7-2018

Efficient representative subset selection over sliding windows

Yanhao WANG
National University of Singapore

Yuchen LI
Singapore Management University, yuchenli@smu.edu.sg

Kian-Lee TAN
National University of Singapore

DOI: https://doi.org/10.1109/TKDE.2018.2854182

Citation

WANG, Yanhao; LI, Yuchen; and TAN, Kian-Lee. Efficient representative subset selection over sliding windows. (2018). *IEEE Transactions on Knowledge and Data Engineering*. 1-14. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/4092

Published in IEEE Transactions on
Knowledge and Data Engineering,
2018 July,
DOI: 10.1109/TKDE.2018.2854182

Efficient Representative Subset Selection over Sliding Windows

Yanhao Wang, Yuchen Li*, and Kian-Lee Tan

Abstract—Representative subset selection (RSS) is an important tool for users to draw insights from massive datasets. Existing literature models RSS as the submodular maximization problem to capture the "diminishing returns" property of the representativeness of selected subsets, but often only has a single constraint (e.g., cardinality), which limits its applications in many real-world problems. To capture the data recency issue and support different types of constraints, we formulate dynamic RSS in data streams as maximizing submodular functions subject to general d-knapsack constraints (SMDK) over sliding windows. We propose a KNAPWINDOW framework (KW) for SMDK. KW utilizes the KNAPSTREAM algorithm (KS) for SMDK in append-only streams as a subroutine. It maintains a sequence of checkpoints and KS instances over the sliding window. Theoretically, KW is $\frac{1-\varepsilon}{1+d}$-approximate for SMDK. Furthermore, we propose a KNAPWINDOWPLUS framework (KW$^+$) to improve upon KW. KW$^+$ builds an index SUBKNAPCHK to manage the checkpoints and KS instances. SUBKNAPCHK deletes a checkpoint whenever it can be approximated by its successors. By keeping much fewer checkpoints, KW$^+$ achieves higher efficiency than KW while still guaranteeing a $\frac{1-\varepsilon'}{2+2d}$-approximate solution for SMDK. Finally, we evaluate the efficiency and solution quality of KW and KW$^+$ in real-world datasets. The experimental results demonstrate that KW achieves more than two orders of magnitude speedups over the batch baseline and preserves high-quality solutions for SMDK over sliding windows. KW$^+$ further runs 5-10 times faster than KW while providing solutions with equivalent or even better utilities.

Index Terms—Data summarization, sliding window, data stream, submodular maximization, approximation algorithm

1 INTRODUCTION

IN THE big data era, a vast amount of data is being continuously generated by various applications, e.g., social media, network traffic, sensors, etc. An imperative task is to extract useful information from massive datasets. A compelling approach is *representative subset selection* [2]–[11] (RSS): extracting a concise subset of representative elements from the source dataset. RSS is often formulated as selecting a subset of elements to maximize a utility function that quantifies the *representativeness* subject to some *constraints*. The utility functions are often chosen to be *submodular* to capture the "diminishing returns" property of representativeness [2], [6], [8], [12]–[14], i.e., adding more elements decreases the marginal representativeness. A number of constraints are used to restrict the selected subset in various ways. For example, a common approach to scaling kernel methods in nonparametric learning is *active set selection* [2], [6] that extracts a subset S with the maximum information entropy as representatives. It restricts the size of S to k (a.k.a cardinality constraint) so that at most k elements are selected for kernel training to reduce the computational costs while still retaining model quality. As another example, *social data summarization* [8], [12] selects a subset S to best preserve the information in a collection of social posts. To restrict the summary size, two constraints are imposed: the number of selected posts in S, as well as their total length, is bounded. Additionally, the influence scores are also modeled as constraints so that more influential elements could be included in the summary [13].

In many cases, data is generated rapidly and only available as a stream [15]–[19]. To address the requirement for summarizing such datasets in real-time, RSS over data streams [2], [6], [12], [20] has been extensively studied in recent years. However, there are two major drawbacks that limit the deployment of existing approaches to many real-world applications. First, most of the streaming RSS algorithms only work with cardinality constraints, i.e., selecting a set of k elements as representatives, and cannot support more complex constraints. As aforementioned, a number of RSS problems consider more general multi-knapsack (a.k.a. d-knapsack) constraints beyond cardinality [3], [11], [13], [21]. However, the algorithms that only support cardinality constraints cannot provide solutions with any quality assurances in more general cases. Second, existing methods are developed for the append-only setting where elements are only inserted into but never deleted from the stream and thus the freshness of solutions is ignored. Data streams are highly dynamic and keep evolving over time, where recent elements are more important than earlier ones. The sliding window [22] model that only considers the W most recent elements is a natural way to capture such an essence. Although a number of RSS algorithms have been developed for append-only streams, RSS over sliding windows is still largely unexplored and, to the best of our knowledge, only one existing method [23] is proposed. It is not surprising that the method is also specific for cardinality constraints.

To address the limitations of existing methods, it requires general RSS frameworks that (i) support different types of submodular utility functions, (ii) work with more than one knapsack constraint, and (iii) extract a subset of representa-

A preliminary version [1] of this article appeared in Proc. ICDE 2018.

- *Y. Wang and K. Tan are with the School of Computing, National University of Singapore, Singapore 117417. E-mail: {yanhao90,tankl}@comp.nus.edu.sg*
- *Y. Li is with the School of Information Systems, Singapore Management University, Singapore 178902. E-mail: yuchenli@smu.edu.sg*

* *Yuchen Li is the corresponding author.*

tive elements over a sliding window efficiently.

In this paper, we formulate dynamic RSS in data streams as maximizing submodular functions with d-knapsack constraints (SMDK) over sliding windows. As SMDK is NP-hard, we focus on designing efficient approximation algorithms for SMDK. First, we devise the KNAPSTREAM algorithm (KS) for SMDK in append-only streams. KS needs a single pass over a stream and provides a $\frac{1-\varepsilon}{1+d}$-approximate solution for SMDK. It improves the state-of-the-art approximation factor of $\frac{1}{1+2d} - \varepsilon$ for SMDK in append-only streams [13]. Then, we propose two novel frameworks, namely KNAPWINDOW (KW) and KNAPWINDOWPLUS (KW$^+$), for SMDK over sliding windows. Both frameworks adapt KS for the sliding window model by maintaining a sequence of KS instances starting at different timestamps (a.k.a *checkpoints*) over the sliding window. Specifically, KW maintains $\mathcal{O}(\sqrt{W})$ checkpoints over a size-W sliding window. The interval between any neighboring checkpoints of KW is always equal. The approximation factor of KW for SMDK is the same as KS, i.e., $\frac{1-\varepsilon}{1+d}$. Furthermore, KW$^+$ is proposed to build an index SUBKNAPCHK to manage the checkpoints based on their achieved utilities. SUBKNAPCHK deletes a checkpoint whenever it can be approximated by its successors. Theoretically, the number of checkpoints in KW$^+$ is independent of W and logarithmic to the range of the utility function. Since KW$^+$ maintains much fewer checkpoints, it achieves higher efficiency than KW. Nevertheless, KW$^+$ can still guarantee $\frac{1-\varepsilon'}{2+2d}$-approximation solutions for SMDK over sliding windows.

Finally, we evaluate the efficiency and effectiveness of KW and KW$^+$ with two real-world applications: *social stream summarization* and *active set selection*. The experimental results show that KW achieves more than two orders of magnitude speedup over the batch baseline and preserves high-quality solutions for SMDK over sliding windows. KW$^+$ further runs 5–10 times faster than KW while providing solutions with equivalent or even better utilities.

Our main contributions are summarized as follows.

- We formulate dynamic RSS as maximizing submodular functions with d-knapsack constraints (SMDK) over sliding windows.
- We propose a novel $\frac{1-\varepsilon}{1+d}$-approximation KW framework for SMDK over sliding windows.
- We devise KW$^+$ to improve upon KW. Although the approximation factor of KW$^+$ drops to $\frac{1-\varepsilon'}{2+2d}$, KW$^+$ has much higher efficiency than KW while providing solutions with equivalent or better quality.
- We demonstrate the efficiency and solution quality of KW and KW$^+$ for real-world applications.

The remaining of this paper is organized as follows. Section 2 defines dynamic RSS as SMDK over sliding windows. Section 3 gives two examples of modeling real-world RSS applications as SMDK. Section 4 and Section 5 present the KW and KW$^+$ frameworks respectively. Section 6 reports the experimental results. Section 7 reviews the related work. Finally, Section 8 concludes the whole paper.

2 PROBLEM FORMULATION

In this section, we first introduce data streams and the sliding window model. Next, we give the notions of *submodular functions* and *knapsack constraints*. Then, we formally define the *representative subset selection* (RSS) problem as submodular maximization with a d-knapsack constraint (SMDK) in the sliding window model. Finally, we show the challenges of SMDK over sliding windows.

Data Stream & Sliding Window. A data stream comprises an unbounded sequence of elements $V = \langle v_1, v_2, \ldots \rangle$ and $v_t \in V$ is the t-th element of the stream. The elements in V arrive one at a time in an arbitrary order. Only *one pass* over the stream is permitted and the elements must be processed in the arrival order. Specifically, we focus on the *sliding window* model for data streams. Let W be the size of the sliding window. At any time t, the *active window* A_t is a subsequence that always contains the W most recent elements (a.k.a. *active elements*) in the stream[1], i.e., $A_t = \langle v_{t'}, \ldots, v_t \rangle$ where $t' = \max(1, t - W + 1)$.

RSS over Sliding Windows. RSS selects a set of representative elements from the ground set according to a utility function with some budget constraint. In this paper, we target the class of *nonnegative monotone submodular* utility functions adopted in a wide range of RSS problems [2], [5], [6], [8], [9], [12], [23].

Given a ground set of elements V, we consider a set function $f : 2^V \to \mathbb{R}_{\geq 0}$ that maps any subset of elements to a nonnegative *utility* value. For a set of elements $S \subseteq V$ and an element $v \in V \setminus S$, the *marginal gain* of $f(\cdot)$ is defined by $\Delta_f(v|S) \triangleq f(S \cup \{v\}) - f(S)$. Then, the *monotonicity* and *submodularity* of $f(\cdot)$ can be defined according to its marginal gain.

Definition 1 (Monotonicity & Submodularity). *A set function $f(\cdot)$ is monotone iff $\Delta_f(v|S) \geq 0$ for any $S \subseteq V$ and $v \in V \setminus S$. $f(\cdot)$ is submodular iff $\Delta_f(v|S) \geq \Delta_f(v|S')$ for any $S \subseteq S' \subseteq V$ and $v \in V \setminus S'$.*

Intuitively, monotonicity means adding more elements does not decrease the utility value. Submodularity captures the "diminishing returns" property that the marginal gain of adding any new element decreases as a set grows larger.

To handle various types of *linear* budget constraints in real-world problems, we adopt the general d-knapsack constraint [3], [11], [13], [21]. Specifically, a *knapsack* is defined by a cost function $c : V \to \mathbb{R}_+$ that assigns a positive cost to each element in the ground set V. Let $c(v)$ denote the cost of $v \in V$. The cost $c(S)$ of a set $S \subseteq V$ is the sum of the costs of its members, i.e., $c(S) = \sum_{v \in S} c(v)$. Given a budget b, we say S satisfies the knapsack constraint iff $c(S) \leq b$. W.l.o.g., we normalize the budget to $b = 1$ and the cost of any element to $c(v) \in (0, 1]$. Then, a d-knapsack constraint ξ is defined by d cost functions $c_1(\cdot), \ldots, c_d(\cdot)$. Formally, we define $\xi = \{S \subseteq V : c_j(S) \leq 1, \forall j \in [d]\}$. We say a set S satisfies the d-knapsack constraint iff $S \in \xi$.

Given the above definitions, we can formulate RSS as an optimization problem of maximizing a monotone submodular utility function $f(\cdot)$ subject to a d-knapsack constraint ξ (SMDK) over the active window A_t. At every time t, RSS returns a subset of elements S_t that (1) only contains active

1. We only discuss the sequence-based sliding window in this paper. Nevertheless, the proposed algorithms can naturally support the time-based sliding window.

