Finding Simplex Items in Data Streams

Zhuochen Fan*, Jiarui Guo*, Xiaodong Li*, Tong Yang*†, Yikai Zhao*,
Yuhan Wu*, Bin Cui*, Yanwei Xu‡, Steve Uhlig§, Gong Zhang‡
*School of Computer Science, and National Engineering Laboratory for Big Data Analysis Technology and Application, Peking University, Beijing, China †Peng Cheng Laboratory, Shenzhen, China
‡Theory Lab, Central Research Institute, 2012 Labs, Huawei Technologies Co., Ltd., Hong Kong SAR, China
§School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK

***Abstract*—In this paper, we propose a new type of item in data streams, called simplex items. Simplex items have frequencies in consecutive p windows that can be approximated by a polynomial of degree at most k, where $k = 0, 1, 2$. These low-order representable simplex items have a wide range of potential applications. For example, when $k = 1$, we can leverage these items whose frequency has obvious linear increase or decrease to speed up the running time of a class of machine learning models and detect network attacks such as distributed denial-of-service (DDoS), *etc.* To find k-degree simplex items in real time, we propose a novel sketch, namely X-Sketch, to accurately record simplex items in a compact space. The key idea of X-Sketch is to effectively filter out non-simplex items with less memory overhead, and then monitor the remaining potential simplex items and keep those items with more consecutive windows. We conduct extensive experiments, and the experimental results show that the F1 Score of X-Sketch is on average 68.6%, 57.9%, and 42.2% higher than the baseline solution for $k = 0, 1, 2$, respectively. Finally, we also provide a case study that applies X-Sketch to "accelerate" the two machine learning models through end-to-end experiments. We have released our source code at GitHub.**

I. Introduction

A. Background and Motivation

Nowadays, the tasks of frequency estimation and finding frequent items in data streams have been well studied by the research community [1]–[6]. Sketches, as a kind of probabilistic data structures, have gained widespread acceptance for these tasks because they can well handle large-scale and high-speed data streams with limited memory overhead and small errors [7]–[10].

However, we find that patterns in which item frequencies present in a certain number of *consecutive* windows are also worth exploring, but have not been investigated. To better describe these patterns, we first divide the data stream into multiple equal-sized and contiguous windows. Then, we propose that such patterns in p continuous windows may be fitted by k-degree polynomials, and only consider the low-order polynomials of $k = 0, 1, 2$, because constant ($k = 0$), primary ($k = 1$), and quadratic ($k = 2$) functions are the most basic function forms that are most common to represent in both mathematics and nature. We name such low-order expressible items as *k-simplex items*, as shown in Figure 1. For the specific mathematical definition of k-simplex items, see Section II-A for details. Next, we list the use cases of k-simplex items:

Co-primary authors: Zhuochen Fan, Jiarui Guo, and Xiaodong Li. Corresponding author: Tong Yang (yangtongemail@gmail.com).

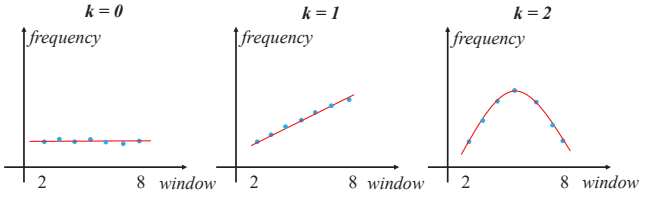


Fig. 1: Examples of k-simplex items, where $p = 7$. The blue dots are the frequencies of the items in their respective windows, and the red lines are the corresponding fitted k-degree polynomials.

When $k = 0$, we are looking for items whose frequencies remain essentially stable over p consecutive windows, and call them 0-simplex items. Here are two example applications for finding such items. 1) If we consider a cache line as an item [11], then 0-simplex items mean that these stable cache lines will be fetched in the near future [12], [13]. Therefore, we can apply 0-simplex items to prefetch the upcoming cache line, thereby improving the cache hit ratio. 2) If we consider a network flow[1] as an item, we can precisely pre-allocate bandwidth for such stable flows in the next time period for network management [14]–[16].

