense to evaluate the function f might be costly. In general, our contributions are as per the following:

- We first propose a deterministic streaming, named **FSA** (Algorithm 1), a $1/10$-approximation algorithm that requires only one pass over the ground set and kn queries.
- We further propose a multi-pass streaming algorithm, named **IFSA** (Algorithm 2) that scans the ground set $O(1/\epsilon)$ times, achieves an approximation ratio $1/4-\epsilon$, and requires $O(kn/\epsilon)$ query complexity for any accuracy parameter $\epsilon > 0$.
- We conduct comprehensive experiments to investigate the performance of our algorithms in two applications of kSMK, k-topic Influence Maximization and k-type Sensor Placement. Experimental results have shown that our algorithms not just need a lot more modest number of queries than that of the cutting-edge non-streaming ones (mentioned in Table I) yet additionally return comparable solutions in terms of quality.

Table I compares our algorithms with several other ones for kSMK.

B. Related work

k-submodular maximizations. Studying on k-submodular functions was first presented by Singh *et al.* [5] who focused on bisubmodular maximization that was k-submodular maximization with the case $k = 2$. From that point forward, more works have focused on the issue of general k. With $k = 1$, the issue becomes submodular maximization. As submodular maximization is NP-hard, k-submodular maximization is also NP-hard. Ward *et al.* [6] then studied about unconstrained maximization of k-submodular function and devised a deterministic Greedy algorithm which gave an approximation ratio of $1/3$. Later on, the authors in [7] presented a random Greedy approach which improved the approximation ratio to $\frac{k}{2k-1}$ by introducing a probability distribution to select a larger marginal element with higher probability. Authors in [18] eliminated the random told in [7], however, the quantity of queries expanded to $O(n^2k^2)$. The unconstrained k-submodular maximization was additionally concentrated in online settings [19].

Researchers have further studied about constrained maximization of k-submodular function. Oshaka *et al.* [1] first worked with monotone k-submodular maximization with size constraints. Utilizing the Greedy algorithm, they proposed a $1/2$-approximation algorithm for the total size constraint and a $1/3$-approximation algorithm for the individual size one and in $O(knB)$ time [1]. The authors in [3] then further proposed a multi-objective evolutionary algorithm for the monotone k-submodular maximization problem under the total size constraint. Their algorithm provided a $1/2$-approximation solution within $8eB$ iterations, each might take a query complexity of $O(kn\log^2 B)$ in expectation. Zheng *et al.* [20] tackled the issue of estimated maximizing k-submodular functions subject to size constraints by offering an approximated k-submodular function, F, of the objective function. Besides, the k-submodular maximization has been investigated under richer constraints. Authors in [10] showed a Greedy algorithm that could return an approximation ratio of $1/2$ for the problem under a matroid constraint. Another algorithm with the same approximation ratio by using the differentially private continuous Greedy method was proposed in [11]. Wang *et al.* [9, 12] first proposed a $(1/2-1/(2e))$-approximation algorithm for the kSMK that inspired by the Greedy algorithm in [21]. This algorithm, however, requires an expensive query complexity of $O(n^4k^3)$ and therefore it is difficult to apply to medium-sized instances even though one can compute the objective function f in $O(1)$ time.

Streaming algorithms. Submodularity is a property of set functions with deep theoretical and practical consequences. Nemhauser *et al.* [22] showed a greatly important thing in data mining that a simple Greedy algorithm created several solutions competitively with the optimal solution. To our best knowledge, proposing algorithms for the submodularity of set functions has enhanced on the basis of Greedy frequently. Using Greedy algorithms, that require random access to the data, can easily be applied if the data fits in the main memory. However, it is impractical when data explosions happen. Therefore, individuals consider streaming fashion where at any point of time the algorithm has access only to a small fraction of data stored in primary memory. This approach not only avoids the requirement of vast amounts of random-access memory but also provides predictions in a timely manner based on the data seen so far, facilitating real-time analytics. Hence, streaming algorithm becomes one of the productive methods for solving submodular maximization problems under various kinds of constraints such as cardinality [23–26], knapsack [27], k-set [28] and matroid [29] yet it is not potential to directly be applied to our problem because of inherent contrasts between submodularity and k-submodularity.

For the k-submodular objective function, the algorithms in [6, 7] work as single-pass streaming ones. The idea of [7]

[1] $B = \sum_{i\in[k]} B_i$ in the case of individual size constraints.

Reference	Num. of passes	Appr. ratio	Query Comp.	Deterministic?
FSA (Algorithm 1, this paper)	1	$1/10$	$O(kn)$	yes
IFSA (Algorithm 2, this paper)	$O(1/\epsilon)$	$1/4-\epsilon$	$O(kn/\epsilon)$	yes
Deterministic Streaming[8]	1	$1/4-\epsilon$	$O(kn\log(n)/\epsilon)$	yes
Random Streaming[8]	1	$k/(4k-1)-\epsilon$	$O(kn\log(n)/\epsilon)$	no
Greedy[9]	$O(n^3k^3)$	$1/2-1/(2e)$	$O(n^4k^3)$	yes

TABLE I: Algorithms for kSMK

is utilizing random selection while the others have improved this idea by submitting a new distribution that can help to select an element with various costs and afterward establishes the relationship between the current solution and the optimal. Nguyen *et al.* [2] first devised streaming algorithms for the k-submodular maximization issue subjected to the total size constraint under noises. They proposed two streaming algorithms that provided the approximation ratio of $O(\epsilon(1-\epsilon)^{-2}B)$ when f was monotone and $O(\epsilon(1-\epsilon)^{-3}B)$ when f was non-monotone. Recently, Pham *et al.* [8] proposed two single-pass streaming algorithms for the k-submodular maximization under the budget constraint within $O(nk\log(n)/\epsilon)$. Note that, the kSMK is a special case of the algorithm with $\beta = 1$ and therefore these algorithms can obtain the approximation ratios of $1/4-\epsilon$ and $k/(4k-1)-\epsilon$ (in expectation). Our IFSA can return the same approximation ratio with the deterministic streaming in [8] but requires fewer the number of queries by a factor of $\log(n)$.

C. Organization

The rest of the paper is organized as follows: The notations and properties of k-submodular functions are presented in Section II. Section III presents our algorithms and theoretical analysis. The extensive experiments are shown in Section IV. Finally, we conclude this work in Section V.

II. Preliminaries

Notations. We use following notations throughout the paper: Given a finite set V and an integer k, as mentioned above, the definitions of $[k]$ and $(k+1)^V$ have referred in Section I. We define $supp_i(\mathbf{x}) = X_i$, $supp(\mathbf{x}) = \cup_{i\in[k]}X_i$, X_i as i-**th set of** $\mathbf{x}$ and an empty k-set $\mathbf{0} = (\emptyset,\ldots,\emptyset)$.

For $\mathbf{x} = (X_1, X_2, \ldots, X_k), \mathbf{y} = (Y_1, Y_2, \ldots, Y_k) \in (k+1)^V$, we set if $e \in X_i$ then $\mathbf{x}(e) = i$ and i is called the **position** of e, otherwise $\mathbf{x}(e) = 0$. Adding an element $e \notin supp(\mathbf{x})$ into X_i can be represented by $\mathbf{x} \sqcup (e,i)$. When $X_i = \{e\}$, and $X_j = \emptyset, \forall j \neq i$, $\mathbf{x}$ is denoted by (e,i). We denote by $\mathbf{x} \sqsubseteq \mathbf{y}$ iff $X_i \subseteq Y_i\ \forall i \in [k]$.