TABLE 1
Frequently used notations

Notation	Description
V, v_t	V is an unbounded stream of elements; $v_t \in V$ is the t-th element in the stream.
d, ξ	d is the dimension of the knapsack constraint; ξ is the family of sets defined by the d-knapsack constraint.
$c_j(v), c_{tj}$	$c_j(v)$ is the cost of v in the j-th knapsack; c_{tj} is the cost of v_t in the j-th knapsack.
γ_t, δ_t	$\gamma_t = \min_{\forall j \in [d]} c_{tj}$ and $\delta_t = \max_{\forall j \in [d]} c_{tj}$ are the minimum and maximum costs of v_t in all d knapsacks.
γ, δ	$\gamma = \min_{\forall t, \forall j} c_{tj}$ and $\delta = \max_{\forall t, \forall j} c_{tj}$ are the lower and upper bounds for the costs of any elements in the stream.
$f(\cdot), \Delta_f(\cdot\|\cdot)$	$f(\cdot)$ is a monotone submodular utility function; $\Delta_f(\cdot\|\cdot)$ is the marginal gain defined on $f(\cdot)$.
W	W is the size of the sliding window.
A_t	$A_t = \langle v_{t'}, \dots, v_t \rangle$ is the active window at time t where $t' = \max\{1, t - W + 1\}$.
$S_t^*, \texttt{OPT}_t$	S_t^* is the optimal solution for SMDK w.r.t. the active window A_t at time t; $\texttt{OPT}_t = f(S_t^*)$ denotes the optimal utility value.
S_t	S_t denotes an approximate solution for SMDK w.r.t. A_t at time t.
X_t, x_i	$X_t = \langle x_1, \dots, x_s \rangle$ is the sequence of s checkpoints at time t maintained by KW and KW$^+$; x_i is the i-th checkpoint in X_t.
$S_{x,y}^*, S_{x,y}$	$S_{x,y}^*$ and $S_{x,y}$ are the optimal solution and an approximate solution for SMDK w.r.t. a substream $V_{x,y} = \langle v_x, \dots, v_y \rangle$.

Window	ID	Set	c_1	c_2
A_4	T_1	$\{w_1, w_3\}$	0.5	0.5
	T_2	$\{w_2, w_4\}$	0.2	0.5
	T_3	$\{w_1, w_3, w_4\}$	0.8	0.7
	T_4	$\{w_2, w_5\}$	0.4	0.1
A_5	T_5	$\{w_1, w_2, w_4\}$	0.3	0.5

For active window A_4:
$S_4^* = \{T_1, T_4\}$
$\texttt{OPT}_4 = |T_1 \cup T_4| = 4$

For active window A_5:
$S_5^* = \{T_4, T_5\}$
$\texttt{OPT}_5 = |T_4 \cup T_5| = 4$

Fig. 1. Toy example of SMDK over sliding windows. We highlight two active windows A_4, A_5 and show their optimal solutions and utilities.

elements, (2) satisfies the d-knapsack constraint ξ, and (3) maximizes the utility function $f(\cdot)$. Formally,

$$\max_{S_t \subseteq A_t} f(S_t) \quad \text{s.t.} \quad S_t \in \xi \tag{1}$$

We use $S_t^* = \text{argmax}_{S_t \subseteq A_t : S_t \in \xi} f(S_t)$ to denote the optimal solution of SMDK at time t.

Example 1. *A toy example of SMDK over sliding windows is given in Figure 1. We consider one of the simplest SMDK problems: budgeted maximum coverage (BMC) [24]. Given a domain of items $\mathcal{W} = \{w_1, \dots, w_5\}$, we have a sequence of sets $\mathcal{T} = \langle T_1, \dots, T_5 \rangle$ where each set $T \in \mathcal{T}$ is a subset of $\mathcal{W}$ associated with two costs c_1 and c_2. Let the window size be 4. The objective of BMC is to select a set of sets S_t^* from 4 most recent sets such that the number of items covered by S_t^* is maximized while S_t^* satisfies the 2-knapsack constraint defined by c_1 and c_2. In Figure 1, we highlight two active windows A_4 and A_5 at time 4 and 5 respectively. Then, we give the optimal solutions S_4^* and S_5^* and their utilities for BMC at time 4 and 5.*

Challenges of SMDK over Sliding Windows. SMDK is NP-hard. According to the definition of the d-knapsack constraint, the cardinality constraint with budget k is a special case of a 1-knapsack constraint when $c(v) = \frac{1}{k}, \forall v \in V$. Because maximizing a submodular function with a cardinality constraint is NP-hard [25], [26], SMDK is NP-hard as well. Due to the submodularity of the utility function, a naïve approach to SMDK over sliding windows is storing the active window A_t and rerunning a batch algorithm for SMDK on A_t from scratch for every window slide. Typical batch algorithms for SMDK are CostEffectGreedy [3], [27] (CEG), an extension of the classic greedy algorithm [25], and ContinuousGreedy [28], [29] (ContG) which is based on the multi-linear relaxation technique. From the theoretical perspective, the approximation ratio of CEG for SMDK depends on the dimension of knapsacks d while ContG can achieve a constant approximation (e.g., $1 - \frac{1}{e} - \varepsilon$) independent of d. But ContG suffers from extremely high time complexity (e.g., $\mathcal{O}(W^{d \cdot \varepsilon^{-4}})$ [28], see Table 4) and is not practical even for very small W. In practice, we implement CEG as the batch baseline. CEG returns near-optimal solutions for SMDK empirically when the cost distribution is not extremely adversary [13], [27]. Nevertheless, for SMDK over sliding windows, CEG still needs to scan the active elements for multiple passes and incurs heavy computational costs. Hence, our challenge is to design efficient frameworks to continuously maintain the solutions for SMDK over sliding windows when new elements arrive rapidly, while guaranteeing a constant approximation ratio w.r.t. a fixed d.

Before moving on to the subsequent sections, we summarize the frequently used notations in Table 1.

3 Applications

In this section, we give two examples of RSS applications and describe how they are modeled as SMDK over sliding windows. The experiments for both applications in real-world datasets will be reported in Section 6. Note that many more RSS problems can also be modeled as SMDK (see Section 7), which could potentially benefit from this work.

3.1 Social Stream Summarization

Massive data is continuously generated as a stream by hundreds of millions of users on social platforms, e.g., Twitter. *Social stream summarization* aims to retain a small portion of representative elements from a user-generated stream. One common approach is topic-preserving summarization [8], [12] that selects a subset of posts that best preserve *latent topics* in the stream. We focus on topic-preserving summarization in the sliding window model to capture the evolving nature of social streams, i.e., topics under discussion change over time [12]. We consider a collection of social posts V is available as a stream in ascending order of timestamp. A social post $v \in V$ is represented as a bag of l words $\{w_1, \dots, w_l\}$ drawing from the vocabulary $\mathcal{W}$. The utility $f(S)$ for a set of elements S is computed by summing up the weights of words in S where the weight of a word w is acquired based on its information entropy [8]. Specifically,

$$f(S) = \sum_{w \in \mathcal{W}} \max_{v \in S} n(v, w) \cdot p(w) \cdot \log \frac{1}{p(w)} \tag{2}$$

where $n(v, w)$ is the frequency of word w in element v, $p(w) = \frac{\sum_{v \in V} n(v,w)}{\sum_{v \in V} \sum_{w \in \mathcal{W}} n(v,w)}$ is the probability of generating a word w from the topic model. $f(S)$ has been proved to be monotone and submodular [8]. Furthermore, the representatives should satisfy the following 3-knapsack constraint. First, a uniform cost $c_1(v)$ is assigned to each element $v \in V$, i.e., $c_1(v) = \frac{1}{k}$, to bound the size of the representative set within k [8], [12]. Second, a cost $c_2(v)$ is assigned to the length l of element v since users prefer shorter summaries to longer ones [3], [21]. For normalization, we compute the average number of words $\bar{l}$ in one element and assign $c_2(v)$ as follows: given an element v of l words, $c_2(v) = \frac{1}{k} \cdot \frac{l}{\bar{l}}$. For example, when $\bar{l} = 5, k = 10$, an element v with $l = 10$ words has a cost $c_2(v) = 0.2$. Third, a cost $c_3(v)$ is assigned according to social influence [8], [13]. Let $fl(v)$ denote the number of followers of the user who posts v. We consider $c_3(v) = \min(\delta, \frac{1}{k} \cdot \frac{\log(1+fl(v))}{\log(1+\overline{fl})})$ where $\overline{fl}$ is the average number of followers of each user and δ is the upper-bound cost. We assign lower costs to the elements posted by more influential users so that the summary could include more influential elements. The upper-bound cost δ is assigned to elements posted by users with very few (e.g., 0 or 1) followers for normalization. To sum up, the social stream summarization is modeled as maximizing $f(\cdot)$ in Equation 2 with a 3-knapsack constraint defined by $c_1(\cdot)$, $c_2(\cdot)$, and $c_3(\cdot)$ over the active window A_t.

3.2 Active Set Selection

Active set selection [2], [6] is a common approach to scaling kernel methods to massive datasets. It aims to select a small subset of elements with the maximal information entropy from the source dataset. In some sites like *Yahoo!*, weblogs are continuously generated by users as a stream. Given a stream of weblogs V, each record $v \in V$ is modeled as a multi-dimensional feature vector. The representativeness of a set of vectors S is measured by the Informative Vector Machine [30] (IVM):

$$f(S) = \frac{1}{2} \log \det(\mathbf{I} + \sigma^{-2} \mathbf{K}_{S,S}) \quad (3)$$

where $\mathbf{K}_{S,S}$ is an $|S| \times |S|$ kernel matrix indexed by S and $\sigma > 0$ is a regularization parameter. For each pair of elements $v_i, v_j \in S$, the (i,j)-th entry $\mathcal{K}_{i,j}$ of $\mathbf{K}$ represents the similarity between v_i and v_j measured via a symmetric positive definite kernel function. We adopt the squared exponential kernel embedded in the Euclidean space, i.e., $\mathcal{K}_{i,j} = \exp(-\frac{\|v_i - v_j\|_2^2}{h^2})$. It has been proved that $f(\cdot)$ in Equation 3 is a monotone submodular function [2]. Furthermore, other than assigning a fixed cost to each feature vector, existing methods also use different schemes to assign costs, e.g., generating from a Gamma distribution or marginal-dependent costs [31]. Thus, we consider a more general case: each feature vector v is associated with a cost $c(v)$ drawing from an arbitrary distribution $\mathcal{D}$ within range $(0, 1)$. The objective is to select a subset S of feature vectors such that $f(S)$ in Equation 3 is maximized subject to a 1-knapsack constraint defined by $c(\cdot)$ over the active window A_t.

4 The KnapWindow Framework

In this section, we propose the KnapWindow (KW) framework for SMDK over sliding windows. The architecture of KW is illustrated in Figure 2. KW always stores all active elements in A_t at any time t. Then, KW adapts the KnapStream (KS) algorithm that provides an approximation solution for SMDK in append-only streams to work in the sliding model in the following manner. It maintains a sequence of *checkpoints* $X_t = \langle x_1, \ldots, x_s \rangle \subseteq [t', t]$ over the active window A_t. The interval between any neighboring checkpoints x_i and x_{i+1} is equal (e.g., the interval is 3 in Figure 2). For each checkpoint x_i, a KS instance $\mathcal{H}(x_i)$ is maintained by processing an append-only stream from v_{x_i} to v_t. To retrieve the solution for SMDK at time t, KW always uses the result from $\mathcal{H}(x_1)$ corresponding to x_1. $\mathcal{H}(x_1)$ first post-processes the active elements before v_{x_1} (e.g., the solid red ones in Figure 2) and uses the result after post-processing as the final solution.

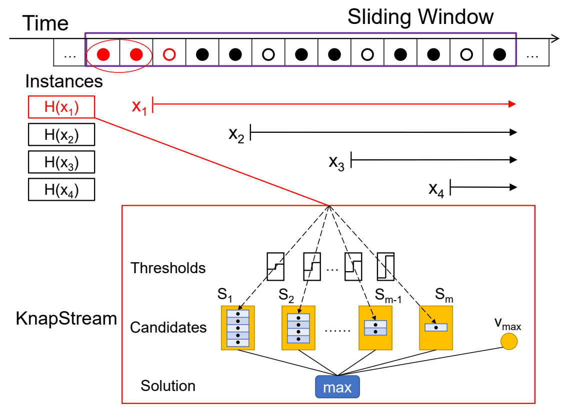


Fig. 2. An illustration of the KnapWindow framework.

The scheme of KS to maintain a solution for SMDK over an append-only stream is also illustrated in Figure 2. First, KS approximates the optimal utility `OPT` for SMDK by a sequence of estimations. Then, KS maintains a *candidate* for each estimation with a unique *threshold* derived from the estimation. Whenever receiving a new element, KS checks whether it can be included into each candidate independently according to the threshold. Finally, KS selects the candidate with the maximum utility among all candidates as the solution for its processed substream.

Next, Section 4.1 will present the KnapStream algorithm for SMDK in append-only streams. Then, Section 4.2 will introduce how the KnapWindow algorithm adapts KnapStream for the sliding window model. Finally, Section 4.3 will analyze both algorithms theoretically.