When $k = 1$, we are looking for items whose frequencies increase or decrease linearly over p consecutive windows, and call them 1-simplex items. Here are two example applications for finding such items. 1) We can consider the slopes of the 1-simplex items as important features for the input of machine learning models. In particular, we can speed up the running of some machine learning models (*e.g.*, regression [17], ARIMA [18], *etc.*) by finding 1-simplex items in data streams in advance, so as to greatly reduce the time required. 2) Some network attacks such as distributed denial-of-service (DDoS) have traffic patterns that can be described linearly after processing [19]–[21]. Finding 1-simplex items can help to detect such attacks dynamically in real time.

When $k = 2$, we are looking for items whose frequencies are parabola shaped over p consecutive windows, and call them 2-simplex items. Here is an example applications for finding such items. Periodic 2-simplex items are considered to be the main traffic patterns generated in some wireless networks

[1]A flow is generally defined as a part of the five tuple: source IP address, destination IP address, source port, destination port, and protocol.

($e.g.$, adopting IEEE 802.15.4 MAC protocol) [22], so we can dynamically monitor such traffic to judge the performance of the corresponding networks.

To the best of our knowledge, no existing literature has given the same or similar definitions for k-simplex items, and there are no direct solutions for finding k-simplex items in data streams, but this can be addressed by adapting combinations of existing solutions. Here, we provide a baseline solution consisting mainly of multiple CM sketches [23], each of which can separately record the frequency of the items in its corresponding window and perform polynomial fitting after the k-simplex definition (Section II-A2) is satisfied. Unfortunately, this baseline solution is unsatisfactory in terms of accuracy and memory efficiency, and requires the cooperation of additional data structures such as hash tables [24], as detailed in Section III-A. Therefore, the goal of this paper is to propose a sketch algorithm that is generic, accurate and compact for specialized finding k-simplex items of $k = 0, 1, 2$ in real time in high-speed data streams.

B. Proposed Solution

Towards the design goal, we propose a novel sketch algorithm, called Simplex-Sketch (**X-Sketch** for short), for accurately and efficiently finding k-simplex items in real time. X-Sketch is generic: It only needs one X-Sketch to find the three types of k-simplex items with $k = 0, 1, 2$. X-Sketch is compact and memory efficient: It only requires 150KB of memory consumption when processing 30 Million items. X-Sketch is accurate: Compared with the baseline solution, X-Sketch improves the F1 Score by 68.6%, 57.9%, and 42.2% on average for $k = 0, 1, 2$, respectively.

X-Sketch includes two parts: Stage 1 and Stage 2. For each input item, Stage 1 uses a more compact data structure in conjunction with our *Short-Term Filtering* technique to obtain the potential simplex items. Stage 2 further identifies simplex items accurately through our *Weight Election* technique. Now, we introduce these techniques as follows.

Short-Term Filtering. We find that items that are simplex in p consecutive windows show a certain simplex pattern in the first s windows ($s < p$). Therefore, we design a processing mechanism to judge the likelihood of becoming simplex items by observing whether the items are simplex in the first s windows. In addition, the proportion of non-simplex items is quite large, while simplex items generally account for very little in data streams. For example, in the IP Trace Dataset [25], simplex items represent only for about 0.44%, 0.018% and 0.0068% of all items, for $k = 0, 1, 2$, respectively. It would be very memory inefficient to process all items, so we integrate the above mechanism by tailoring TowerSketch [26] to effectively filter out non-simplex items. After Short-Term Filtering, we further design a reasonable indicator called *Potential* for each remaining item as an optimization, to further evaluate whether it can become a true simplex item after entering Stage 2. In this way, we screen again to get potential simplex items that can be delivered to Stage 2.

Weight Election. The more ideal a simplex item is, the longer the number of consecutive windows it has. Based on this design philosophy, we define a weight for each item entering Stage 2, the size of which is the number of consecutive windows of the item. Further, we design a suitable replacement strategy based on the weight to favor/keep those items with higher weights as efficiently as possible. More details are provided in Section III-D.

Further, we analyse our scheme rigorously in Section IV. Then, we conduct extensive experiments on real-world streams and the synthetic dataset. The results show that X-Sketch has clear advantages over the baseline scheme in finding k-simplex items in data streams. See Section V for details. Finally, we apply X-Sketch to reduce the running time of the linear regression and ARIMA models mostly by more than 100× in Section VI. All related codes are released at GitHub [27].