The objective function. The function $f : (k+1)^V \mapsto \mathbb{R}_+$ is k-**submodular** iff for any $\mathbf{x} = (X_1, X_2, \ldots, X_k)$ and $\mathbf{y} = (Y_1, Y_2, \ldots, Y_k) \in (k+1)^V$, we have:

$$f(\mathbf{x}) + f(\mathbf{y}) \geq f(\mathbf{x} \sqcap \mathbf{y}) + f(\mathbf{x} \sqcup \mathbf{y}) \tag{2}$$

where

$$\mathbf{x} \sqcap \mathbf{y} = (X_1 \cap Y_1, \ldots, X_k \cap Y_k)$$

and

$$\mathbf{x} \sqcup \mathbf{y} = (Z_1, \ldots, Z_k), \text{ where } Z_i = X_i \cup Y_i \setminus (\bigcup_{j\neq i} X_j \cup Y_j)$$

In this work, we consider f is *monotone*, i.e, for any $\mathbf{x} \in (k+1)^V, e \notin supp(\mathbf{x})$ and $i \in [k]$, we have:

$$\begin{aligned}\Delta_{(e,i)}f(\mathbf{x}) =& f(X_1, \ldots, X_{i-1}, X_i \cup \{e\}, X_{i+1}, \ldots, X_k) \\ &- f(X_1, \ldots, X_k) \geq 0\end{aligned}$$

From [6], the k-submodularity of f implies the *orthant submodularity*, i.e,

$$\Delta_{(e,i)}f(\mathbf{x}) \geq \Delta_{(e,i)}f(\mathbf{y}) \tag{3}$$

for any $\mathbf{x}, \mathbf{y} \in (k+1)^V, e \notin supp(\mathbf{x}), \mathbf{x} \sqsubseteq \mathbf{y}$ and $i \in [k]$; and the *pairwise monotonicity*, i.e, for any $i, j \in [k], i \neq j$:

$$\Delta_{(e,i)}f(\mathbf{x}) + \Delta_{(e,j)}f(\mathbf{x}) \geq 0 \tag{4}$$

We only consider $k \geq 2$ because if $k = 1$, the k-submodular function becomes the submodular function. In the context of the kSMK problem, we assume that each element e has the cost $c(e)$ satisfies $c(e) \leq B$ otherwise, we can remove it.

III. Proposed Algorithms

In this section, we introduce two deterministic streaming algorithms for kSMK. The first algorithm is a single-pass streaming algorithm, named *Fast Streaming Approximation* (**FSA**) that has an approximation ratio of $1/10$ and takes nk queries. Although this approximation ratio is small, it is the **first algorithm** that gives a constant approximation ratio within only $O(kn)$ queries. Our second algorithm improves the approximation ratio from $1/10$ to $1/4-\epsilon$ by reusing the solution of the first algorithm to give an appropriate range to bound the optimal value opt. Along with this, it scans the ground set $O(1/\epsilon)$ times and integrates with the decreasing threshold strategy to get the near-optimal solution.

A. Fast Streaming Approximation algorithm

The main idea of our **FSA** algorithm is that (1) divides the ground set V into two subsets: the **first subset** contains the elements that have the costs larger than $B/2$, the **second one** contains the remaining, and (2) seeks and combines near-optimal solutions on two above subsets.

Specifically, the algorithm first receives an instance (V, f, k, B) of kSMK and initiates a candidate solution $\mathbf{s}$ as $\mathbf{0}$ and a tuple (e_m, i_m) as $(\emptyset, 1)$. The tuple (e_m, i_m) is to update the optimal solution on the first subset while the candidate solution $\mathbf{s}$ is to find a near-optimal solution on the second. For each incoming element e, the algorithm finds "the best" position i_e in terms of the set i in k sets that returns the highest value $f((e, i_e))$. If its cost is greater than $B/2$ the algorithm updates (e_m, i_m) as the best solution of the current first subset (line 5). Otherwise, it adds the tuple (e, i_e) into $\mathbf{s}$

if $\Delta_{(e,i_e)}f(\mathbf{s}) \geq c(e)f(\mathbf{s})/B$. At the end of the main loop, the algorithm selects a k-set $\mathbf{s}'$ as the set of last j tuples adding into $\mathbf{s}$ with the maximum total cost nearest to B (line 12). Finally, the algorithm returns the final solution $\mathbf{s}^f$ as the best one between (e_m, i_m) and $\mathbf{s}'$. The details of the algorithm are fully presented in Algorithm 1.

Algorithm 1: FSA

Input: V, f, k, $B > 0$.
Output: A solution $\mathbf{s}$

1: $\mathbf{s} \leftarrow \mathbf{0}$; $(e_m, i_m) \leftarrow (\emptyset, 1)$
2: **foreach** $e \in V$ **do**
3: $\quad i_e \leftarrow \arg\max_{i\in[k]} f((e,i))$
4: $\quad$ **if** $c(e) > B/2$ **then**
5: $\quad\quad (e_m, i_m) \leftarrow \arg\max_{(e',i')\in\{(e_m,i_m),(e,i_e)\}} f((e',i'))$
6: $\quad$ **else**
7: $\quad\quad$ **if** $\Delta_{(e,i_e)}f(\boldsymbol{s}) \geq c(e)f(\boldsymbol{s})/B$ **then**
8: $\quad\quad\quad \mathbf{s} \leftarrow \mathbf{s} \sqcup (e, i_e)$
9: $\quad\quad$ **end**
10: $\quad$ **end**
11: **end**
12: $\mathbf{s}' \leftarrow \arg\max_{\mathbf{s}_j : j\leq t, c(\mathbf{s}_j)\leq B} c(\mathbf{s}_j)$, where $t = |supp(\mathbf{s})|$ and $\mathbf{s}_j = \{(e_{t-j}, i_{t-j}), (e_{t-j+1}, i_{t-j+1}), \ldots, (e_t, i_t)\}$ is the last j tuples added into $\mathbf{s}$.
13: $\mathbf{s}^f \leftarrow \arg\max_{\mathbf{s}\in\{(e_m,i_m),\mathbf{s}'\}} f(\mathbf{s})$
14: **return** $\mathbf{s}^f$

At a high level, our algorithm resembles the "divide and conquer" strategy in which it uses an appropriate subset division based on the costs of elements. Dividing the ground set brings the effect when both the optimal solution on the first subset is explored in the linear time since feasible solutions have at most one element and the approximation solution on the second one can be found in linear time. For the second subset, we were inspired by the suggestion from the above idea of Kuhnle *et al.* [26] in which elements with marginal gains that are over the *ratio between the objective value of the current solution and the limited cost*, will be kept and the remaining is released. Their idea is powerful in diminishing the number of queries of a constant factor approximation algorithm. However, to deal with the cost and the k-submodular function, we need to make a non-trivial analysis to give an approximation ratio. In the following, we analyze the theoretical guarantee of the Algorithm 1. We first define the notations as follows:

- $V_1 = \{e \in V : c(e) > B/2\}, V_2 = \{e \in V : c(e) \leq B/2\}$.
- $\mathbf{o}$ is an optimal solution of the problem over V and the optimal value $\mathsf{opt} = f(\mathbf{o})$.
- $\mathbf{o}'_1 = \{(e, \mathbf{o}(e)) : e \in V_1\}, \mathbf{o}'_2 = \{(e, \mathbf{o}(e)) : e \in V_2\}$.
- $\mathbf{o}_1$ is an optimal solution of the problem over V_1.
- $\mathbf{o}_2$ is an optimal solution of the problem over V_2.
- (e_j, i_j) as the j-th element added of the main loop of the algorithm.
- $\mathbf{s}^t = \{(e_1, i_1), \ldots, (e_t, i_t)\}$ the k-set $\mathbf{s}$ after ending the main loop, $t = |supp(\mathbf{s})|$.
- $\mathbf{s}^j = \{(e_1, i_1), \ldots, (e_j, i_j)\}$: the k-set $\mathbf{s}$ (in the main loop) after adding j elements, $j \leq t$.
- $\mathbf{s}_j = \{(e_{t-j}, i_{t-j}), (e_{t-j+1}, i_{t-j+1}), \ldots, (e_t, i_t)\}$ is the set of last j elements added into $\mathbf{s}$.
- $\mathbf{o}_2^j = (\mathbf{o}_2 \sqcup \mathbf{s}^j) \sqcup \mathbf{s}^j$.
- $\mathbf{o}_2^{j-1/2} = (\mathbf{o}_2 \sqcup \mathbf{s}^j) \sqcup \mathbf{s}^{j-1}$.
- $\mathbf{s}^{j-1/2}$: If $e_j \in supp(\mathbf{o}_2)$, then $\mathbf{s}^{j-1/2} = \mathbf{s}^{j-1} \sqcup (e_j, \mathbf{o}_2(e_j))$. If $e_j \notin supp(\mathbf{o}_2)$, $\mathbf{s}^{j-1/2} = \mathbf{s}^{j-1}$.
- $\mathbf{u}^t = \{(u_1, i_1), (u_2, i_2), \ldots, (u_r, i_r)\}$ is a set of elements that are in $\mathbf{o}_2^t$ but not in $\mathbf{s}^t$, $r = |supp(\mathbf{u}^t)|$.
- $\mathbf{u}_l^t = \mathbf{s}^j \sqcup \{(u_1, i_1), (u_2, i_2), \ldots, (u_l, i_l)\}, \forall 1 \leq l \leq r$ and $\mathbf{u}_0^t = \mathbf{s}^t$.

Supposing that $\mathbf{s}'$ gets T last tuples in $\mathbf{s}$, i.e, $\mathbf{s}' = \mathbf{s}_T$. Denote $Q = t - T$, we have $\mathbf{s} = \mathbf{s}^Q \sqcup \mathbf{s}'$. We have the Lemmas:

Lemma 1. *$f(\boldsymbol{o}_2) - f(\boldsymbol{o}_2^j) \leq f(\boldsymbol{s}^j)$ for all $0 \leq j \leq t$.*

Proof: We have: $\mathbf{o}_2^0 = (\mathbf{o}_2 \sqcup \mathbf{s}^0) \sqcup \mathbf{s}^0$. Hence, $f(\mathbf{o}_2) = f(\mathbf{o}_2^0)$. For all $0 \leq j \leq t$, we have:

$$f(\mathbf{o}_2) - f(\mathbf{o}_2^j) = \sum_{i=1}^{j} (f(\mathbf{o}_2^{i-1}) - f(\mathbf{o}_2^i)) \tag{5}$$

$$\leq \sum_{i=1}^{j} (f(\mathbf{o}_2^{i-1}) - f(\mathbf{o}_2^{i-1/2})) \tag{6}$$

$$\leq \sum_{i=1}^{j} (f(\mathbf{s}^{i-1/2}) - f(\mathbf{s}^{i-1})) \tag{7}$$

$$\leq \sum_{i=1}^{j} (f(\mathbf{s}^{i}) - f(\mathbf{s}^{i-1})) \tag{8}$$

$$\leq f(\mathbf{s}^j) \tag{9}$$

where the inequality (6) due to the monotoncity of f, the inequality (7) due to the k-submodularity of f and the inequality (8) due to the selection of the algorithm. The proof is completed. ■

Lemma 2. *$f(\boldsymbol{s}') \geq f(\boldsymbol{s}^t)/3$.*

Proof. If $c(\mathbf{s}) \leq B$, $\mathbf{s}' = \mathbf{s}$ and the Lemma holds. Therefore, we just consider the case $c(\mathbf{s}) > B$. By selecting a tuple (e, i_e) into $\mathbf{s}$ in the main loop and the monotonicity of f, we have:

$$f(\mathbf{s}^t) - f(\mathbf{s}^Q) = \sum_{j=Q+1}^{T} \Delta_{(e_j,i_j)} f(\mathbf{s}_{j-1}) \tag{10}$$

$$\geq \sum_{j=Q+1}^{T} c(e_j) f(\mathbf{s}_{j-1})/B \tag{11}$$

$$\geq \sum_{j=Q+1}^{T} c(e_j) f(\mathbf{s}^Q)/B \tag{12}$$

$$\geq c(\mathbf{s}') f(\mathbf{s}^Q)/B \tag{13}$$

Since $\mathbf{s}'$ is chosen from $\mathbf{s}^t$ so that the total cost of $\mathbf{s}$ is closest to B and each element $e \in supp(\mathbf{s})$ has the cost at most $B/2$ thus

$$c(\mathbf{s}') > B - B/2 \geq B/2$$

It implies that $c(\mathbf{s}')f(\mathbf{s}^Q)/B \geq f(\mathbf{s}^Q)/2$. Hence $f(\mathbf{s}^Q) \leq 2f(\mathbf{s}^t)/3$. In the other hand, due to the k-submodularity of f we have $f(\mathbf{s}^t) \leq f(\mathbf{s}^Q) + f(\mathbf{s}')$. Thus

$$f(\mathbf{s}') \geq f(\mathbf{s}^t) - f(\mathbf{s}^Q) \geq f(\mathbf{s}^t)/3 \tag{14}$$

The proof is completed. □

Lemma 3. $f(\boldsymbol{o}_2^t) \leq 2f(\boldsymbol{s}^t)$.