4.1 The KnapStream Algorithm

In this subsection, we propose the KnapStream (KS) algorithm to maintain a solution for SMDK w.r.t. an append-only stream $V_{x,y} = \langle v_x, \ldots, v_y \rangle$ from time x to y. KS follows the threshold-based framework [6], [32] for streaming submodular maximization. Its mechanism depends on estimating the optimal utility value `OPT` for SMDK w.r.t. $V_{x,y}$. Although `OPT` cannot be exactly determined unless P=NP, KS tracks the lower and upper bounds for `OPT` from the observed elements online and maintains a sequence of candidates with different estimations for `OPT` in the range. Each candidate derives a unique threshold for the marginal gain according to its estimation for `OPT`. When a new element

Algorithm 1 KNAPSTREAM

Input: A stream $V_{x,y} = \langle v_x, \ldots, v_y \rangle$, a parameter λ
Output: The solution $S_{x,y}$ for SMDK w.r.t. $V_{x,y}$
1: $\Phi = \langle (1+\lambda)^l | l \in \mathbb{Z} \rangle$
2: **for all** $\phi \in \Phi$ **do** $S_\phi \leftarrow \emptyset$
3: Initialize $m, M \leftarrow 0$ and $v_{max} \leftarrow nil$
4: **for** $t \leftarrow x, \ldots, y$ **do**
5:     **if** $f(\{v_t\}) > f(\{v_{max}\})$ **then** $v_{max} \leftarrow v_t$
6:     $\delta_t = \max_{\forall j \in [d]} c_{tj}, \gamma_t = \min_{\forall j \in [d]} c_{tj}$
7:     **if** $\frac{f(\{v_t\})}{\gamma_t} > M$ **then**
8:         $M \leftarrow \frac{f(\{v_t\})}{\gamma_t}, m \leftarrow f(\{v_t\})$
9:     $\Phi_t = \langle (1+\lambda)^l | l \in \mathbb{Z}, m \leq (1+\lambda)^l \leq M \cdot (1+d) \rangle$
10:     Delete S_ϕ if $\phi \notin \Phi_t$
11:     **for all** $\phi \in \Phi_t$ **do**
12:         **if** $\Delta_f(v_t|S_\phi) \geq \frac{\delta_t \cdot \phi}{1+d} \wedge S_\phi \cup \{v_t\} \in \xi$ **then**
13:             $S_\phi \leftarrow S_\phi \cup \{v_t\}$
14: $S_{max} \leftarrow \text{argmax}_{\phi \in \Phi} f(S_\phi)$
15: **return** $S_{x,y} \leftarrow \text{argmax}(f(S_{max}), f(\{v_{max}\}))$

Algorithm 2 KNAPWINDOW

Input: A stream $V = \langle v_1, v_2, \ldots \rangle$, the window size W, the interval L for neighboring checkpoints
Output: The solution S_t for SMDK at time t
1: Initialize $s \leftarrow 0$, $X_0 \leftarrow \emptyset$
2: **for** $t \leftarrow 1, 2, \ldots$ **do**
3:     **if** $t \in \{x | x = j \cdot L, j \in \mathbb{N}\}$ **then**
4:         $s \leftarrow s+1$, $x_s \leftarrow t$, and $X_t \leftarrow X_{t-L} \circ \langle x_s \rangle$
5:         Initiate a KS instance $\mathcal{H}(x_s)$
6:     **while** $t > W \wedge x_1 < t'$ **do**
7:         $X_t \leftarrow X_t \setminus \langle x_1 \rangle$, terminate $\mathcal{H}(x_1)$
8:         Shift the remaining checkpoints, $s \leftarrow s-1$
9:     **for** $i \leftarrow 1, \ldots, s$ **do**
10:         $\mathcal{H}(x_i)$ processes v_t according to Algorithm 1
11:     *// The post-processing procedure at time t*
12:     $\mathcal{H}(x_1)$ processes each element from $v_{t'}$ to v_{x_1-1} according to Algorithm 1
13:     **return** $S_t \leftarrow$ the solution of $\mathcal{H}(x_1)$

arrives, a candidate decides whether to include it based on the marginal gain of adding it into the candidate and the candidate's threshold. After processing the stream, the candidate with the maximum utility is used as the solution.

Although having a similar scheme, the algorithms in [6] and [32] only work with one cardinality constraint, whereas KS is different from them in two aspects to achieve an approximation guarantee for general d-knapsack constraints: (1) the criterion for the inclusion of an element considers not only its marginal gain but also its costs, i.e., it checks the cost-effectiveness of adding the element in each knapsack and includes it only when its cost-effectiveness reaches the threshold in d knapsacks; (2) the singleton element with the maximum self-utility is also a candidate solution.

The pseudo-code of KS is presented in Algorithm 1. Three auxiliary variables are maintained by KS (Lines 5–8): v_{max} stores the element with the maximum self-utility; M and m track the upper and lower bounds for `OPT`. Specifically, M is the maximum cost-effectiveness any observed element can achieve and m is the corresponding self-utility. We will explain why they are the upper and lower bounds for `OPT` in the proof of Theorem 1. The sequence of estimations $\Phi = \langle (1+\lambda)^l | l \in \mathbb{Z}, m \leq (1+\lambda)^l \leq M \cdot (1+d) \rangle$ and corresponding candidates are updated based on the up-to-date m and M (Lines 9–10). Then, given an element v_t, each candidate checks whether to include it independently. For each $\phi \in \Phi_t$, if the marginal gain $\Delta_f(v_t|S_\phi)$ of adding v_t to S_ϕ reaches $\frac{\delta_t \cdot \phi}{1+d}$ where $\delta_t = \max_{\forall j \in [d]} c_{tj}$ and the d-knapsack constraint is still satisfied after adding v_t, v_t will be included into S_ϕ (Lines 11–13). Finally, after processing every element in the stream, it first finds S_{max} with the maximum utility among the candidates and then compares the utility of S_{max} with that of $\{v_{max}\}$. The one with the higher utility is returned as the solution $S_{x,y}$ for SMDK w.r.t. the stream $V_{x,y}$ (Lines 14 and 15).

4.2 The KnapWindow Algorithm

In this subsection, we present the KNAPWINDOW (KW) algorithm. It adapts KS for SMDK in the sliding window model by maintaining a sequence of *checkpoints* and corresponding KS instances over the sliding window. At any time t, KW maintains a sequence of s checkpoints $X_t = \langle x_1, \ldots, x_s \rangle \subseteq [t', t]$. The interval between any neighboring checkpoints in X_t is always equal. Given the interval $L \in \mathbb{Z}^+$, KW only creates a new checkpoint and initiates a new KS instance for every L elements. For each checkpoint x_i, a KS instance $\mathcal{H}(x_i)$ is maintained by processing a substream from element v_{x_i} to the up-to-date element v_t. Whenever the first checkpoint x_1 expires from the sliding window ($x_1 < t'$ where $t' = max(1, t-W+1)$), it will be deleted from X_t. The corresponding KS instance $\mathcal{H}(x_1)$ will be terminated as well. To provide the solution for SMDK w.r.t. A_t, it uses the result from $\mathcal{H}(x_1)$. But it is noted that the elements from $v_{t'}$ to v_{x_1-1} have not been processed by $\mathcal{H}(x_1)$ yet. Therefore, it feeds the unprocessed elements to $\mathcal{H}(x_1)$ before returning the final solution.

The pseudo-code of KW is presented in Algorithm 2. The sequence of checkpoints is initialized to $X_0 = \varnothing$. A checkpoint $x_s = t$ is created and appended to the end of X_t at time $t = L, 2L, \ldots$. A KS instance $\mathcal{H}(x_s)$ is initiated accordingly (Lines 3–5). Then, it deletes the expired checkpoints from X_t (Lines 6–8). Subsequently, each checkpoint processes v_t and updates the result independently. This procedure follows Lines 6–13 of Algorithm 1. To provide the solution S_t for SMDK at time t, $\mathcal{H}(x_1)$ post-processes the elements from $v_{t'}$ to v_{x_1-1} (Line 12). Finally, the solution of $\mathcal{H}(x_1)$ after post-processing is returned as S_t (Line 13).

4.3 Theoretical Analysis

In this subsection, we analyze the approximation ratios and complexities of KS and KW. In the theoretical analysis, we assume the cost of any element is bounded by γ and δ, i.e., $0 < \gamma \leq c_{tj} \leq \delta \leq 1$ for all t, j. It is noted that the algorithms do not need to know γ and δ in advance.

The roadmap of our analysis is as follows. First of all, we present the approximation ratio of KS. We first show that if we knew the optimal utility `OPT` for SMDK w.r.t. $V_{x,y}$ in advance, the candidate whose estimation is the closest to `OPT` would be a $\frac{(1-\lambda)(1-\delta)}{1+d}$ approximate solution (Lemma 1). However, the approximation ratio depends on δ and may degrade arbitrarily when δ increases. Therefore, we further show that if the singleton element with the maximum self-utility is also considered as a candidate solution (Line 14 of

Algorithm 1), there is a lower bound for the approximation ratio regardless of δ (Lemma 2). Then, as OPT is unknown unless P=NP, we analyze how KS can track the lower and upper bounds for OPT and how many different estimations are required to guarantee that at least one of them approximates OPT within a bounded error ratio (Theorem 1). As KS maintains one candidate for each OPT estimation, we can get its time and space complexity accordingly. After providing the theoretical results for KS, we extend these results to KW. Specifically, KW retains the approximation ratio of KS because it is guaranteed that the solution of KW is returned only after processing all active elements (Theorem 2). Finally, we analyze the complexity of KW.

Lemma 1. *Assuming there exists $\phi \in \Phi$ such that $(1-\lambda)\texttt{OPT} \leq \phi \leq \texttt{OPT}$ where OPT is the optimal utility of SMDK w.r.t. $V_{x,y}$, S_ϕ satisfies that $f(S_\phi) \geq \frac{(1-\lambda)(1-\delta)}{1+d} \cdot \texttt{OPT}$.*

Proof. Let s_i be the i-th element added to S_ϕ, S_ϕ^i be $\{s_1, \ldots, s_i\}$ for $i \in [0, |S_\phi|]$ with $S_\phi^0 = \emptyset$, $b_j = c_j(S_\phi)$ for $j \in [d]$ be the cost of S_ϕ in the j-th knapsack, and $b = \max_{j \in [d]} b_j$ be the maximal cost of S_ϕ among d knapsacks. According to Line 12 in Algorithm 1, we have $\Delta_f(s_i|S_\phi^{i-1}) \geq \frac{c_j(s_i)\cdot\phi}{1+d}$ for $j \in [d]$. It holds that:

$$f(S_\phi) = \sum_{i=1}^{|S_\phi|} \Delta_f(s_i|S_\phi^{i-1}) \geq \frac{\phi}{1+d} \cdot c_j(S_\phi) = \frac{\phi}{1+d} \cdot b_j$$

Therefore, $f(S_\phi) \geq \frac{\phi}{1+d} \cdot b$.

Next, we discuss two cases separately as follows.

Case 1. When $b \geq (1-\delta)$, we have:

$$f(S_\phi) \geq \frac{b\cdot\phi}{1+d} \geq \frac{(1-\delta)\cdot\phi}{1+d} \geq \frac{(1-\lambda)(1-\delta)}{1+d} \cdot \texttt{OPT}$$

Case 2. When $b < (1-\delta)$, we have $\forall v \in V \setminus S_\phi$, $S_\phi \cup \{v\} \in \xi$. Let S^* be the optimal solution for V and a be an element in $S^* \setminus S_\phi$. Since a is not added to S_ϕ, there must exist $\mu(a) \in [d]$ such that $\Delta_f(a|S'_\phi) < \frac{c_{\mu(a)}(a)\cdot\phi}{1+d}$, where $S'_\phi \subseteq S_\phi$ is the subset of S_ϕ when a is processed. We consider $S_j^* = \{a | a \in S^* \setminus S_\phi \wedge \mu(a) = j\}$ for $j \in [d]$. Due to the submodularity of $f(\cdot)$, we acquire:

$$f(S_\phi \cup S_j^*) - f(S_\phi) \leq \sum_{a \in S_j^*} \Delta_f(a|S_\phi) < \frac{\phi \cdot c_j(S_j^*)}{1+d} \leq \frac{\phi}{1+d}$$

Then, because $S^* \setminus S_\phi = \cup_{j=1}^d S_j^*$, we have:

$$f(S^* \cup S_\phi) - f(S_\phi) \leq \sum_{j=1}^d f(S_\phi \cup S_j^*) - f(S_\phi) < \frac{d\phi}{1+d}$$

Finally, we get $f(S_\phi) > \texttt{OPT} - \frac{d}{1+d}\texttt{OPT} \geq \frac{1}{1+d} \cdot \texttt{OPT}$.

Considering both cases, we conclude the proof. □

Lemma 1 has proved that KS achieves a good approximation ratio when δ is small. Next, we further analyze the case where $\delta > 0.5$ and prove that the approximation ratio has a lower bound regardless of δ.

Lemma 2. *When $\delta > 0.5$, it satisfies that at least one of $f(S_\phi)$ and $f(\{v_{max}\})$ is greater than $\frac{0.5(1-\lambda)}{1+d} \cdot \texttt{OPT}$.*

Proof. Lemma 2 naturally follows when $b \geq 0.5$ (Case 1 of Lemma 1) or for all $a \in S^* \setminus S_\phi$, a is excluded from S_ϕ because its marginal gain does not reach the threshold in some knapsack (Case 2 of Lemma 1).