II. Problem Statement And Related Work

A. Problem Statement

The symbols frequently used in this paper and their meanings are shown in Table I in our Appendices [28].

1) Definitions of Data Streams and Windows:

Definition 1. ***Data Stream Model.*** *A data stream $\mathcal{S} = (e_1, e_2, e_3, \ldots, e_i, \ldots)$ is an unbounded sequence of items appearing one by one. Each item may appear more than once, and the number of occurrences is the frequency.*

Definition 2. ***Count-based Window Model.*** *We divide the data stream $\mathcal{S}$ into many equal-sized and continuous windows, and the window size is defined as a fixed number of items.*

2) Definition of K-Simplex Items:

An item e is called k-simplex item from window w if and only if its frequencies in p consecutive windows $f_w, f_{w+1}, \cdots, f_{w+p-1}$ satisfy:

(1) $f_{w+i} > 0, \forall i = 0, 1, \cdots, p-1$;

(2) There exists a k^{th}-degree polynomial

$$f(n) = a_0 + a_1 n + a_2 n^2 + \cdots + a_k n^k = \sum_{j=0}^{k} a_j n^j,$$

such that the mean squared error (MSE) ε satisfies

$$\varepsilon = \frac{1}{p} \sum_{i=0}^{p-1} (f(i) - f_{w+i})^2 \leq T;$$

where T is a predefined threshold.

B. Related Work

1) Prior Art:

In fact, there is no work directly related to simplex items. Therefore, we first introduce finding persistent items, which looks a bit like finding 0-simplex items but is actually quite different. Then, we introduce the sketch algorithms related to frequency estimation, because finding simplex items requires first estimating the frequency of items in each window.

The relationship and difference between simplex and persistent items. Items are defined as persistent items if the number of windows in which they appear exceeds a predefined threshold [29]–[36], and simplex items are also related to the number of windows in which the items appear. However, the statistical process of a persistent item is only related to whether the item has appeared in a window, regardless of how many times it has appeared in this window, and is

also completely independent of whether these windows are continuous or whether the item frequencies satisfy a k-degree polynomial over these windows ($i.e.$, the *Definition* above).

2) Frequency Estimation:

Simple sketches: The CM sketch (CM) [23] is composed of d counter arrays, each associated with a hash function and having a certain number of counters. When inserting an item, CM first maps it to a counter in each array by computing the hash function, and then increments all d mapped counters by 1. When querying an item, CM reports the minimum value among the d mapped counters. The CU sketch (CU) [37] is similar to CM, except that CU only increments the minimum value among the d mapped counters by 1 when inserting the item, and does not support deletion. Other common schemes include sketches of Count [38] and CSM [39].

Advanced sketches: 1) Frequency estimation for *frequent items*. Cold Filter [40] is a double-layer CU that that first records the frequencies of all items, then filters infrequent items and leaves frequent items by setting a threshold. LogLog Filter [41] inherits Cold Filter and replaces CU with LogLog structure [42], [43] to filter a wider range of infrequent items. Other state-of-the-art schemes include PyramidSketch [44], MV-Sketch [45], and ElasticSketch [46], *etc.* 2) Frequency estimation for *per-item*. TowerSketch (Tower) [26] uses different-sized counters for different arrays while allocating the same memory for each array, and supports both CM and CU insertion. Higher-level arrays therefore have fewer counters, while its counters have larger sizes. In this way, frequent items overflow in lower-level counters, so their frequencies are kept in higher-level/large counters, whereas the frequencies of infrequent items are kept in lower-level/small counters. The number of small counters at the bottom of the "tower" is significantly higher than the number of large counters at the top of the "tower", so Tower estimates the frequency of infrequent items more accurately.