Proof: We let $\mathbf{u}^t = \{(u_1, j_1), (u_2, j_2), \dots (u_r, j_r)\}$ as the set of elements in $\mathbf{o}_2^t$ but not in $\mathbf{s}^t$ where $r = |supp(\mathbf{u}^t)|$. It means the elements in $\mathbf{u}^t$ do not pass the condition in Line 7 of the Algorithm 1 and $c(\mathbf{u}^t) \leq B$. Denote by $\mathbf{s}^{t_{u_i}}$ as $\mathbf{s}$ right before u_i arrives. We have:

$$\begin{aligned}
f(\mathbf{o}_2^t) - f(\mathbf{s}^t) &= f(\mathbf{u}^t \sqcup \mathbf{s}^t) - f(\mathbf{s}^t) = \sum_{i=1}^{r}(f(\mathbf{u}_i^t) - f(\mathbf{u}_{i-1}^t)) \\
&\leq \sum_{i=1}^{r}(f(\mathbf{s}^{t_{u_i}} \sqcup (u_i, j_i)) - f(\mathbf{s}^{t_{u_i}}) && (15)\\
&\leq \sum_{i=1}^{r}\Delta_{(u_i,j_i)}f(\mathbf{s}^{t_{u_i}}) && (16)\\
&\leq \sum_{i=1}^{r}c(u_i)f(\mathbf{s}^{t_{u_i}})/B && (17)\\
&\leq \sum_{i=1}^{r}c(u_i)f(\mathbf{s}^t)/B && (18)\\
&\leq c(\mathbf{u}^t)f(\mathbf{s}^t)/B \leq f(\mathbf{s}^t) && (19)
\end{aligned}$$

where the inequality (15) due to the k-submodularity of f, the inequality (16) due to the definition of $\mathbf{s}^{t_{u_i}}$, the inequality (17) due to the selection of the algorithm and the inequality (18) due to the monotoncity of f. Thus, we have: $f(\mathbf{o}_2^t) \leq 2f(\mathbf{s}^t)$. ■

Lemma 4. $f(\boldsymbol{s}') \geq f(\boldsymbol{o}_2)/9$.

Proof. Applying the Lemmas 1, 3, with $j = t$, we have

$$\begin{aligned}
f(\mathbf{o}_2) - f(\mathbf{s}^t) &= f(\mathbf{o}_2) - f(\mathbf{o}_2^t) + f(\mathbf{o}_2^t) - f(\mathbf{s}^t) && (20)\\
&\leq f(\mathbf{s}^t) + f(\mathbf{o}_2^t) - f(\mathbf{s}^t) = f(\mathbf{o}_2^t) && (21)\\
&\leq 2f(\mathbf{s}^t) && (22)
\end{aligned}$$

Thus $f(\mathbf{s}^t) \geq f(\mathbf{o}_2)/3$. Combine with the Lemma 2, we have $f(\mathbf{s}') \geq f(\mathbf{o}_2)/9$. □

Theorem 1. *Algorithm 1 is a single-pass streaming algorithm, returns an approximation ratio of $\frac{1}{10}$ and takes nk queries.*

Proof. The algorithm takes only one time to scan over the ground set and each element e has k queries to find the position i_e. Therefore the number of queries is nk. We now prove the approximation ratio of the algorithm. We have:

$$\begin{aligned}
f(\mathbf{o}) &\leq f(\mathbf{o}_1') + f(\mathbf{o}_2') && (23)\\
&\leq f(\mathbf{o}_1) + f(\mathbf{o}_2) && (24)\\
&\leq f((e_m, i_m)) + 9f(\mathbf{s}') \leq 10f(\mathbf{s}^f) && (25)
\end{aligned}$$

where (23) due to the k-submodularity of f, (24) and (25) due to the definition of $\mathbf{o}_1$ and $\mathbf{o}_2$ and (e_m, i_m). The proof was completed. □

B. Improve Fast Streaming Approximation algorithm

We next propose a multi-pass streaming algorithm, named *Improved Fast Streaming Approximation* (**IFSA**) which improves the approximation ratio to $(1/4-\epsilon)$ and takes $O(kn/\epsilon)$ query complexity.

IFSA takes an instance (V, f, k, B) of kSMK and an accuracy parameter ϵ as inputs. IFSA first calls FSA as a subroutine and uses FSA's solution $\mathbf{s}_{max}$ to obtain a bound range of the optimal solution (line 1). From Theorem 1, we have $\Gamma \leq \mathsf{opt} \leq 10\Gamma$.

The algorithm consists of two loops: the outer and the inner. In each iteration of the outer loop, the algorithm initiates a candidate solution $\mathbf{s} = \mathbf{0}$. It takes one pass over the ground set to sequentially select each incoming element e if the total cost $c(\mathbf{s}) + c(e)$ does not exceed B and the ratio of marginal gain per its cost is at least θ. The values of the threshold $\theta = 5(1-\epsilon)^t\Gamma/B$ in which θ is first set to $5\Gamma/B$ and decreases by a factor of $1-\epsilon$ after each iteration. The main loop terminates if $\theta < (1-\epsilon)\Gamma/B$ and the algorithm updates the best current solution in line 11. Finally, the algorithm returns the best one between (e_m, i_m) and $\mathbf{s}_{max}$ as the final solution. The details of the algorithm are fully presented in Algorithm 2.

Algorithm 2: IFSA

Input: V, f, k, $B > 0$, $\epsilon > 0$.
Output: A solution $\mathbf{s}$

1: $\mathbf{s}_{max} \leftarrow$ result of Algorithm 1; $\Gamma \leftarrow f(\mathbf{s}_{max})$
2: $(e_{max}, i_{max}) \leftarrow \arg\max_{e\in V, i\in[k]} f((e, i))$
3: **for** $t = 0$ *to* $\lceil \log_{\frac{1}{1-\epsilon}}(10)\rceil + 1$ **do**
4: $\quad\theta = 5(1-\epsilon)^t\Gamma/B$; $\mathbf{s} \leftarrow \mathbf{0}$
5: $\quad$**for** $e \in V$ **do**
6: $\quad\quad i_e \leftarrow \arg\max_{i\in[k]} \Delta_{(e,i)}f(\mathbf{s})$
7: $\quad\quad$**if** $c(e) + c(\boldsymbol{s}) \leq B$ ***and*** $\Delta_{(e,i_e)}f(\boldsymbol{s})/c(e) \geq \theta$ **then**
8: $\quad\quad\quad\mathbf{s} \leftarrow \mathbf{s} \sqcup (e, i_e)$
9: $\quad\quad$**end**
10: $\quad$**end**
11: $\quad\mathbf{s}_{max} \leftarrow \arg\max_{\mathbf{s}'\in\{\mathbf{s}, \mathbf{s}_{max}\}} f(\mathbf{s}')$
12: **end**
13: $\mathbf{s} \leftarrow \arg\max_{\mathbf{s}'\in\{(e_{max}, i_{max}), \mathbf{s}_{max}\}} f(\mathbf{s}')$
14: **return** $\mathbf{s}$

Theorem 2. *For $0 < \epsilon < 1/4$, the Algorithm 2 is a multi-pass streaming algorithm, returns an approximation ratio of $1/4-\epsilon$,*

within $O(\log(10)/\epsilon)$ *passes and* $2kn + kn(\lceil \log_{\frac{1}{1-\epsilon}}(10) \rceil + 1)$ *queries.*

Proof. The algorithm needs $2nk$ queries to call FSA algorithm and finds (e_{max}, i_{max}). The algorithm uses $\lceil \log_{\frac{1}{1-\epsilon}}(10) \rceil + 1$ passes over over the ground set in which each takes nk queries. Combine all tasks, we obtain the query complexity of the algorithm.