Thus, we only need to consider the following case: there exists some elements whose marginal gains reach the threshold in all knapsacks but are excluded from S_ϕ because including them into S_ϕ violates the d-knapsack constraint. Assuming a is such an element for S_ϕ, we have $\Delta_f(a|S'_\phi) \geq \frac{c_j(a)\cdot\phi}{1+d}$ and $c_j(S'_\phi) + c_j(a) > 1$ for some $j \in [d]$. In this case, we have:

$$f(S'_\phi \cup \{a\}) \geq \frac{\phi}{1+d} \cdot \Big(c_j(S'_\phi) + c_j(a)\Big) > \frac{\phi}{1+d}$$

Due to the monotonicity and submodularity of $f(\cdot)$, we get:

$$\frac{\phi}{1+d} \leq f(S'_\phi \cup \{a\}) \leq f(S'_\phi) + f(\{a\}) \leq f(S_\phi) + f(\{v_{max}\})$$

Therefore, at least one of $f(S_\phi)$ and $f(\{v_{max}\})$ is greater than $\frac{0.5\phi}{1+d} \cdot \texttt{OPT}$ and we conclude the proof. □

Given Lemmas 1 and 2, we prove that KS achieves an approximation factor of $\frac{(1-\delta)(1-\lambda)}{1+d}$ (when $\delta \leq 0.5$) or $\frac{0.5(1-\lambda)}{1+d}$ (when $\delta > 0.5$).

Theorem 1. *The solution $S_{x,y}$ returned by Algorithm 1 satisfies $f(S_{x,y}) \geq \frac{1-\varepsilon}{1+d} \cdot f(S_{x,y}^*)$ where $S_{x,y}^*$ is the optimal solution for SMDK w.r.t. $V_{x,y}$ and $\varepsilon = \min(\delta+\lambda, 0.5+\lambda)$.*

Proof. By Lemmas 1 and 2, we can say Theorem 1 naturally holds if there exists at least one $\phi \in \Phi$ such that $(1-\lambda)\texttt{OPT} \leq \phi \leq \texttt{OPT}$. First, we show m and M are the lower and upper bounds for OPT. It is easy to see $m \leq \texttt{OPT}$ as $m \leq f(\{v_{max}\})$ and $\{v\} \in \xi$ for any $v \in V$. M maintains the maximum cost-effectiveness among all elements. We have $M \geq \frac{f(\{v_i\})}{c_{ij}}$, $\forall i \in [x, y]$ and $\forall j \in [d]$. Let $S_{x,y}^* = \{a_1, \ldots, a_{|S^*|}\}$ be the optimal solution for $V_{x,y}$. As $f(\cdot)$ is monotone submodular, $\texttt{OPT} \leq \sum_{i=1}^{|S_{x,y}^*|} f(\{a_i\}) \leq c_j(S^*)M$ for $j \in [d]$. As $c_j(S_{x,y}^*) \leq 1$, we have $M \geq \texttt{OPT}$. KS estimates OPT by a sequence $\langle(1+\lambda)^l | l \in \mathbb{Z}, m \leq (1+\lambda)^l \leq M(1+d)\rangle$. Then, there exists at least one estimation ϕ such that $\phi \leq \texttt{OPT} \leq (1+\lambda)\phi$. Equivalently, $(1-\lambda)\texttt{OPT} \leq \phi \leq \texttt{OPT}$. Therefore, we conclude the proof by combining this result with Lemma 1 and 2. □

The Complexity of KS. As only one pass over the stream is permitted, to avoid missing elements with marginal gains of greater than $\frac{M}{1+d}$, KS maintains the candidates for estimations within an increased range $[m, (1+d)M]$ instead of $[m, M]$. Then, because $\frac{m}{M} \leq \gamma$ (Line 8 of Algorithm 1), the number of candidates in KS is bounded by $\lceil \log_{1+\lambda} \gamma^{-1}(1+d) \rceil$. Thus, we have KS maintains $\mathcal{O}(\frac{\log(d\cdot\gamma^{-1})}{\varepsilon})$ candidates. For each candidate, one function call is required to evaluate whether to add a new element. Thus, the time complexity to update one element is $\mathcal{O}(\frac{\log(d\cdot\gamma^{-1})}{\varepsilon})$. Finally, at most γ^{-1} elements can be maintained in each candidate. Otherwise, the d-knapsack constraint must not be satisfied. Therefore, the number of elements stored is $\mathcal{O}(\frac{\log(d\cdot\gamma^{-1})}{\gamma\cdot\varepsilon})$.

Next, we present the approximation factor of KW.

Theorem 2. *The solution S_t returned by Algorithm 2 satisfies $f(S_t) \geq \frac{1-\varepsilon}{1+d} \cdot \texttt{OPT}_t$ where $\texttt{OPT}_t$ is the optimal utility for SMDK w.r.t. A_t at time t and $\varepsilon = \min(\delta+\lambda, 0.5+\lambda)$.*

It is obvious that $\mathcal{H}(x_1)$ must have processed every element in A_t after post-processing. As the approximation ratio of KS is order-independent, i.e., no assumption is made for the arrival order of elements, Theorem 2 holds.

The Complexity of KW. KW maintains $s = \lceil \frac{W}{L} \rceil$ checkpoints for A_t and thus updates the KS instances for an element in $\mathcal{O}(\frac{s\cdot\log(d\cdot\gamma^{-1})}{\lambda})$ time. In addition, it takes

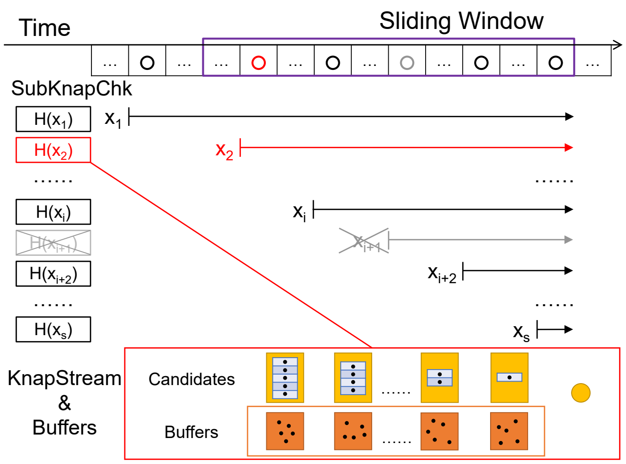


Fig. 3. An illustration of the KNAPWINDOWPLUS framework

$\mathcal{O}(\frac{L \cdot \log(d \cdot \gamma^{-1})}{\lambda})$ time for post-processing. The time complexity of KW is $\mathcal{O}(\frac{(s+L) \cdot \log(d \cdot \gamma^{-1})}{\lambda})$. When $s = L = \sqrt{W}$, it becomes $\mathcal{O}(\frac{\sqrt{W} \cdot \log(d \cdot \gamma^{-1})}{\lambda})$. Finally, because all active elements must be stored, the space complexity of KW is $\mathcal{O}(W)$.

5 THE KNAPWINDOWPLUS FRAMEWORK

Although KW can provide approximation solutions for SMDK with a theoretical bound, it still suffers from two drawbacks that limit its application for a large window size W. First, KW cannot handle the case when the window does not fit in the main memory. Second, as $\mathcal{O}(\sqrt{W})$ KS instances are maintained for a size-W sliding window, the efficiency of KW degrades with increasing W. To improve upon KW, we further propose the KNAPWINDOWPLUS framework (KW^+) in this section.

The architecture of KW^+ is illustrated in Figure 3. The basic idea of KW^+ is similar to KW: it also keeps the sequence of checkpoints $X_t = \langle x_1, \ldots, x_s \rangle$ and maintains a KS instance to process a substream from v_{x_i} to v_t at time t in each checkpoint $x_i \in X_t$. However, KW^+ is substantially different from KW in the following four aspects. First, KW^+ does not store the entire active window but only keeps the elements within each KS instance. The number of elements kept by KW^+ is empirically much smaller than W. Second, KW^+ builds an index SUBKNAPCHK for checkpoint maintenance. Instead of maintaining a sequence of checkpoints with equal interval, KW^+ creates a checkpoint and the corresponding KS instance for every arrival element. Then, SUBKNAPCHK manages the checkpoints based on their utilities and deletes a checkpoint whenever it can be approximated by its successors. By using SUBKNAPCHK, the number of checkpoints in KW^+ is independent of W. Third, KW^+ will keep one expired checkpoint (i.e., x_1) when $t > W$. It tracks the optimal utility $\texttt{OPT}_t$ for SMDK w.r.t. A_t to guarantee the theoretical soundness of the solutions. Fourth, KW^+ maintains a buffer with tunable size along with each candidate of the KS instances. In the post-processing procedure, the elements in buffers are added into the candidates to improve the utilities of solutions.

Next, Section 5.1 will introduce the KW^+ algorithm. Then, Section 5.2 will provide a theoretical analysis for KW^+. Finally, Section 5.3 will discuss how to adapt KW and KW^+ to the scenario where the sliding window shifts for more than one element at a time.

5.1 The KnapWindowPlus Algorithm

In this subsection, we describe the KNAPWINDOWPLUS algorithm (KW^+) in detail. We first present a novel index *Submodular Knapsack Checkpoints* (SUBKNAPCHK) to maintain a sequence of *checkpoints* and corresponding KS instances over the sliding window. Then, we show the procedures for buffer maintenance and post-processing.

Submodular Knapsack Checkpoints. At time t, an index *Submodular Knapsack Checkpoints* (SUBKNAPCHK) comprises a sequence of s checkpoints $X_t = \langle x_1, \ldots, x_s \rangle$ where $x_1 < \ldots < x_s = t$. For each checkpoint x_i, a KS instance $\mathcal{H}(x_i)$ is maintained. $\mathcal{H}(x_i)$ processes a substream from v_{x_i} to v_t and will be terminated when x_i is deleted from SUBKNAPCHK. When $t > W$, the first checkpoint x_1 expires (i.e., $x_1 < t'$) but is not deleted from SUBKNAPCHK immediately. It is maintained to track the upper bound for the optimal utility $\texttt{OPT}_t$ of SMDK w.r.t. A_t. However, the result of $\mathcal{H}(x_1)$ cannot be used as the solution at time t in this case because it may contain expired elements. SUBKNAPCHK restricts the number of expired checkpoints to at most 1. Therefore, x_2 must not expire and the result of $\mathcal{H}(x_2)$ is returned as the solution for SMDK w.r.t. A_t when $t > W$.

The idea of maintaining a sequence of *checkpoints* over sliding windows is inspired by *smooth histograms* [33]. However, according to the analysis in [23], the method in [33] cannot be directly applied to SMDK because it requires an append-only streaming algorithm with at least 0.8-approximation for each checkpoint. Unfortunately, [25], [26] show that there is no polynomial algorithm for SMDK that can achieve an approximation ratio of better than $1 - \frac{1}{e} \approx 0.63$ unless P=NP. Therefore, we devise a novel strategy to maintain an adequate sequence of checkpoints so that (1) the number of checkpoints is as few as possible for high efficiency; (2) the utilities of the solutions still achieve a bounded approximation ratio to the optimal one.

Towards both objectives, we propose the following strategy to maintain the checkpoints in SUBKNAPCHK: (1) create a checkpoint and a KS instance for each arrival element; (2) delete a checkpoint and terminate its KS instance once it can be approximated by any successive checkpoint. Let $f[x_i, t]$ denote the utility of the solution returned by $\mathcal{H}(x_i)$ at time t. Given three neighboring checkpoints x_i, x_{i+1}, x_{i+2} ($i \in [1, s-2]$) and a parameter $\beta > 0$, if $f[x_{i+2}, t] \geq (1-\beta) f[x_i, t]$, we consider the second checkpoint x_{i+1} can be approximated by the third one x_{i+2}. In this case, x_{i+1} will be deleted from SUBKNAPCHK. We will formally analyze the soundness of such a strategy in Section 5.2.

Buffer Maintenance and Post-Processing. To further improve the empirical performance of KW^+, we maintain buffers along with the candidates in KS instances and use these buffers for post-processing before returning the final solution. The reasons why the buffers and post-processing are essential are as follows. First, by using SUBKNAPCHK, the solutions of $\mathcal{H}(x_2)$ are always used for A_t when $t > W$. As it is common that $x_2 \gg t'$, all elements between $v_{t'}$ and $v_{x_2 - 1}$ are missing from the solutions of $\mathcal{H}(x_2)$. Second, the candidates with high thresholds in KS instances are hard to be filled, even if more elements could still be added without violating the d-knapsack constraint. Therefore, we maintain the buffers for post-processing to improve the solution quality of KW$^+$.