III. X-Sketch Design

A. Baseline Solution

Based on the CM sketch [23], we can divide our baseline solution into two parts. For recording the estimated frequencies for the latest p windows, we construct p CM sketches. In each CM sketch, we construct d counter arrays $A_i(1 \le i \le d)$ to maintain the frequencies of items, and each array is associated with one hash function f_i. In CM sketch, when an item e is coming, it is first mapped into each array with the hash function f_i and increments the counter $A_i[f_i(e)]$ by 1. Then, we select the counter $min(A_i[f_i(e)])$ as the frequency of item e in the query process. In addition, we use a set to store the item ID of the potential simplex items in each window, and combine a hash table to record the lasting time of simplex items. The sketches and set are periodically cleared. Our insertion procedure is as follows. When an item appears in a window w, we first compute w modulo p (denoted as $w\%p$) to select the CM sketch representing the current window and increment its recorded frequency by 1. At the same time, we check the sketch representing previous windows and identify the continuity. If it is not interrupted, we insert it into the set as a potential simplex item. At the end of each window, we traverse the entire set and get the frequencies in previous p windows, and perform polynomial fitting. Finally, if it satisfies the k-simplex definition, we insert it into the hash table and record its lasting time. Although our baseline solution is able to detect simplex items, there is still much room for improvement in terms of memory overhead and accuracy.

B. Finding the Polynomial with Minimum MSE

In statistics, a **linear regression model** assumes that given the input $X_i = (x_{i0}, \cdots, x_{ik})(0 \le i \le p-1)$, the response y_i satisfies

$$y_i = \beta_0 x_{i0} + \beta_1 x_{i1} + \cdots + \beta_k x_{ik} + e_i = \sum_{j=0}^{k} \beta_j x_{ij} + e_i, \quad (1)$$

where $\beta_j(0 \le j \le k)$ are unknown parameters and $e_i(0 \le i \le p-1)$ are i.i.d. random with expectation equal to zero. If we define a $p \times (k+1)$ matrix $X = (x_{ij})(0 \le i \le p-1, 0 \le j \le k)$ and $Y = (y_0, \cdots, y_{p-1})^T$, $e = (e_0, \cdots, e_{p-1})^T$, $\beta = (\beta_0, \cdots, \beta_k)^T$, the linear regression model can be written as

$$Y = X\beta + e. \quad (2)$$

In X-Sketch, for a potential simplex item e with its frequencies in p consecutive windows $f_w, f_{w+1}, \cdots, f_{w+p-1}$, our task is to find a polynomial with minimum MSE ε.

$$\begin{aligned}(a_0, \cdots, a_k) &= \arg\min_{\beta \in \mathbb{R}^{k+1}} \frac{1}{p} \sum_{i=0}^{p-1} (f(i) - f_{w+i})^2 \\ &= \arg\min_{\beta \in \mathbb{R}^{k+1}} \frac{1}{p} ||Y - X\beta||^2.\end{aligned} \quad (3)$$

Hence $y_i = f_{w+i}$, $x_{ij} = i^j$, and

$$X = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \cdots & \vdots \\ 1 & p-1 & \cdots & (p-1)^k \end{pmatrix}. \quad (4)$$

By the normal equation, the parameter β with minimum MSE satisfies

$$X^T X \beta = X^T Y, \quad (5)$$

which can be simplified as $\beta = (X^T X)^{-1} X^T Y$. Since simplex items with different ID and different start windows share the same matrix $(X^T X)^{-1} X^T$, this matrix can be calculated and stored in advance. Finding the polynomial hence only involves matrix multiplication for one time within $O(pk^2)$ time, and computing the matrix $(X^T X)^{-1} X^T$ in advance can be done in $O(pk^2)$ time, which is time-efficient and space-efficient in practice.

C. Optimization: Threshold for Coefficient of Highest Order

Since the MSE will not increase when we increase the degree of the polynomial, an item e will surely be k-simplex if it is already $(k-1)$-simplex. For instance, a train with a constant velocity will always have a stable acceleration (equal to 0), and the velocity of a car in a parking lot cannot change sharply. To distinguish k-simplex items from the $(k-1)$-simplex ones, we propose the following method: the absolute value of the coefficient of the highest order in the polynomial $f(n) = a_0 + a_1 n + \cdots + a_k n^k$ cannot be too small, and