By Theorem 1, we have $\Gamma \leq \mathsf{opt} \leq 10\Gamma$. Therefore, there exists an integer number $t \in [0, \lceil \log_{\frac{1}{1-\epsilon}}(10) \rceil] + 1]$ so that $(1-\epsilon)\mathsf{opt} \leq 10(1-\epsilon)^t\Gamma \leq \mathsf{opt}$. Assume that $\mathbf{s}_\theta = \{(e_1, i_1), (e_2, i_2), \ldots, (e_q, i_q)\}$ is the candidate solution (the solution after ending the inner loop) with respect to $\theta = 5(1-\epsilon)^t\Gamma/B$ and $\mathbf{s}_\theta^j = \{(e_1, i_1), (e_2, i_2), \ldots, (e_j, i_j)\}, j \leq q$. We have:

$$f(\mathbf{s}_\theta^j) = \sum_{i=1}^{j}(f(\mathbf{s}_\theta^i) - f(\mathbf{s}_\theta^{i-1})) \geq \sum_{i=1}^{j} c(e_i)\theta = c(\mathbf{s}_\theta^j)\theta \quad (26)$$

We denote by $\mathbf{s}_\theta^e$ as $\mathbf{s}_\theta$ immediately before e is processed. We consider the following cases:

Case 1. There exists an element $o \in supp(\mathbf{o}) \setminus supp(\mathbf{s}_\theta)$ so that $\Delta_{(o,\mathbf{o}(o))} f(\mathbf{s}_\theta^o) \geq \theta$ and $c(\mathbf{s}_\theta^o) + c(o) > B$, we have:

$$\begin{aligned}
f(\mathbf{s}_{max}) &\geq \max\{f(\mathbf{s}_\theta), f((e_{max}, i_{max}))\} && (27)\\
&\geq \max\{f(\mathbf{s}_\theta^o), f((o, \mathbf{o}(o)))\} && (28)\\
&\geq \frac{f(\mathbf{s}_\theta^o) + f((o, \mathbf{o}(o)))}{2} && (29)\\
&\geq \frac{f(\mathbf{s}_\theta^o \sqcup (o, \mathbf{o}(o))}{2} = \frac{\Delta_{(o,\mathbf{o}(o))} f(\mathbf{s}_\theta^o) + f(\mathbf{s}_\theta^o)}{2} && (30)\\
&\geq \frac{\theta c(o) + \theta c(\mathbf{s}_\theta^o)}{2} \geq \frac{B\theta}{2} \geq \frac{(1-\epsilon)\mathsf{opt}}{4} && (31)\\
&\geq (\frac{1}{4} - \epsilon)\mathsf{opt} && (32)
\end{aligned}$$

Case 2. There is no such an element $o \in supp(\mathbf{o}) \setminus supp(\mathbf{s}_\theta)$. Denote $\mathbf{o}^j = (\mathbf{o} \sqcup \mathbf{s}_\theta^j) \sqcup \mathbf{s}_\theta^j$, $\mathbf{u} = \{(u_1, i_1), (u_2, i_2), \ldots, (u_r, i_r)\}$ as a set of elements that are in $\mathbf{o}$ but not in $\mathbf{s}_\theta$, $r = |supp(\mathbf{u})|$ and $\mathbf{u}_l = \mathbf{s}_\theta \sqcup \{(u_1, i_1), (u_2, i_2), \ldots, (u_l, i_l)\}, \forall 1 \leq l \leq r$ and $\mathbf{u}_0 = \mathbf{s}_\theta$. By the similar argument of Lemma 2, we also have: $f(\mathbf{o}) - f(\mathbf{s}_\theta) \leq f(\mathbf{s}_\theta) + f(\mathbf{o}^q) - f(\mathbf{s}_\theta)$ and thus:

$$\begin{aligned}
f(\mathbf{o}) - f(\mathbf{s}_\theta) &= f(\mathbf{o}) - f(\mathbf{o}^q) + f(\mathbf{o}^q) - f(\mathbf{s}_\theta) && (33)\\
&\leq f(\mathbf{s}_\theta) + \sum_{i=1}^{r}(f(\mathbf{u}_i) - f(\mathbf{u}_{i-1})) && (34)\\
&\leq f(\mathbf{s}_\theta) + \sum_{i=1}^{r} \Delta_{(u_i,j_i)} f(\mathbf{s}^{u_i}) && (35)\\
&\leq f(\mathbf{s}_\theta) + \sum_{i=1}^{r} c(u_i)\theta && (36)\\
&\leq f(\mathbf{s}_\theta) + B\theta \leq f(\mathbf{s}_\theta) + \mathsf{opt}/2 && (37)
\end{aligned}$$

where the inequality (35) due to the k-submodularity, the inequality (36) due to the definition of $\mathbf{s}_\theta^o$ for any $o \in supp(\mathbf{o}) \setminus supp(\mathbf{s}_\theta)$. We imply that $f(\mathbf{s}_\theta) \geq \mathsf{opt}/4$ in this case. By combining two above cases we obtain the proof. □

IV. Experiments

In this section, we compare the performance between our algorithms and state-of-the-art algorithms for kSMK problem listed below:

- **Greedy**: the $(1/2 - 1/(2e))$-approximation algorithm within $O(n^4k^3)$ in [9].
- **Deterministic Streaming (DS)**[2]: A streaming algorithm in [8] which returns an approximation ratio of $1/4 - \epsilon$, requires one pass and $O(kn\log(n)/\epsilon)$ queries.
- **Random Streaming (RS)**: Another streaming algorithm in [8] which returns an approximation ratio of $k/(4k-1) - \epsilon$ in expectation, requires one pass and $O(kn\log(n)/\epsilon)$ queries.

We conduct experiments on specific applications which are k-**topic Influence Maximization under knapsack constraint (kIMK)** and k-**type Sensor Placement under Knapsack constraint (kSPK)** on three major measurements: the oracle value of the objective function, the number of queries, and running time. We additionally further show the trade-off between the solution quality and the number of queries of algorithms with various settings of budget B. We also use the dataset as mentioned in [8, 13, 30] to illustrate the performance of compared algorithms (Table II). To illustrate the performance of algorithms via the above three measurements, we show some figures numbered and captioned, in which the terms Fig, K, and M stand for the term Figure, thousands, and millions, respectively.

All the implementations are on Linux machine with configurations of 2× Intel(R) Xeon(R) CPU E5-2697 v4 @2.30GHz and 10 threads ×128 GB DIMM ECC DDR4 @2400MHz.

TABLE II: The dataset

Database	#Nodes	#Edges	Types
Facebook [31]	4039	88234	directed
Hept [30]	15233	58894	directed
Enron [30]	36692	367662	directed
Intel Lab sensors[32]	56	-	-

A. k-topic Influence Maximization under Knapsack constraint

The information diffusion model, called Linear Threshold (LT) model [4, 33], was briefed and the kIMK using problem this model was defined as follows:

a) LT model: A social network is modeled by a directed graph $G = (V, E)$, where V, E represent a set of users and a set of links, respectively. Each edge $(u, v) \in E$ is assigned weights $\{w^i(u, v)\}_{i\in[k]}$, where each $w^i(u, v)$ represents how powerful u influences to v on the i-th topic. Each node $u \in V$ has a *influence threshold* with topic i, denoted by $\theta^i(u)$, which is chosen uniformly at random in $[0, 1]$. Given a seed set $\mathbf{s} = (S_1, S_2, \ldots, S_k) \in (k+1)^V$, the information propagation for topic i happens in discrete steps $t = 0, 1, \ldots$ as follows. At step $t = 0$, all nodes in S_i become active by topic i. At step

[2]The kSMK problem is a special case of the k-submodular maximization under the budget constraint in [8] with $\beta = 1$.