We consider a buffer $B_\phi = \emptyset$ is initialized when each candidate S_ϕ in a KS instance $\mathcal{H}(x_i)$ ($i \in [1, s]$) is created. When processing an element v_t, if adding v_t to S_ϕ achieves a marginal gain of slightly lower than the threshold, i.e., $\Delta_f(v_t|S_\phi) \geq \alpha \cdot \frac{\delta_t \cdot \phi}{1+d}$, v_t will be added to B_ϕ. Here, $\alpha \in (0, 1)$ is used to control the lower bound for an element to be added to B_ϕ. Furthermore, we restrict the buffer size to η. When the number of elements in B_ϕ exceeds η, we first drop each element v if $S_\phi \cup \{v\} \notin \xi$. Then, we drop the elements with the least cost-effectivenesses w.r.t. S_ϕ until $|B_\phi| = \eta$, where the cost-effectiveness of element v is computed by $\frac{\Delta_f(v|S_\phi)}{\delta(v)}$, $\delta(v) = \max_{j \in [d]} c_j(v)$. Before returning the solution at time t, we perform the post-processing procedure using buffers of $\mathcal{H}(x_1)$ and $\mathcal{H}(x_2)$ (if $t < W$, only the buffers of $\mathcal{H}(x_1)$ is used). Specifically, for each candidate S_ϕ, we run CostEffectGreedy [3] to add elements in buffers to S_ϕ. After post-processing each candidate, we also return the candidate with the maximum utility as the final solution.

Algorithmic Description. The pseudo-code of KW$^+$ is presented in Algorithm 3. The maintenance of SubKnapChk is shown in Lines 3–15. At time t, a new checkpoint $x_s = t$ and a KS instance $\mathcal{H}(x_s)$ are created for v_t. Then, if there is more than one expired checkpoint in SubKnapChk, all except the last one will be deleted (Lines 6–8). This guarantees that there is only one expired checkpoint in X_t. Subsequently, for each checkpoint x_i, $\mathcal{H}(x_i)$ processes v_t and updates the candidates independently according to Lines 6–13 of Algorithm 1. After updating the candidates of $\mathcal{H}(x_i)$ for v_t, it performs the buffer maintenance procedure as follows (Lines 25–31). If $\Delta_f(v_t|S_\phi) \geq \alpha \cdot \frac{\delta_t \cdot \phi}{1+d}$, v_t is added to B_ϕ. When the number of elements in B_ϕ exceeds η, it first drops any $v \in B_\phi$ if $S_\phi \cup \{v\} \notin \xi$ and then drops the element v' with the least cost-effectiveness in B_ϕ until $|B_\phi| = \eta$. Next, it maintains the checkpoints in SubKnapChk. The checkpoints that can be approximated by its successor are identified and deleted from SubKnapChk (Lines 13–15). After the SubKnapChk maintenance, for any $x \in X_t$, there is at most one checkpoint $x' \in X_t$ such that $x' > x$ and $f[x', t] \geq (1 - \beta) f[x, t]$. Finally, the post-processing procedure is executed before returning the solution S_t for SMDK w.r.t. A_t. When $t < W$, $\mathcal{H}(x_1)$ will provide S_t. Each candidate S_ϕ in $\mathcal{H}(x_1)$ considers B_ϕ for post-processing. Otherwise, $\mathcal{H}(x_2)$ will provide S_t. We first add the non-expired elements in S'_ϕ and B'_ϕ of $\mathcal{H}(x_1)$ to B_ϕ for post-processing. Starting from S_ϕ, the post-processing procedure greedily adds the element v^* with the maximum cost-effectiveness in B_ϕ to S_ϕ until none of the remaining elements in B_ϕ can be included without violating the d-knapsack constraint (Lines 32–35). After the post-processing, it also returns the candidate with the maximum utility among the candidates in $\mathcal{H}(x_1)$ or $\mathcal{H}(x_2)$ as the final solution S_t.

5.2 Theoretical Analysis

Next, we analyze the approximation ratio and complexity of KW$^+$. We first prove the properties of the checkpoints

Algorithm 3 KnapWindowPlus

Input: A stream $V = \langle v_1, v_2, \ldots \rangle$, the window size W, the buffer size η, the parameters α and β
Output: The solution S_t for SMDK at time t

1: Initialize $s \leftarrow 0$, $X_0 \leftarrow \emptyset$
2: **for** $t \leftarrow 1, 2, \ldots$ **do**
3: $s \leftarrow s + 1$, $x_s \leftarrow t$, and $X_t \leftarrow X_{t-1} \circ \langle x_s \rangle$
4: Initiate a KS instance $\mathcal{H}(x_s)$
5: **for all** S_ϕ of $\mathcal{H}(x_s)$ **do** Initialize a buffer $B_\phi \leftarrow \emptyset$
6: **while** $t > W \wedge x_2 < t'$ **do**
7: $X_t \leftarrow X_t \setminus \langle x_1 \rangle$, terminate $\mathcal{H}(x_1)$
8: Shift the remaining checkpoints, $s \leftarrow s - 1$
9: **for** $i \leftarrow 1, \ldots, s$ **do**
10: $\mathcal{H}(x_i)$ processes v_t according to Algorithm 1
11: *// buffer maintenance*
12: **for all** S_ϕ of $\mathcal{H}(x_i)$ **do** Buffer(S_ϕ, B_ϕ, v_t)
13: **while** $\exists i \in [1, s-2] : f[x_{i+2}, t] \geq (1 - \beta) f[x_i, t]$ **do**
14: $X_t \leftarrow X_t \setminus \langle x_{i+1} \rangle$, terminate $\mathcal{H}(x_{i+1})$
15: Shift the remaining checkpoints, $s \leftarrow s - 1$
16: *// post-processing*
17: **if** $x_1 \geq t'$ **then**
18: **for all** S_ϕ of $\mathcal{H}(x_1)$ **do** CostEffectGreedy(S_ϕ, B_ϕ)
19: **return** $S_t \leftarrow$ the result of $\mathcal{H}(x_1)$
20: **else**
21: **for all** S_ϕ of $\mathcal{H}(x_2)$ **do**
22: Add each element v in S'_ϕ and B'_ϕ of $\mathcal{H}(x_1)$ to B_ϕ if v does not expire and $S_\phi \cup \{v\} \in \xi$
23: CostEffectGreedy(S_ϕ, B_ϕ)
24: **return** $S_t \leftarrow$ the result of $\mathcal{H}(x_2)$
25: **procedure** Buffer(S_ϕ, B_ϕ, v_t)
26: **if** $v_t \notin S_\phi \wedge \Delta_f(v_t|S_\phi) \geq \alpha \cdot \frac{\delta_t \cdot \phi}{1+d}$ **then**
27: $B_\phi \leftarrow B_\phi \cup \{v_t\}$
28: **while** $|B_\phi| > \eta$ **do**
29: **for all** $v \in B_\phi$ **do** $B_\phi \leftarrow B_\phi \setminus \{v\}$ if $S_\phi \cup \{v\} \notin \xi$
30: $v' \leftarrow \mathrm{argmin}_{v \in B_\phi} \frac{\Delta_f(v|S_\phi)}{\delta(v)}, \delta(v) = \max_{j \in [d]} c_j(v)$
31: $B_\phi \leftarrow B_\phi \setminus \{v'\}$
32: **procedure** CostEffectGreedy(S_ϕ, B_ϕ)
33: **while** $\exists v \in B_\phi : S_\phi \cup \{v\} \in \xi$ **do**
34: $v^* \leftarrow \mathrm{argmax}_{v \in B_\phi \wedge S_\phi \cup \{v\} \in \xi} \frac{\Delta_f(v|S_\phi)}{\delta(v)}$
35: $S_\phi \leftarrow S_\phi \cup \{v^*\}$, $B_\phi \leftarrow B_\phi \setminus \{v^*\}$

in SubKnapChk (Lemma 3). Based on the properties, we prove the approximation ratio of KW$^+$ (Theorem 3). Finally, we analyze the number of checkpoints in SubKnapChk, calculate the cost of buffer maintenance and post-processing for KW$^+$, and acquire the complexity of KW$^+$.

First of all, we prove the properties of the checkpoints in SubKnapChk.

Lemma 3. *Given a parameter $\beta \in (0, 1)$, each checkpoint $x_i \in X_t$ where $i \in [1, s]$ maintained by* SubKnapChk *at time t satisfies one of the following properties:*

1) *if $f[x_{i+1}, t] \geq (1-\beta) f[x_i, t]$, $f[x_{i+2}, t] < (1-\beta) f[x_i, t]$ or $x_{i+1} = x_s$.*
2) *if $x_{i+1} \neq x_i + 1$ and $f[x_{i+1}, t] < (1 - \beta) f[x_i, t]$, there exists some $t' < t$ such that $f[x_{i+1}, t'] \geq (1 - \beta) f[x_i, t']$.*
3) *$x_{i+1} = x_i + 1$ and $f[x_{i+1}, t] < (1 - \beta) f[x_i, t]$.*

Proof. We prove the lemma by induction on t. As the base case, we first check the condition when $t = 2$ and $X_2 = \langle x_1 = 1, x_2 = 2 \rangle$. Then, Property (1) holds if $f[x_2, 2] \geq (1 - \beta) f[x_1, 2]$; otherwise, Property (3) holds.

Next, we assume Lemma 3 holds at time t and show that it still holds after performing Lines 3–15 of Algorithm 3 at time $t+1$. Let x_i be a checkpoint that is created before $t+1$ and not deleted during the maintenance at time $t+1$ and x_{i+1} be the checkpoint next to x_i at time t. We discuss all possible cases during the maintenance at time $t+1$.

Case 1. $x_{i+1} \neq x_i + 1$ and x_{i+1} is deleted from SUBKNAPCHK at time $t+1$. In this case, we have $f[x_{i+2}, t+1] \geq (1-\beta)f[x_i, t+1]$ (Line 13 of Algorithm 3). As x_{i+2} becomes the successor of x_i at time $t+1$, Property (1) holds.

Case 2. $x_{i+1} \neq x_i + 1$ and x_{i+1} is not deleted from SUBKNAPCHK at time $t+1$. In this case, we consider x_{i+1} becomes the successor of x_i at some time $t' \leq t$. Then, it must hold that $f[x_{i+1}, t'] \geq (1-\beta)f[x_i, t']$. Since x_{i+1} is not deleted at time $t+1$, either Property (1) (if $f[x_{i+1}, t+1] \geq (1-\beta)f[x_i, t+1]$) or Property (2) (if $f[x_{i+1}, t+1] < (1-\beta)f[x_i, t+1]$) holds.

Case 3. $x_{i+1} = x_i + 1$. No matter whether x_{i+1} is deleted at time $t+1$, Property (1) holds if $f[x_{i+1}, t+1] \geq (1-\beta)f[x_i, t+1]$; otherwise, Property (3) holds.

We show that the properties of SUBKNAPCHK still hold at time $t+1$ in all possible cases and conclude the proof. □

Given the properties of SUBKNAPCHK, we can analyze the approximation ratio of S_t returned by Algorithm 3 for SMDK w.r.t. A_t.

Theorem 3. *The solution S_t returned by Algorithm 3 satisfies that $f(S_t) \geq \frac{1-\varepsilon'}{2(1+d)} \cdot \mathtt{OPT}_t$ at any time t where $\varepsilon' = \varepsilon + \beta$.*

Proof. We consider the first two checkpoints x_1 and x_2 of SUBKNAPCHK at time t and assume that post-processing does not change the solution S_t. If $t \leq W$, $x_1 = 1$ does not expire and $\mathcal{H}(x_1)$ are maintained over $A_t = \langle v_1, \dots, v_t \rangle$. Thus, $f(S_t) = f[x_1, t] \geq \frac{1-\varepsilon}{1+d}\mathtt{OPT}_t$ for $t \leq W$ by Theorem 1. Next, we consider $t > W$ and $x_2 = x_1 + 1$. In this case, x_1 expires and x_2 corresponds to the starting point of A_t. Similarly, $f(S_t) = f[x_2, t] \geq \frac{1-\varepsilon}{1+d}\mathtt{OPT}_t$.

Subsequently, we consider other cases for $t > W$. We use $\mathtt{OPT}^x_y$ to denote the optimal utility of SMDK w.r.t. $\langle v_x, \dots, v_y \rangle$.

Case 1. If $f[x_2, t] \geq (1-\beta)f[x_1, t]$, $f(S_t) = f[x_2, t] \geq (1-\beta)f[x_1, t]$. By Theorem 1, $f[x_1, t] \geq \frac{1-\varepsilon}{1+d}\mathtt{OPT}^{x_1}_t$. As $x_1 < t'$, we have $A_t \subset \langle v_{x_1}, \dots, v_t \rangle$ and $\mathtt{OPT}_t \leq \mathtt{OPT}^{x_1}_t$. Finally, we have $f(S_t) \geq \frac{(1-\beta)(1-\varepsilon)}{1+d}\mathtt{OPT}_t$.