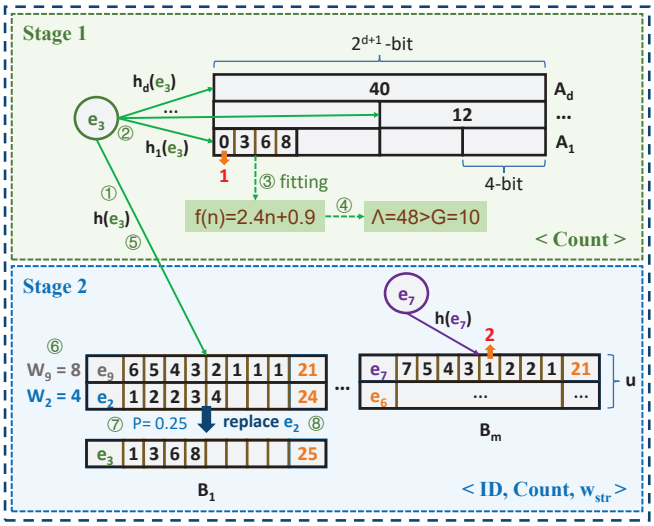


Fig. 2: Data structure and example of X-Sketch.

only items with large $|a_k|$ will be reported as k-simplex items. Specifically, we add a threshold L for a_k. In Section V, we set $L = 1$ by default. After finding out the polynomial with minimum MSE, we check whether the inequality $|a_k| \geq L$ holds. If $|a_k| < L$, then our algorithm will not report e as a k-simplex item, even if it has a small MSE. This way, the highest order of the polynomial dominates the frequencies of the item and we prevent over-fitting.

D. The X-Sketch

Overview: As shown in Figure 2, X-Sketch consists of two parts, called Stage 1 and Stage 2. Stage 1 starts with recording item frequencies. It filters out non-simplex items using the *Short-Term Filtering* technique, and then evaluates the remaining items using the *Potential* optimization to obtain a set of potential simplex items to be handled by Stage 2. Next, Stage 2 keeps and reports final simplex items, using the *Weight Election* technique.

1) Stage 1:

Data Structure: Stage 1 is a Tower [26] (Section II-B2) consisting of d arrays: $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_d$, each of which has $q_i (1 \leq i \leq d)$ counters of size 2^{i+1} and is associated with a hash function $h_i(.)$. Each counter has s sub-counters to record the item frequencies of the latest s windows, where s is a predefined value less than p.

Rationale: Instead of directly use p sub-counters to record the item frequencies of p consecutive windows, we use the shorter s sub-counters to record the item frequencies of s consecutive windows to save memory, and replace the occupation for the extra $p - s$ sub-counters with Stage 2 of less memory.

Key Technique I: Short-Term Filtering. Its key idea is to leverage the data structure of Towers to record item frequencies accurately and quickly filter out the items that do not meet the ***Preliminary Condition***, *which requires that the items satisfy more than s consecutive windows and these frequencies can be fitted with polynomials of k-degree.* Specifically, we use lower-level small counters to record non-simplex items that are easily interrupted for filtering, while higher-level large counters are used to record the relatively continuous potential simplex items. The process slides every s windows and clears the non-simplex items. We compare the performance of Tower and other state-of-the-art filtering schemes in Section V-B.

Optimization: Potential. Here, we define a `Potential` Λ for each item that has passed Short-Term Filtering to further filter out items with insufficient potential. Λ integrates the two most critical variables: ε and $|a_k|$, as follows.

$$\Lambda = \frac{|a_k|}{\varepsilon + \Delta} \tag{6}$$

where $|a_k|$ is the k^{th} coefficient, ε is the MSE of the polynomial fitting, and Δ is a very small number that prevents the denominator from being 0. Next, if $\Lambda \geq G$ (where G is a pre-set potential threshold), we estimate that it will continue to be a truly simplex item for at least p continuous windows after being sent to Stage 2. Finally, we insert it into Stage 2 as a potential simplex item.

2) Stage 2:

Data Structure: Stage 2 is a hash table with m buckets: $\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_m$. There are u cells in each bucket, and each cell contains three fields: ID, Count, and starting window w_{str}. The Count field consists of p counters, which are used to record the frequencies of the latest p windows for an item. The hash function $h(.)$ randomly maps the item to one of the buckets.

Rationale: Stage 2 not only finds the items that satisfy the k-simplex definition, but also accurately selects and keeps the simplex items with a larger number of consecutive windows.