$t \geq 1$, a node u becomes active if $\sum_{\text{active node } v} w^i(v,u) \geq \theta^i(u)$.

The information diffusion process on topic i ends at step t if there is no new active node and the diffusion process of a topic is independent from the others. Denote by $\sigma(\mathbf{s})$ as the number of nodes which becomes active in at least one of k topics after the diffusion process of a seed k-set $\mathbf{s}$, i.e,

$$\sigma(\mathbf{s}) = \mathbb{E}[|\cup_{i\in[k]} \sigma_i(S_i)|] \tag{38}$$

where $\sigma_i(S_i)$ is a random variable representing the set of active users for topic i with the seed S_i.

b) The kIMK problem: The problem is formally defined as follows:

Definition 1 (kIMK problem)**.** Assuming that each user e has a cost $c(e)$ for every i-th topic which manifests how hard it is to initially influence the respective person for that topic. Given the budget B, the problem asks to find a seed set $\mathbf{s}$ with $c(\mathbf{s}) = \sum_{e\in S_i, i\in k} c(e) \leq B$ so that $\sigma(\mathbf{s})$ is maximal.

c) Experiment settings: We use three different databases: Facebook, Hept and Enron which are popular in Information Maximization problems [8, 13, 30] and setup the model as the recent work [4].

Since the computation of $\sigma(\cdot)$ is #P-hard [14], we adapt the sampling method in [4, 34] to give an estimation $\hat{\sigma}(\cdot)$ with a (λ, δ)-approximation that is:

$$\Pr[(1+\lambda)\sigma(\mathbf{s}) \geq \hat{\sigma}(\mathbf{s}) \geq (1-\lambda)\sigma(\mathbf{s})] \geq 1-\delta \tag{39}$$

In the experiment, we set parameters $\lambda = 0.8, \delta = 0.2$, $k = 3$ and $\epsilon = 0.1$ as in [4] to show a trade-off between solution quality and number of queries. We also budget for kIMK with several B in $\{500, 700, 1000, 1200, 1500, 2000\}$ to illustrate the fact that the expense to influence k topics via large networks such as social networks, is not a small number. We set the cost of each element according to Normalized Linear model [8]. Whereby, we set the cost from 1 to 10 with Facebook and increase the cost range from 1 to 50 with Hept and Enron.

d) Experiment results: To provide a comprehensive experiment, we ran the above algorithms several times and collected results about objective values, the number of queries, and the running time according to the B milestones. For each milestone, the average values were calculated. Figure 1 illustrates the result on the above databases. The graph lines of the algorithm Greedy in [9] were not the final results because it took too long to complete running this algorithm. To experiment fit within our system configuration, we had to limit the time to process the Greedy according to the number of nodes in each database, in which $4K$-node Facebook ran within 12 hours, $15K$-node Hept ran in 2 days and $36K$-node Enron worked for 4 days.

First, Figure 1(a)(b)(c) represent the quality of algorithms via values of objective function $\sigma(\cdot)$. The Greedy line just shows the estimation of $\sigma(\cdot)$ in time limit because it wasted extremely long to complete. In the remaining, IFSA and RS look insignificantly different, FSA marks the lowest points while DS gives the highest values in the two first Figures. In these Figures, the largest gaps between FSA, IFSA, DS, and RS are at $B = 2K$ in which DS is 2.68 times and 1.82 times on Facebook and 2.89 times and 1.13 times on Hept higher than FSA and IFSA, respectively. However, IFSA and FSA show outstanding results the others in Figure 1(c). Clearly, the graph in this Figure clusters algorithms into three groups from top to bottom: IFSA-FSA, RS-DS, and Greedy. For instance, at $B = 1K$, IFSA is 1.61 times and FSA is 1.53 times higher than RS, respectively. Similarly, IFSA is 1.85 times and FSA 1.76 times higher than DS, respectively. We can see the gap between algorithms in each group, when B grows from 300 to 1500, which seems quite small, and separate when B increases more than 1500. These lines also fluctuate according to each B milestone yet it's able to realize the general trend of IFSA, DS and RS goes up while FSA values seem unpredictable.

Second, Figure 1(d)(e)(f) display amounts of queries called to run these algorithms. FSA shows an advantage over others in terms of query complexity. It is sharply from several to a few dozens of times lower than the remaining. Specifically, DS is about 10 times and 1.5 times higher while RS is about 27 times and 3.8 times higher than FSA and IFSA, respectively on Facebook and Hept. On Enron, DS is about 12 times and 1.8 times while RS is 28 times and 4 times higher than FSA and IFSA, respectively. Moreover, FSA explicitly determines and does not belong to B milestones. Besides, the several queries of IFSA are also constant over B's milestones. On the whole, the number of queries of DS is a little higher while the number of RS is significantly higher than IFSA. In Figure1(f), the query line of RS has a small fluctuation according to B values. Besides, Greedy takes the most quantities of queries in this experiment. Although Greedy is processed in a limited time, it still expenses almost $1M$ queries on Hept and $5M$ queries on Enron while FSA just uses $60K$ on Hept and $150K$ on Enron and IFSA uses $400K$ on Hept and $1M$ on Enron. On small databases like Facebook, Greedy increases to 50 times higher than FSA and 7 times higher than IFSA. Finally, the quantities of queries of our algorithms give better results than the others.

As the query complexity make directly influences running time, the representation of the time graph in Figure 1(g)(h)(i) looks quite similar to the representation on the query graph in Figure 1(d)(e)(f) in which FSA line was drawn typically lowest. It shows the running time of FSA is from several to a few dozens of times faster than the others. IFSA runs a little faster than DS, considerably faster than RS while Greedy runs slowest.

From the above figures, we can see the trade-off between our proposed algorithms' solution quality and the query complexity. FSA tries to target the near-optimal value by dividing the ground into two subsets according to cost values of elements and reduces query complexity by the filtering condition at line 7 of the algorithm 1. Hence, the query complexity decreases significantly. Nevertheless, the performance of FSA in terms of solution quality is not high. IFSA enhances FSA by using FSA as an input and the decreasing constant threshold at