Case 2. If $f[x_2, t] < (1-\beta)f[x_1, t]$, we have $f[x_2, t'] \geq (1-\beta)f[x_1, t']$ for some $t' < t$. Let $S^*_{x_1,t}$ denote the optimal solution for $\langle v_{x_1}, \dots, v_t \rangle$. We can split $S^*_{x_1,t}$ into two subsets S_1 and S_2, where $S_1 = \{v_i | v_i \in S^*_{x_1,t} \wedge i \in [x_1, t']\}$ and $S_2 = \{v_i | v_i \in S^*_{x_1,t} \wedge i \in [x_2, t]\}$. Let $\mathtt{OPT}_1 = f(S_1)$ and $\mathtt{OPT}_2 = f(S_2)$. For $S^*_{x_1,t} = S_1 \cup S_2$ and the submodularity of $f(\cdot)$, $\mathtt{OPT}^{x_1}_t \leq \mathtt{OPT}_1 + \mathtt{OPT}_2$. Then, as $S_1 \in \xi$ and $S_2 \in \xi$, it holds that $\mathtt{OPT}_1 \leq \mathtt{OPT}^{x_1}_{t'}$ and $\mathtt{OPT}_2 \leq \mathtt{OPT}^{x_2}_t$. In addition, for any $t_1 < t_2$, the solution returned by KNAPSTREAM satisfies that $f[x, t_1] \leq f[x, t_2]$. As $t > t'$, we have:

$$f[x_2, t] \geq \frac{(1-\beta)(1-\varepsilon)}{1+d} \cdot \mathtt{OPT}^{x_1}_{t'} \geq \frac{(1-\beta)(1-\varepsilon)}{1+d} \cdot \mathtt{OPT}_1$$

We also have:

$$f[x_2, t] \geq \frac{1-\varepsilon}{1+d} \cdot \mathtt{OPT}^{x_2}_t \geq \frac{1-\varepsilon}{1+d} \cdot \mathtt{OPT}_2$$

Adding the above two inequalities, we prove:

$$f(S_t) = f[x_2, t] \geq \frac{(1-\beta)(1-\varepsilon)}{2(1+d)} \cdot \mathtt{OPT}_t \quad (4)$$

Finally, because the post-processing procedure must not decrease the utility of any candidate, Equation 4 still holds after post-processing. Thus, we conclude the proof by replacing λ and ε with ε' in Equation 4. □

The Complexity of KW$^+$. According to Lemma 3, either $f[x_{i+1}, t]$ or $f[x_{i+2}, t]$ is less that $(1-\beta)f[x_i, t]$ at any time t. Given $\theta = \frac{f[x_1,t]}{f[x_s,t]}$, the number of checkpoints in SUBKNAPCHK is at most $\lceil \frac{2\log\theta}{\log(1-\beta)^{-1}} \rceil$. Therefore, the number of checkpoints is $\mathcal{O}(\frac{\log\theta}{\beta})$. Accordingly, KW$^+$ performs $\mathcal{O}(\frac{\log\theta \cdot \log(d\cdot\gamma^{-1})}{\varepsilon'^2})$ function calls to update the candidates in the checkpoints for one element and stores at most $\mathcal{O}(\frac{\log\theta\cdot\log(d\cdot\gamma^{-1})}{\gamma\cdot\varepsilon'^2})$ elements within the candidates. In practice, the buffer of each candidate is implemented by a min-heap and the buffer size $\eta = \mathcal{O}(\gamma^{-1})$. The complexity of adding an element to the buffer is $\mathcal{O}(\log\gamma^{-1})$ and dropping elements from the buffer is $\mathcal{O}(\gamma^{-1})$. Thus, the amortized computational cost for buffer maintenance is $\mathcal{O}(\frac{\log\theta\cdot\log(d\cdot\gamma^{-1})}{\varepsilon'^2})$ and the total number of elements in buffers is $\mathcal{O}(\frac{\log\theta\cdot\log(d\cdot\gamma^{-1})}{\gamma\cdot\varepsilon'^2})$. The post-processing for one candidate handles $\mathcal{O}(\gamma^{-1})$ elements and runs at most γ^{-1} iterations. Therefore, the post-processing requires $\mathcal{O}(\frac{\log(d\cdot\gamma^{-1})}{\gamma^2\cdot\varepsilon'})$ function calls. Generally, KW$^+$ runs in $\mathcal{O}\big(\frac{\log(d\cdot\gamma^{-1})}{\varepsilon'} \cdot (\gamma^{-2} + \frac{\log\theta}{\varepsilon'})\big)$ time to process one element and stores $\mathcal{O}(\frac{\log\theta\cdot\log(d\cdot\gamma^{-1})}{\gamma\cdot\varepsilon'^2})$ elements in total.

5.3 Discussion

In practice, it is no need to update the solution for every arrival element. The update is often performed in a batch manner. Specifically, we consider the sliding window receives T new elements while the earliest T elements become expired at time t. Both KW and KW$^+$ can handle the scenario with trivial adaptations. For KW, it also stores the active elements in A_t and creates a checkpoint for every L elements. The only difference is that the interval L becomes $\sqrt{W \cdot T}$ while the number of checkpoints s decreases to $\sqrt{\frac{W}{T}}$. For KW$^+$, it creates one checkpoint at each time t and updates existing checkpoints by processing a batch of elements from v_{t-T+1} to v_t collectively. In this way, the total number of checkpoints created is $\lceil \frac{W}{T} \rceil$. The number of checkpoints in SUBKNAPCHK is determined by the utilities and thus is not affected. In addition, any other theoretical results, the buffer maintenance, and the post-processing procedure are also not affected by these adaptations.

6 EXPERIMENTS

In this section, we report our experimental results for two RSS applications (as presented in Section 3) in real-world datasets. First, we introduce the experimental setup in Section 6.1. Then, we evaluate the effectiveness and efficiency of our proposed frameworks compared with several baselines in Section 6.2.

6.1 Experimental Setup

Datasets. Two real-world datasets are used in our experiments. First, we use the *Twitter* dataset for *social stream summarization* (see Section 3.1). It is collected via the streaming

TABLE 2
The parameters tested in the experiments

Parameter	Values
d	1, 2, 3, 4, 5
c	0.02, **0.04**, 0.06, 0.08, 0.1
W	100k, **200k**, 300k, 400k, 500k
λ	0.05, **0.1**, 0.15, 0.2, 0.25
β	0.05, **0.1**, 0.15, 0.2, 0.25

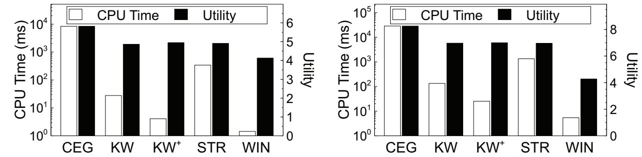
API[2] and contains $18,770,231$ tweets and $8,071,484$ words. The average number of words in each tweet is $\bar{l} = 4.8$ and the average number of followers of each user is $\overline{fl} = 521.4$. In the experiments, we feed each tweet to the compared approaches one by one in ascending order of timestamp. Second, we use the *Yahoo! Webscope* dataset[3] for *active set selection* (see Section 3.2). It consists of $45,811,883$ user visits from the Featured Tab of the Today module on the Yahoo! front page. Each user visit is a 5-dimensional feature vector. We set $h = 0.75$ and $\sigma = 1$ in Equation 3 following [6]. The costs are generated from a uniform distribution $\mathcal{U}(0.02, 0.08)$. In the experiments, we feed all user visits to the compared approaches one by one in the same order.

Additional constraints. To evaluate the compared approaches with varying the dimension of knapsacks, i.e., d, we generate additional constraints by assigning random costs to each element in both datasets. Specifically, we generate a 5-dimensional cost vector $\mathbf{c}(v) = \{c_1(v), \dots, c_5(v)\}$ for each element v. And each cost is generated independently from a uniform distribution $\mathcal{U}(0.02, 0.08)$. We set d to range from 1 to 5 in the experiments and use the first d dimensions of $\mathbf{c}(v)$ for the d-knapsack constraint.

Compared Approaches. The approaches compared in our experiments are listed as follows.

- CostEffectGreedy (CEG). We implement the CostEffectGreedy algorithm [27] as the batch baseline. Since CEG is designed for submodular maximization with a 1-knapsack constraint, we slightly adapt it for SMDK: the cost-effectiveness of v w.r.t. S is computed by $\frac{\Delta_f(v|S)}{\delta(v)}$ where $\delta(v) = \max_{j \in [d]} c(v)$. To work in the sliding window model, it stores the active elements in A_t and recomputes the solution from scratch for each window slide.
- Streaming (STR). We implement the state-of-the-art append-only streaming algorithm for SMDK [13] as a baseline. To work in the sliding window model, it also stores the active elements in A_t and recomputes the solution from scratch for each window slide.
- Window (WIN). We implement the state-of-the-art algorithm for submodular maximization over sliding windows [23] as a baseline. Since it only works with one cardinality constraint, we cast the d-knapsack constraint to the cardinality constraint by setting the budget $k = \frac{1}{\gamma}$ where γ is the average cost of elements. When maintaining the solutions over sliding windows, it only considers the marginal gains of elements and treats the cost of any element as 1.
- KnapWindow (KW). We implement the KnapWindow framework in Section 4.

2. http://twitter4j.org/en/index.html
3. http://webscope.sandbox.yahoo.com



(a) Twitter (b) Yahoo! Webscope

Fig. 4. The overall experimental results.

- KnapWindowPlus (KW$^+$). We implement the KnapWindowPlus framework in Section 5. We set $\alpha = 0.5$ and $\eta = 20$ for buffer maintenance.

Parameters. The parameters tested in our experiments are listed in Table 2 with default values in bold. d is the dimension of the knapsack constraint. We use $d = 3$ for social stream summarization and $d = 1$ for active set selection by default as introduced in Section 3; c is the average cost of each element. For the *Twitter* dataset, we set $k = \frac{1}{c}$ to assign the costs $c_1(v), c_2(v), c_3(v)$ accordingly as introduced in Section 3.1. For the *Yahoo! Webscope* dataset, the average of generated costs is $c = 0.05$. We scale the costs linearly in the experiments for varying c. W is the size of the sliding window. We set the number of elements for each window slide to $T = 0.01\% \cdot W$. The interval for neighboring checkpoints in KW is $L = \sqrt{W \cdot T} = 1\% \cdot W$ (Section 5.3). λ is the parameter used in KW, KW$^+$, STR, and WIN for the balance between the number of candidates maintained for processing append-only streams and solution quality. β is the parameter for KW$^+$ to balance between the number of checkpoints and solution quality.

Metrics. We consider the following metrics to evaluate the compared approaches.

- **CPU time** is the average CPU time of an approach to process one window slide. It is used to measure the efficiency of compared approaches.
- **Utility** is the average utility value of the solution returned by an approach for each window. It evaluates the solution quality of compared approaches.
- **#checkpoints** and **#elements** are the average numbers of checkpoints and elements maintained by KW$^+$, which are used to measure its space usage.

Experimental Environment. All the above approaches are implemented in Java 8 and the experiments are conducted on a server running Ubuntu 16.04 with a 1.9GHz Intel Xeon E7-4820 processor and 128 GB memory.

6.2 Experimental Results

Overall Results. In Figure 4, we present the *CPU time* and *utilities* of compared approaches in the default setting. Although CEG achieves the best utilities, it takes around 10s to process each window slide, which is far lower than the rates of real-world data streams. KW and KW$^+$ run over two and three orders of magnitude faster than CEG respectively and can process each window slide within 100ms. Meanwhile, the utilities of the solutions provided by KW and KW$^+$ are about 85% of those of CEG. Furthermore, KW$^+$ significantly improves the efficiency upon KW, achieving speedups of at least 6x in both datasets. Compared with STR, KW and KW$^+$ run dozens of times faster while providing solutions

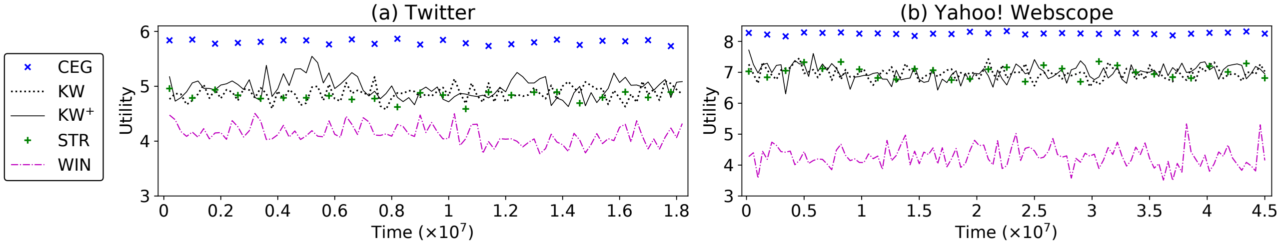


Fig. 5. The utilities of compared approaches over time. Note that we retrieve the solutions of CEG and STR only at sampled timestamps. The solutions of KW, KW$^+$, and WIN are returned for every window slide.