Key Technique II: Weight Election. Its key idea is to define a `Weight` $\mathcal{W}$ for each potential simplex item entering Stage 2 with a reasonable replacement strategy, to protect/keep those simplex items with greater weights from being replaced. Here, we define $\mathcal{W}$ as the lasting time t (number of consecutive windows) of the item, which can also be calculated as the difference between the current window w and the starting window w_{str}, with the following equation:

$$\mathcal{W} = w - w_{str} = t \tag{7}$$

where w_{str} is defined as the earliest start window where the current item may start satisfying the k-simplex definition to the current window. For a potential simplex item that has passed Stage 1 and is about to be inserted into Stage 2, w_{str} is logically set to $w - s + 1$. Next, we design a replacement strategy based on Equation 7 for each potential simplex item. When the mapped bucket is full, the inserted item tries to replace the item with the smallest weight with the probability of $\mathcal{P} = \frac{1}{\mathcal{W}_{min}}$, where $\mathcal{W}_{min}$ is the smallest weight in the bucket, which means it has the largest/latest w_{str}.

3) Operations:

Insertion: The pseudo-code of the insertion operation is shown in Algorithm 1. When an item e appears in the current window w, we first map it to the bucket $\mathcal{B}[h(e)]$ in Stage 2 through the hash function $h(e)$. We show how to insert e into $\mathcal{B}[h(e)]$, which can be summarized in two cases as follows.

Case 1: If e is already recorded in $\mathcal{B}[h(e)]$, we just update the information of the cell that records e: select the counter

representing the current window by computing $w\%p$, and increment its Count field by 1.

Case 2: If e is not in $\mathcal{B}[h(e)]$, then we first try to insert e into Stage 1: we map e to counters $\mathcal{A}_1[h_1(e)]$, $\mathcal{A}_2[h_2(e)]$, $\cdots$, $\mathcal{A}_i[h_i(e)]$, $\cdots$, $\mathcal{A}_d[h_d(e)]$ by computing d hash functions. For this, we first select the sub-counters representing the current window by computing $w\%s$ among the mapped counters, and then choose one of the following two versions of the operations: (1) increment the frequencies of all these sub-counters by 1, which we call the CM version of X-Sketch (**XS-CM**); (2) increment the frequency of the sub-counter with the smallest value among them by 1, which we call the CU version of X-Sketch (**XS-CU**). Next, we read the frequencies of the latest s windows, and return if there is 0. Conversely, if the *Preliminary Condition* (Section III-D1) is satisfied, we perform polynomial fitting on the frequencies of these s windows $< f_{w-s+1}, \cdots, f_w >$. Then, we calculate the `Potential` Λ_e of e by Equation 6. If $\Lambda_e < G$, then we return. Otherwise, since e has become a potential simplex item, we set w_{str} to $w-s+1$, and insert e into Stage 2 again. There are two sub-cases:

1) If bucket $\mathcal{B}[h(e)]$ contains at least one empty cell, we insert e and its frequencies in s windows into an arbitrary empty cell.
2) If the bucket $\mathcal{B}[h(e)]$ is full, we try to replace the item with the smallest weight in the bucket with e according to $\mathcal{P} = \frac{1}{\mathcal{W}_{min}}$. If the replacement condition holds, e occupies the cell containing the item with the smallest weight. It updates its frequencies in the last s windows to the query results from Stage 1, and sets its frequencies in other windows to 0. Finally, the frequencies of e are set to $< 0, \cdots, f_{w-p+1}, \cdots, f_w >$. Otherwise, we return.

Report: At the end of each window, we only need to traverse Stage 2: for each potential simplex item, we perform polynomial fitting on it, and report if the item satisfies the k-simplex definition.

Cleaning Policy: In Stage 1, we clean all counters at the end of each window. In Stage 2, if the frequency of an item in the current window is 0, then we clear it directly. If the number of windows for the recorded item frequencies is less than p, then we return. Otherwise, we perform polynomial fitting on the frequencies of these p windows $< f_{w-p+1}, \cdots, f_w >$. Further, if the k-simplex definition is satisfied, the starting window w_{str} is unchanged and the item is reported. Otherwise, w_{str} is updated to $w-p+2$. Finally, we clear all the earliest counters to record the item frequency of the next window. The pseudo-code of the window transition procedure in Stage 2 is shown in Algorithm 2.