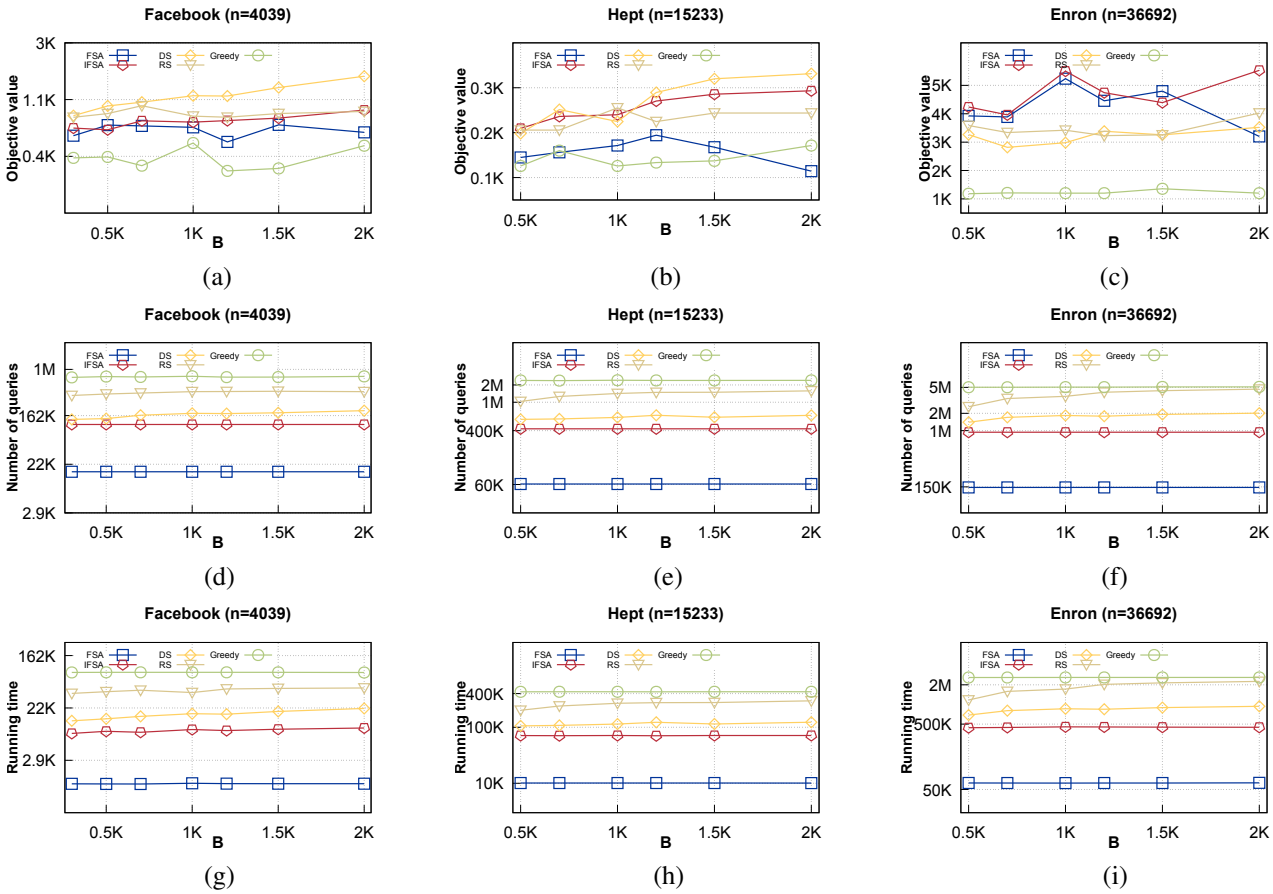


Fig. 1: Performance of algorithms for kSMK on three databases Facebook, Hept and Enron: (a-c) The objective values; (d-f) The number of queries; (g-i) The running time

line 4 of the algorithm 2. As the result, the objective of IFSA is better than FSA while the number of queries is a little higher but still deterministic. Moreover, when the ground set and B value grow up, the solution quality seems better while running time and query complexity are still constant. This is extremely important when working with big data.

Besides, the difference in quality between our algorithms and ones in [8] is due to the construction of algorithms. DS and RS focus on finding more candidate solutions than FSA and IFSA, therefore, they have more opportunities to find out a better solution. However, the disadvantage is that they waste more memory and time usage. When input data grows up, IFSA and FSA overcome others both solution quality and the number of queries.

In addition, from the above figures, the experiment shows our algorithms a few dozens of times outperform Greedy in terms of the number of queries and running time. This correctly reflects theory guarantees that our query complexity is $O(kn)$ while Greedy complexity is $O(n^4k^3)$.

B. k-type Sensor Placement under Knapsack constraint

We further study the performance of algorithms for *k-type Sensor Placement under Knapsack constraint* (kSPK) problem which is formally defined as follows:

Definition 2 (kSPK problem)**.** Given k types of sensors for different measures and a set V of n locations, each of which is assigned with only one sensor. Assuming that each sensor e has a cost $c(e)$ for every i-th type. Given the budget B, the problem aims to locate these sensors to maximize the information gained with the total cost is at most B.

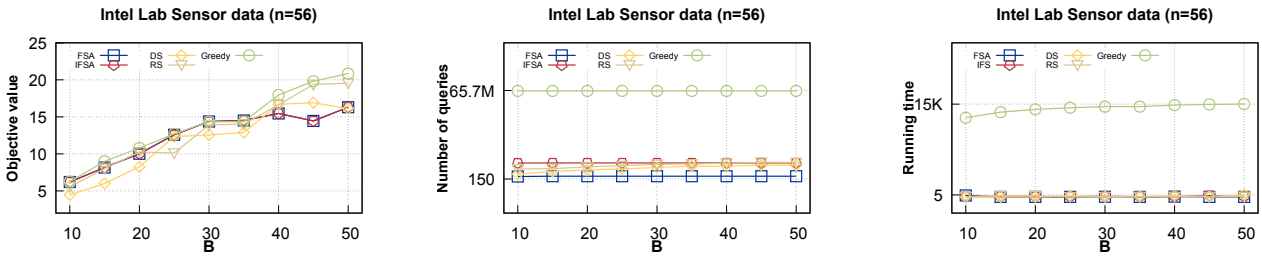
Denote by X_e^i a random variable representing the observation collected from a i-type sensor and the information gained of a k-set $\mathbf{s}$ is

$$f(\mathbf{s}) = H(\cup_{e \in supp(\mathbf{s})}\{X_e^i\}) \tag{40}$$

where H is *entropy function*. The function f is monotone and k-submodular [1].

a) Experiment settings: We use Intel Lab dataset [32] to illustrate the kSPK problem. Moreover, we set $k = 3$, $\epsilon = 0.1$, cost range from 1 to 10 values as in the experiment of kIMK but the values of B are set at several points from 10 to 50. This setting depends on the number of sensors and the similarity among algorithms.

b) Experiment results: First, regarding the objective function of kSPK, that is an oracle of Entropy function. Be-



(a) Information gained (b) Number of queries (c) Running time

Fig. 2: Performance of algorithms for kSPK

sides, the number of sensor nodes in this case is small. Hence the discrimination between objective values of experimented algorithms is not large. Nevertheless, we can see the result of Greedy are better than the others, especially at $B = 50$, as the objective line of Greedy always lies on the others on the graph. FSA and IFSA overlap while RS and DS result a little fluctuating. Overall, the general trend is increasing.

Second, the gap between Greedy line and the others in Figure 2b are significantly large. While Greedy needs millions of queries to output the final solution, the remaining algorithms just use approximately 150 queries. Similarly, running time of Greedy is also thousands of times higher than the others. Moreover, query lines and time lines of the above algorithms are almost horizontal over B's milestones. On the other hand, the number of queries and time of FSA are smallest and there is no significant difference between IFSA, DS and RS. This result illustrates the query complexity of our algorithms is more optimized than the others.

On the whole, from two genuine uses of kIMK and kSPK, our proposed algorithms are described to outperform or be a comparable effect to the state-of-the-art.

V. Conclusion

This paper studies problem of maximizing a k-submodular function under knapsack constraint. We propose two deterministic streaming algorithms which take just only $O(kn)$ query complexity. The core of our algorithms is to just keep which elements that are over a given appropriate threshold then choose among them the last elements so that the total cost does not exceed a given budget B.