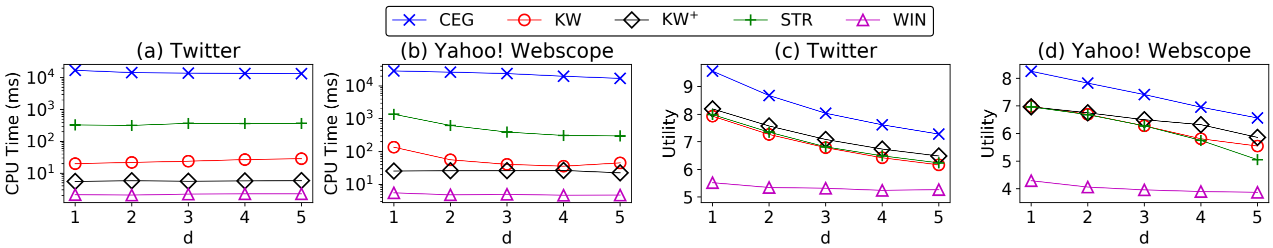


Fig. 6. The CPU time and utilities of compared approaches with varying the dimension d of the knapsack constraint.

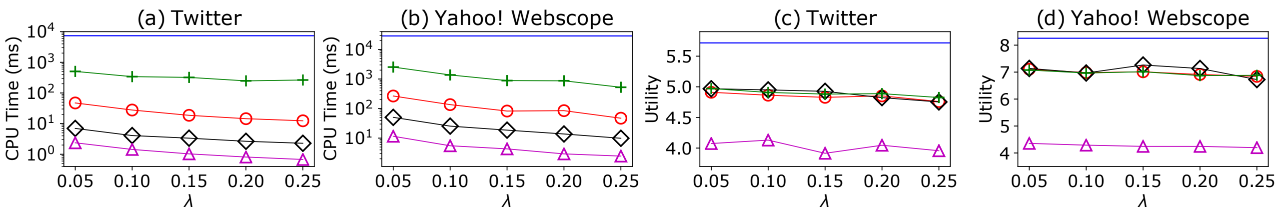


Fig. 7. The CPU time and utilities of compared approaches with varying the parameter λ. Note that CEG is not affected by λ. We use horizontal blue lines to represent the CPU time and utilities of CEG for ease of comparison.

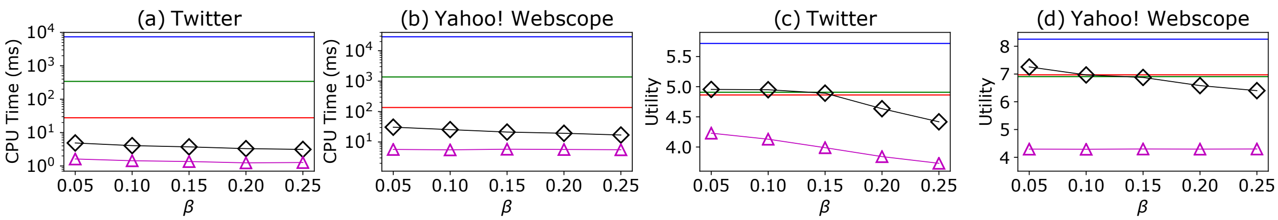


Fig. 8. The CPU time and utilities of compared approaches with varying the parameter β. Note that CEG, KW, and STR are not affected by β. For ease of comparison, we use horizontal blue, red, and green lines to represent the CPU time and utilities of CEG, KW, and STR respectively.

with similar utilities. Finally, we observe WIN runs faster than other approaches but shows obviously inferior solution quality. This is because WIN treats the costs of any element equally and only considers marginal utility gains when adding an element. As a result, the solutions of WIN contain fewer elements than other approaches, which leads to both higher efficiency and worse solution quality.

In Figure 5, we present the utilities of compared approaches from time $t = W$ to the end of the stream $t = n$. The solutions returned by CEG achieve the highest utilities all the time. the solution utilities of KW, KW$^+$, and STR fluctuate over time and are generally close to each other. But remember that KW$^+$ takes much less CPU time than KW while KW runs significantly faster than STR (as illustrated in Figure 4). Also as expected, the solution quality of WIN cannot match any other approaches.

To sum up, KW$^+$ achieves the best balance between efficiency and solution quality: compared with CEG, it runs more than three orders of magnitude faster while providing solutions with 85% average utility; it has much higher efficiency than KW and STR but achieves equivalent solution quality; it significantly improves the solution quality upon WIN at a little expense of efficiency.

Effect of d. The CPU time and utilities of compared approaches with varying d are shown in Figure 6. The CPU time of CEG decreases when d increases. This is because the average solution size becomes smaller when there are more constraints. The CPU time of KW shows different trends in both datasets: it decreases in the *Yahoo! Webscope* dataset but keeps steady in the *Twitter* dataset when d becomes larger. There are two observations behind such trends: First, since KS maintains the candidates for estimations from m to $M(1+d)$ (see Algorithm 1), a KS instance maintains more candidates with increasing d. Second, the average solution size decreases with d. In the *Twitter* dataset, the extra costs for maintaining more candidates cancel out the benefits of smaller solutions and thus the overall CPU time keeps steady. However, in the *Yahoo! Webscope* dataset, the time complexity of evaluating IVM in Equation 3 for a set S is $\mathcal{O}(|S|^3)$. As a result, the CPU time for each IVM evaluation is very sensitive to $|S|$. Although more candidates are maintained, the overall CPU time of KW still becomes much

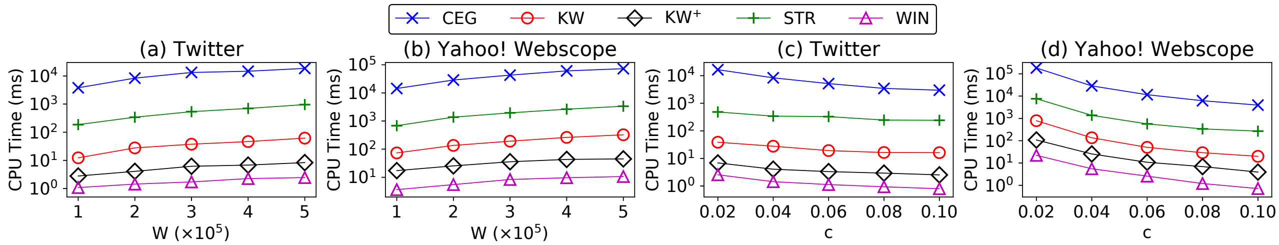


Fig. 9. The CPU time of compared approaches with varying the window size W and the average cost c.

TABLE 3
The number of checkpoints and elements (including candidates and buffers) maintained by KW$^+$.

Dataset	Parameter	W					c				
	Value	100k	200k	300k	400k	500k	0.02	0.04	0.06	0.08	0.1
Twitter	#checkpoints	4.89	4.49	4.32	4.34	4.27	4.44	4.49	4.15	4.42	4.68
	#elements	3949.9	3770.4	3674.2	3725.3	3676.1	5489.4	3770.4	2843.4	2618.1	2509.5
Yahoo! Webscope	#checkpoints	4.44	3.58	3.62	3.5	2.8	4.4	3.58	3.18	3.44	2.7
	#elements	3258.6	2617.54	2735.38	2647.16	2019.9	6036.86	2617.54	1680.88	1454.92	908.16

lower. The CPU time of KW$^+$ shows a similar trend to KW in the *Twitter* dataset. But it keeps steady with increasing d in the *Yahoo! Webscope* dataset. The reason behind such an observation is, although the CPU time to update the checkpoints decreases, the post-processing takes longer time when d increases. The utilities of all compared approaches decrease when d increases because of smaller solution sizes. Compared with KW and STR, KW$^+$ shows slightly better solution quality for a larger d due to the benefits of post-processing. In addition, the ratios between the utilities of the solutions of STR, KW, and KW$^+$ and those of CEG are 84%–90% and remain stable for different d.

Robustness against λ and β. The experimental results of compared approaches with varying parameters λ are shown in Figure 7. For all compared approaches except CEG, the CPU time obviously drops with increasing λ. This is because the number of candidates is inversely correlated to λ. However, we observe that their utilities are rather robust against λ and only slightly decrease for a larger λ. The utility of KW$^+$ in the *Yahoo! Webscope* dataset even increases when $\lambda = 0.15$ thanks to the post-processing.

The experimental results of compared approaches with varying parameters β are shown in Figure 8. Because the number of checkpoints and KS instances in SubKnapChk is inversely correlated to β, the CPU time of KW$^+$ decreases when β increases. However, the robustness of KW$^+$ against β is worse than its robustness against λ. The utilities show drastic drops when $\beta = 0.2$ or 0.25. As the intervals between the first two checkpoints increase with β, the errors of using the results from the second checkpoint as the solutions inevitably increase. Considering the results, we advise using a small β so that KW$^+$ can achieve good solution quality.

Scalability. In Figure 9, we present the CPU time of compared approaches with varying W and c. The CPU time to process each window slide increases with W. This is because the number of elements processed for each window slide is set to $0.01\% \cdot W$ which increases linearly with W. For all compared approaches, it takes a longer CPU time when c decreases because the solution size is inversely proportional to c. In the *Yahoo! Webscope* dataset, the CPU time increases drastically when c decreases because the time complexity of the IVM function evaluation is $\mathcal{O}(|S|^3)$. Thus, all compared approaches spend much more CPU time for each evaluation of $f(S)$ when the solution size grows. Nevertheless, the CPU time of KW$^+$ and that of KW are within 100ms and 1s respectively in all parameter settings.

We list the number of checkpoints and the number of elements in both candidates and buffers maintained by KW$^+$ with varying W and c in Table 3. First, because the number of checkpoints and the number of elements are independent of W and bounded by the ratio of the utilities of the solutions provided by the first and last checkpoints, both metrics hardly increases with W. In addition, the number of elements in KW$^+$ increases when c decreases because each candidate maintains more elements. Generally, KW$^+$ only stores several thousand elements when W ranges from 100k to 500k. Taking $W = 500$k as an example, KW$^+$ merely stores 0.7% of the active elements. Therefore, the space usage of KW$^+$ is much smaller than KW, CEG, and STR, which need to store the entire active window. Furthermore, the number of elements maintained by KW$^+$ does not increase with the window size W because the space complexity of KW$^+$ is independent of W. Hence, KW$^+$ is scalable for large window sizes.

7 Related Work

Representative subset selection (RSS) is an important tool to draw insights from massive datasets. Existing RSS techniques can be categorized into four classes based on the utility functions used to evaluate the representativeness: (1) *coverage-based RSS* [3]–[5], [8], [12], [20], [21]; (2) *entropy-based RSS* [2], [6], [7], [23]; (3) *clustering-based RSS* [2], [6], [9], [10]; (4) *diversity-aware RSS* [3], [11], [21]. Coverage-based approaches treat RSS as the maximum coverage problem [20] and its variants, e.g., budgeted coverage [3], [21], weighted coverage [8], [12], and probabilistic coverage [5]. They consider all information in a dataset as a collection of *information units*. The objective of RSS is to select a subset of elements so as to maximally cover the *information units* in the source dataset. Entropy-based RSS [2], [6], [7], [23] (a.k.a. active set selection) aims to select a subset of elements with the highest information entropy. Active set selection is

TABLE 4
A theoretical comparison of existing submodular maximization algorithms. The algorithms proposed in this work are highlighted by *.

Algorithm	Data model	Constraint	Approximation	Time complexity
Sviridenko [34]	batch	1-knapsack	$1-\frac{1}{e}$	$\mathcal{O}(W^5)$
Kulik et al. [28]	batch	d-knapsack	$1-\frac{1}{e}-\varepsilon$	$\mathcal{O}(W^{d\cdot\varepsilon^{-4}})$
Badanidiyuru et al. [35]	batch	1-knapsack	$1-\frac{1}{e}-\varepsilon$	$\mathcal{O}(W^2\cdot(\varepsilon^{-1}\cdot\log W)^{\varepsilon^{-8}})$
Leskovec et al. [27] & Lin et al. [3]	batch	1-knapsack	$\frac{1}{2}(1-\frac{1}{e})$	$\mathcal{O}(\gamma^{-1}\cdot W)$
Badanidiyuru et al. [6] & Kumar et al. [32]	append-only stream	cardinality	$\frac{1}{2}-\varepsilon$	$\mathcal{O}(\frac{\log k}{\varepsilon})$
Huang et al. [36]	append-only stream	1-knapsack	$\frac{4}{11}-\varepsilon$	$\mathcal{O}((\frac{\log\gamma^{-1}}{\varepsilon})^4)$
Yu et al. [13]	append-only stream	d-knapsack	$\frac{1}{1+2d}-\varepsilon$	$\mathcal{O}(\frac{\log\gamma^{-1}}{\varepsilon})$
Epasto et al. [23]	sliding window	cardinality	$\frac{1}{3}-\varepsilon$	$\mathcal{O}(\frac{\log^2(k\cdot\theta)}{\varepsilon^2})$
KnapStream (KS)*	append-only stream	d-knapsack	$\frac{1-\varepsilon}{1+d}$	$\mathcal{O}(\frac{\log(d\cdot\gamma^{-1})}{\varepsilon})$
KnapWindow (KW)*	sliding window	d-knapsack	$\frac{1-\varepsilon}{1+d}$	$\mathcal{O}(\frac{\sqrt{W}\cdot\log(d\cdot\gamma^{-1})}{\varepsilon})$
KnapWindowPlus (KW$^+$)*	sliding window	d-knapsack	$\frac{1-\varepsilon'}{2+2d}$	$\mathcal{O}(\frac{\log(d\cdot\gamma^{-1})}{\varepsilon'}\cdot(\gamma^{-2}+\frac{\log\theta}{\varepsilon'}))$

considered as a powerful tool for large-scale nonparametric learning [2], [6]. Clustering-based RSS [2], [6], [9], [10] (a.k.a. exemplar clustering) selects a subset of elements such that the average distance from the remaining elements in the dataset to their nearest neighbor in the selected subset is minimized. Diversity-aware RSS [3], [11], [21] integrates a coverage/clustering based utility function with a diversity function to avoid including highly similar elements into the selected subset. Generally, the utility functions used in the aforementioned RSS problems are all *submodular* because the representativeness naturally satisfies the "diminishing returns" property. But most of them [3]–[5], [7]–[11], [21] can only work in the batch setting and are very inefficient to process data streams.