Example: As shown in Figure 2, we assume $w = 28$, $k = 1$, $s = 4$, $p = 8$, $\Delta = 10^{-6}$, $d = m = 3$, $u = 2$, $G = 10$, and $L = T = 1$. **(1)** When receiving item e_7, X-Sketch maps it to $\mathcal{B}_3$. Since there is a cell storing e_7, X-Sketch selects the corresponding counter by calculating $28\%8 = 4$ and increments its frequency by 1: $1 + 1 = 2$. Then, X-Sketch performs polynomial fitting on its frequencies and gets $f(n) = -0.7738n + 5.8333$. Since $|a_1| < L = 1$, we do not report it and return. **(2)** When receiving item e_3, ① X-Sketch maps it to Stage 2 and finds not present in any bucket. Then, ② X-Sketch maps it to one of the buckets in each array of Stage 1, selects the corresponding counters by calculating $28\%4 = 0$, and increments the frequency of the counter with the smallest value among them by 1: $0 + 1 = 1$. Since the s counters in this bucket of the $\mathcal{A}_3$ array are full, ③ X-Sketch performs a 1-degree polynomial fitting on its frequencies and gets $f(n) = 2.4n+0.9$, where $a_1 = 2.4$ and $\varepsilon = 0.05$. Next, ④ X-Sketch calculates $\Lambda_{e_3} = 48$, which is greater than G, then ⑤ inserts e_3 into Stage 2 again and maps it to $\mathcal{B}_1$. While $\mathcal{B}_1$ is full, ⑥ X-Sketch calculates the weights of all items in $\mathcal{B}_1$: $\mathcal{W}_{e_2} = 4$ and $\mathcal{W}_{e_9} = 8$. Finally, ⑦ X-Sketch uses e_3 to replace the smallest weight item e_2 with probability $\mathcal{P} = \frac{1}{4} = 0.25$. Assuming that the probability condition is satisfied, ⑧ then e_3 successfully evicts e_2.

Algorithm 1: Insertion Procedure

Input: an item e in window w;
map e into bucket $\mathcal{B}[h(e)]$ in Stage 2;
if *e is in Stage 2* **then**
  suppose e in the bucket is $< e, f_{w-p+1}, \cdots, f_w >$;
  update e into $< e, f_{w-p+1}, \cdots, f_w + 1 >$;
else
  for $1 \le i \le d$ **do**
    map e into counter $\mathcal{A}_i[h_i(e)]$ in Stage 1;
    $\mathcal{A}_i[h_i(e)] \leftarrow \mathcal{A}_i[h_i(e)] + 1$;
  query Stage 1, it reports $< f'_{w-s+1}, \cdots, f'_w >$;
  if $\exists i \in \{w-s+1, \cdots, w\}$, *s.t.* $f'_i = 0$ **then**
    return;
  calculate the potential of e, suppose it is Λ_e;
  if $\Lambda_e < G$ **then**
    return;
  find the item $e*$ in bucket $\mathcal{B}[h(e)]$ with smallest weight $\mathcal{W}_{min}$;
  e replaces $e*$ with probability $\mathcal{P} = \frac{1}{\mathcal{W}_{min}}$;
  if *e succeeds to replace $e*$ in bucket $\mathcal{B}[h(e)]$* **then**
    insert $< e, 0, \cdots, f'_{w-s+1}, \cdots, f'_w >$ into bucket $\mathcal{B}[h(e)]$ and set w_{str} to $w-s+1$;

E. The Effects of X-Sketch's Key Parameters

We divide the key parameters of X-Sketch into two categories: 1) parameters for problem definition; 2) parameters for algorithm design. Please see the Appendices [28] for details.

IV. Mathematical Analysis

In this section, we first show the error bound of frequency in Stage 1 and claim the no estimation error of frequency in Stage 2. Then, we use the error bound of frequency to derive the error bound of a_k and ε. Finally, we analyze the time complexity of X-Sketch.