In order to investigate the performance of our algorithms in practice, we conduct some experiments on two applications of Influence maximization and Sensor placement. Experimental results have shown that our algorithms not only return acceptable good solutions in terms of quality requirement but also take a sharply smaller number of queries than that of the state-of-the-art algorithms. In the future, we further investigate the problem when objective function is non-monotone.

Acknowledgment

The author Dung T.K. Ha was funded by Vingroup JSC and supported by the PhD Scholarship Programme of Vingroup Innovation Foundation (VINIF), Institute of Big Data, code VINIF.2021.TS.131. And this work was also supported by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under Grant No. 102.01-2020.21.

References

[1] N. Ohsaka and Y. Yoshida, "Monotone k-submodular function maximization with size constraints," in *In Proc. of Annual Conference on Neural Information Processing Systems (NIPS)*, 2015, pp. 694–702.

[2] A. Rafiey and Y. Yoshida, "Fast and private submodular and k-submodular functions maximization with matroid constraints," in *In Proc. of the International Conference on Machine Learning (ICML)*, 2020, pp. 7887–7897.

[3] C. Qian, J. Shi, K. Tang, and Z. Zhou, "Constrained monotone k-submodular function maximization using multiobjective evolutionary algorithms with theoretical guarantee," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 595–608, 2018.

[4] L. Nguyen and M. Thai, "Streaming k-submodular maximization under noise subject to size constraint," in *In Proc. of the International Conference on Machine Learning (ICML)*, 2020, pp. 7338–7347.

[5] A. P. Singh, A. Guillory, and J. A. Bilmes, "On bisubmodular maximization," in *In Proc. of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012, pp. 1055–1063.

[6] J. Ward and S. Zivný, "Maximizing bisubmodular and k-submodular functions," in *In Proc. of Symposium on Discrete Algorithms (SODA)*, 2014, pp. 1468–1481.

[7] S. Iwata, S. Tanigawa, and Y. Yoshida, "Improved approximation algorithms for k-submodular function maximization," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, R. Krauthgamer, Ed. SIAM, 2016, pp. 404–413.

[8] C. V. Pham, Q. C. Vu, D. K. Ha, T. T. Nguyen, and N. D. Le, "Maximizing k-submodular functions under budget constraint: applications and streaming algorithms," *J. Comb. Optim.*, vol. 44, no. 1, pp. 723–751, 2022.

[9] "On maximizing a monotone k-submodular function under a knapsack constraint," *Operations Research Letters*, vol. 50, no. 1, pp. 28–31, 2022.

[10] S. Sakaue, "On maximizing a monotone k-submodular

function subject to a matroid constraint," *Discret. Optim.*, vol. 23, pp. 105–113, 2017.

[11] A. Rafiey and Y. Yoshida, "Fast and private submodular and k-submodular functions maximization with matroid constraints," in *In Proc. of International Conference on Machine Learning (ICML)*, 2020, pp. 7887–7897.

[12] Z. Tang, C. Wang, and H. Chan, "On maximizing a monotone k-submodular function under a knapsack constraint," *CoRR*, vol. abs/2105.15159, 2021. [Online]. Available: https://arxiv.org/abs/2105.15159

[13] M. Z. M. Mitrovic, E. Kazemi and A. Karbasi, "Data summarization at scale: A two-stage submodular approach," in *In Proc. of the International Conference on Machine Learning (ICML)*, 2016, p. 3593–3602.

[14] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," in *In Proc. of IEEE International Conference on Data Mining (ICDM)*, 2010, pp. 88–97.

[15] W. Chen, L. V. S. Lakshmanan, and C. Castillo, *Information and Influence Propagation in Social Networks*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013.

[16] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Growth of the flickr social network," in *Proceedings of the first Workshop on Online Social Networks, WOSN 2008, Seattle, WA, USA, August 17-22, 2008*, C. Faloutsos, T. Karagiannis, and P. Rodriguez, Eds. ACM, 2008, pp. 25–30.

[17] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks," in *In Proc. of the International Conference on Management of Data (SIGMOD)*, 2016, pp. 695–710.

[18] H. Oshima, "Derandomization for k-submodular maximization," in *In Proc. of International Workshop Combinatorial Algorithms (IWOCA)*, L. Brankovic, J. Ryan, and W. F. Smyth, Eds., 2017, pp. 88–99.

[19] T. Soma, "No-regret algorithms for online k-submodular maximization," in *In Proc. of International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019, pp. 1205–1214.

[20] L. Zheng, H. Chan, G. Loukides, and M. Li, "Maximizing approximately k-submodular functions," in *In Proc. of the 2021 SIAM International Conference on Data Mining (SDM)*, 2021, pp. 414–422.

[21] M. Sviridenko, "A note on maximizing a submodular set function subject to a knapsack constraint," *Oper. Res. Lett.*, vol. 32, no. 1, pp. 41–43, 2004.

[22] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions - I," *Math. Program.*, vol. 14, no. 1, pp. 265–294, 1978.

[23] R. Gomes and A. Krause, "Budgeted nonparametric learning from data streams," in *In Proc. of the International Conference on Machine Learning (ICML)*, J. Fürnkranz and T. Joachims, Eds., 2010, pp. 391–398.

[24] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause, "Streaming submodular maximization: massive data summarization on the fly," in *In Proc. of International Conference on Knowledge Discovery and Data Mining (KDD)*, 2014, pp. 671–680.

[25] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani, "Fast greedy algorithms in mapreduce and streaming," in *In Proc. of Symposium on Parallelism in Algorithms and Architectures, (SPAA)*, 2013, pp. 1–10.

[26] A. Kuhnle, "Quick streaming algorithms for maximization of monotone submodular functions in linear time," in *In Proc. of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021, pp. 1360–1368.

[27] C. Huang, N. Kakimura, and Y. Yoshida, "Streaming algorithms for maximizing monotone submodular functions under a knapsack constraint," *Algorithmica*, vol. 82, no. 4, pp. 1006–1032, 2020.

[28] R. Haba, E. Kazemi, M. Feldman, and A. Karbasi, "Streaming submodular maximization under a k-set system constraint," in *In Proc. of the International Conference on Machine Learning (ICML)*, 2020, pp. 3939–3949.

[29] A. Chakrabarti and S. Kale, "Submodular maximization meets streaming: matchings, matroids, and more," *Math. Program.*, vol. 154, no. 1-2, pp. 225–247, 2015.

[30] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "A billion-scale approximation algorithm for maximizing benefit in viral marketing," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2419–2429, 2017.

[31] J. Leskovec and Krevl, "A. snap datasets: Stanford large network dataset collection," 2014. [Online]. Available: http://snap. stanford.edu/data

[32] P. Bodik, W. Hong, C. Guestrin, S. Madden, M. Paskin, and R. Thibaux, "Intel lab," 2004. [Online]. Available: http://db.csail.mit.edu/labdata/labdata.html

[33] D. Kempe, J. M. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *In Proc. of the International Conference on Knowledge Discovery and Data Mining (KDD)*, 2003, pp. 137–146.

[34] C. Borgs, M. Brautbar, J. T. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *In Proc. of Symposium on Discrete Algorithms (SODA)*, 2014, pp. 946–957.