Recently, we have witnessed the growth of RSS studies in the data stream model. RSS in append-only streams where new elements arrive continuously but old ones never expire is studied in [2], [6], [12], [20]. Mirzasoleiman et al. [37] further propose a method for deletion-robust RSS where a limited number of old elements can be deleted from the stream. However, these techniques neither support general constraints beyond cardinality nor consider the recency of selected subsets. In many scenarios, data streams are highly dynamic and evolve over time. Therefore, recent elements are more important and interesting than earlier ones. The *sliding window* [22] model is widely adopted in many data-driven applications [38], [39] to capture the recency constraint. RSS over sliding windows is still largely unexplored yet and, to the best of our knowledge, there is only one existing method [23] for dynamic RSS over sliding windows. But it is specific for the cardinality constraint. In this paper, we propose more general frameworks for RSS than any existing ones, which work with various submodular utility functions, support d-knapsack constraints, and maintain the representatives over sliding windows.

Submodular maximization (SM) has been extensively studied in recent years. Due to its theoretical consequences, SM is seen as a "silver bullet" for many different applications [13], [27], [39]–[41]. Here, we focus on reviewing existing literature on SM that is closely related to our paper: SMDK and SM in data streams. Sviridenko [34] and Kulik et al. [28] first propose approximation algorithms for SM subject to 1-knapsack and d-knapsack constraints respectively. Both algorithms have high-order polynomial time complexity and are not scalable to massive datasets. More efficient algorithms for SM subject to 1-knapsack constraints are proposed in [3], [27] and [35] respectively. These algorithms cannot be applied to SMDK directly. Badanidiyuru et al. [6] and Kumar et al. [32] propose the algorithms for SM with cardinality constraints in append-only streams with sublinear time complexity. Then, Huang et al. [36] propose an algorithm for SM in append-only streams with 1-knapsack constraints. Yu et al. [13] propose an algorithm for SMDK in append-only streams. More recently, there are a few attempts at SM over sliding windows. Epasto et al. [23] propose an algorithm for SM over sliding windows with cardinality constraints. To the best of our knowledge, there is no existing literature on SMDK over sliding windows yet.

We compare the above SM algorithms theoretically in Table 4. We present their data models, supported constraints, approximation factors, and time complexities respectively. According to the results, our contributions in this paper are two-fold: (1) KS improves the approximation factor of SMDK in append-only streams from $\frac{1}{1+2d}-\varepsilon$ to $\frac{1-\varepsilon}{1+d}$; (2) KW and KW$^+$ are among the first algorithms for SMDK in the sliding window model.

8 Conclusion

In this paper, we studied the *representative subset selection* (RSS) problem in data streams. First of all, we formulated dynamic RSS as maximizing a monotone submodular function subject to a d-knapsack constraint (SMDK) over sliding windows. We then devised the KW framework for this problem. Theoretically, KW provided solutions for SMDK over sliding windows with an approximation factor of $\frac{1-\varepsilon}{1+d}$. Furthermore, we proposed a more efficient $\frac{1-\varepsilon'}{2+2d}$-approximation KW$^+$ framework for SMDK over sliding windows. The experimental results demonstrated that KW and KW$^+$ run orders of magnitude faster than the batch baseline while preserving high-quality solutions.

References

[1] Y. Wang, Y. Li, and K.-L. Tan, "A sliding-window framework for representative subset selection," in *ICDE*, 2018, pp. 1268–1271.

[2] R. Gomes and A. Krause, "Budgeted nonparametric learning from data streams," in *ICML*, 2010, pp. 391–398.

[3] H. Lin and J. A. Bilmes, "A class of submodular functions for document summarization," in *ACL*, 2011, pp. 510–520.
[4] K. Wei, Y. Liu, K. Kirchhoff, and J. A. Bilmes, "Using document summarization techniques for speech data subset selection," in *NAACL-HLT*, 2013, pp. 721–726.
[5] J. Xu, D. V. Kalashnikov, and S. Mehrotra, "Efficient summarization framework for multi-attribute uncertain data," in *SIGMOD*, 2014, pp. 421–432.
[6] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause, "Streaming submodular maximization: massive data summarization on the fly," in *KDD*, 2014, pp. 671–680.
[7] K. Wei, R. K. Iyer, and J. A. Bilmes, "Submodularity in data subset selection and active learning," in *ICML*, 2015, pp. 1954–1963.
[8] H. Zhuang, R. Rahman, X. Hu, T. Guo, P. Hui, and K. Aberer, "Data summarization with social contexts," in *CIKM*, 2016, pp. 397–406.
[9] E. M. Lindgren, S. Wu, and A. G. Dimakis, "Leveraging sparsity for efficient submodular data summarization," in *NIPS*, 2016, pp. 3414–3422.
[10] D. Malioutov, A. Kumar, and I. E.-H. Yen, "Large-scale submodular greedy exemplar selection with structured similarity matrices," in *UAI*, 2016.
[11] B. Mirzasoleiman, A. Badanidiyuru, and A. Karbasi, "Fast constrained submodular maximization: Personalized data summarization," in *ICML*, 2016, pp. 1358–1367.
[12] N. T. Tam, M. Weidlich, D. C. Thang, H. Yin, and N. Q. V. Hung, "Retaining data from streams of social platforms with minimal regret," in *IJCAI*, 2017, pp. 2850–2856.
[13] Q. Yu, E. L. Xu, and S. Cui, "Submodular maximization with multi-knapsack constraints and its applications in scientific literature recommendations," in *GlobalSIP*, 2016, pp. 1295–1299.
[14] Q. Fan, Y. Li, D. Zhang, and K.-L. Tan, "Discovering newsworthy themes from sequenced data: A step towards computational journalism," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 7, pp. 1398–1411, 2017.
[15] Y. Li, D. Zhang, Z. Lan, and K.-L. Tan, "Context-aware advertisement recommendation for high-speed social news feeding," in *ICDE*, 2016, pp. 505–516.
[16] W. Guo, Y. Li, M. Sha, and K.-L. Tan, "Parallel personalized pagerank on dynamic graphs," *PVLDB*, vol. 11, no. 1, pp. 93–106, 2017.
[17] M. Sha, Y. Li, B. He, and K.-L. Tan, "Accelerating dynamic graph analytics on gpus," *PVLDB*, vol. 11, no. 1, pp. 107–120, 2017.
[18] Y. Li, Z. Bao, G. Li, and K.-L. Tan, "Real time personalized search on social networks," in *ICDE*, 2015, pp. 639–650.
[19] D. Zhang, Y. Li, J. Fan, L. Gao, F. Shen, and H. T. Shen, "Processing long queries against short text: Top-k advertisement matching in news stream applications," *ACM Trans. Inf. Syst.*, vol. 35, no. 3, pp. 28:1–28:27, 2017.
[20] B. Saha and L. Getoor, "On maximum coverage in the streaming model & application to multi-topic blog-watch," in *SDM*, 2009, pp. 697–708.
[21] H. Lin and J. A. Bilmes, "Multi-document summarization via budgeted maximization of submodular functions," in *NAACL-HLT*, 2010, pp. 912–920.
[22] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM J. Comput.*, vol. 31, no. 6, pp. 1794–1813, 2002.
[23] A. Epasto, S. Lattanzi, S. Vassilvitskii, and M. Zadimoghaddam, "Submodular optimization over sliding windows," in *WWW*, 2017, pp. 421–430.
[24] S. Khuller, A. Moss, and J. Naor, "The budgeted maximum coverage problem," *Inf. Process. Lett.*, vol. 70, no. 1, pp. 39–45, 1999.
[25] 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.
[26] U. Feige, "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, no. 4, pp. 634–652, 1998.
[27] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. Van Briesen, and N. S. Glance, "Cost-effective outbreak detection in networks," in *KDD*, 2007, pp. 420–429.
[28] A. Kulik, H. Shachnai, and T. Tamir, "Maximizing submodular set functions subject to multiple linear constraints," in *SODA*, 2009, pp. 545–554.
[29] M. Feldman, J. Naor, and R. Schwartz, "A unified continuous greedy algorithm for submodular maximization," in *FOCS*, 2011, pp. 570–579.
[30] N. D. Lawrence, M. W. Seeger, and R. Herbrich, "Fast sparse gaussian process methods: The informative vector machine," in *NIPS*, 2002, pp. 609–616.
[31] N. Cuong and H. Xu, "Adaptive maximization of pointwise submodular functions with budget constraint," in *NIPS*, 2016, pp. 1244–1252.
[32] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani, "Fast greedy algorithms in mapreduce and streaming," *ACM Trans. Parallel Comput.*, vol. 2, no. 3, pp. 14:1–14:22, 2015.
[33] V. Braverman and R. Ostrovsky, "Smooth histograms for sliding windows," in *FOCS*, 2007, pp. 283–293.
[34] 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.
[35] A. Badanidiyuru and J. Vondrák, "Fast algorithms for maximizing submodular functions," in *SODA*, 2014, pp. 1497–1514.
[36] C.-C. Huang, N. Kakimura, and Y. Yoshida, "Streaming algorithms for maximizing monotone submodular functions under a knapsack constraint," in *APPROX-RANDOM*, 2017, pp. 11:1–11:14.
[37] B. Mirzasoleiman, A. Karbasi, and A. Krause, "Deletion-robust submodular maximization: Data summarization with "the right to be forgotten"," in *ICML*, 2017, pp. 2449–2458.
[38] X. Wang, Y. Zhang, W. Zhang, X. Lin, and Z. Huang, "Skype: Top-k spatial-keyword publish/subscribe over sliding window," *PVLDB*, vol. 9, no. 7, pp. 588–599, 2016.
[39] Y. Wang, Q. Fan, Y. Li, and K.-L. Tan, "Real-time influence maximization on dynamic social streams," *PVLDB*, vol. 10, no. 7, pp. 805–816, 2017.
[40] Y. Li, J. Fan, D. Zhang, and K.-L. Tan, "Discovering your selling points: Personalized social influential tags exploration," in *SIGMOD*, 2017, pp. 619–634.
[41] Y. Li, J. Fan, Y. Wang, and K.-L. Tan, "Influence maximization on social graphs: A survey," *IEEE Trans. Knowl. Data Eng.*, 2018.

Yanhao Wang is currently a PhD student at the School of Computing, National University of Singapore (NUS). He received the B.Eng. and M.Eng. degrees in computer science from Shandong University, Jinan, China and Renmin University of China, Beijing, China, in 2011 and 2014, respectively. His research interests include data stream algorithms, social network analytics and spatial-temporal databases.

Yuchen Li is an assistant professor at the School of Information Systems, Singapore Management University (SMU). Before joining SMU, he was a research fellow in the School of Computing, National University of Singapore (NUS). He received the double B.Sc. degrees in applied math and computer science (both degrees with first class honors) and the PhD degree in computer science from NUS, in 2013 and 2016, respectively. His research interests include graph analytics and heterogeneous computing.

Kian-Lee Tan is a professor at the School of Computing, National University of Singapore (NUS). He received his PhD degree in computer science in 1994 from NUS. His current research interests include query processing and optimization in multiprocessor and distributed systems, database performance, data analytics, and database security. He was a co-recipient of Singapore's President Science Award in 2011. He is also a 2013 IEEE Technical Achievement Award recipient. He is an associate editor of the ACM Transactions on Database Systems (TODS) and the World Wide Web Journal. He has also served in the editorial board of the Very Large Data Base (VLDB) Journal (associate editor: 2007-2009; editor-in-chief: 2009-2015) and the IEEE Transactions on Knowledge and Data Engineering (2009-2013). Kian-Lee is a member of ACM and IEEE (and IEEE CS).