A. Error Bound of Stage 1

Theorem 1. *Please refer to the Appendices [28] for details.*

Proof. The proof of this theorem is also provided in [28]. □

Algorithm 2: Stage 2 Transition Procedure

for $1 \le i \le m$ **do**
  for $1 \le j \le u$ **do**
    suppose the item in the cell is $< e, f_{w-p+1}, \cdots, f_w >$;
    if $f_w = 0$ **then**
      evict e from the bucket;
      continue;
    perform polynomial fitting on e;
    if $a_k \ge L$ ***and*** $\varepsilon \le T$ **then**
      report $(e, w-p+1)$ as a simplex item;
    else
      $\mathcal{B}[i][j].w_{str} \leftarrow w-p+2$;
    update e to $< e, f_{w-p+2}, \cdots, f_w, 0 >$;

B. Proof of no Estimation Error in Stage 2

Theorem 2. *For an arbitrary item e, assume e enters stage 2 in window w_1, and is evicted from stage 2 in window w_2. Let f_w be its real frequency in window w, and $\hat{f}_w$ be its frequency in window w reported by stage 2, then $f_w = \hat{f}_w$ for $w_1 < w < w_2$.*

Proof. For $w_1 < w < w_2$, since e is inserted into stage 2 in window w_1 and is evicted from stage 2 in window w_2, item e stays in stage 2 throughout window w, As a result, if item e comes for f_w times in window w, the counter of window w in stage 2 will increment for f_w times. Hence $f_w = \hat{f}_w$. □

C. Error Bound of a_k and ε

In this part, we give error bounds for the fitting polynomial w.r.t. the error of frequency in Stage 1 and Stage 2. For an arbitrary item e and a start window w, we use $Y = (f_w, \cdots, f_{w+p-1})$ to denote its real frequency, and $\hat{Y} = (\hat{f}_w, \cdots, \hat{f}_{w+p-1})$ to denote its frequency reported by X-Sketch. We use $\beta = (a_0, \cdots, a_k)$ to denote its real coefficients of the polynomial with minimum MSE, $\hat{\beta} = (\hat{a_0}, \cdots, \hat{a_k})$ to denote the coefficients reported by X-Sketch, and ε be the real MSE, $\hat{\varepsilon}$ be the MSE reported by X-Sketch.

For an n-dimensional vector x, we use $||x||_n$ to denote its norm, *i.e.*

$$||x||_n = ||(x_1, \cdots, x_n)||_n = \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{2}}.$$

For a $m \times n$ matrix A, we use $||A||$ to denote its norm, *i.e.*

$$||A|| = \sup_{x \in \mathbb{R}^n/\{0\}} \frac{||Ax||_m}{||x||_n}.$$

Theorem 3. *The estimation error of a_k is bounded.*

$$|a_k - \hat{a}_k| \le ||(X^T X)^{-1} X^T|| \cdot ||Y - \hat{Y}||_p. \tag{8}$$

Proof. Since $\beta = (X^T X)^{-1} X^T Y$ and $\hat{\beta} = (X^T X)^{-1} X^T \hat{Y}$, then

$$\begin{aligned} ||\beta - \hat{\beta}||_{k+1} &= ||(X^T X)^{-1} X^T (Y - \hat{Y})||_{k+1} \\ &\le ||(X^T X)^{-1} X^T|| \cdot ||Y - \hat{Y}||_p. \end{aligned} \tag{9}$$

Hence, Equation 8 holds since

$$||\beta - \hat{\beta}||_{k+1} = \left(\sum_{i=0}^{k} (a_i - \hat{a}_i)^2\right)^{\frac{1}{2}} \ge |a_k - \hat{a}_k|. \tag{10}$$

□

Theorem 4. *The estimation error of ε is bounded.*

$$|\varepsilon - \hat{\varepsilon}| \le \frac{2}{p} \max\{||Y||_p, ||\hat{Y}||_p\} \cdot ||A|| \cdot ||Y - \hat{Y}||_p, \tag{11}$$

where matrix A is defined as

$$A = I_p - X(X^T X)^{-1} X^T$$

and I_p is the identity matrix of size p.

Proof. Since β and $\hat{\beta}$ satisfy

$$X^T X \beta = X^T Y, X^T X \hat{\beta} = X^T \hat{Y}$$

and $X^T X$ is invertible, then

$$\begin{aligned} \epsilon &= \frac{1}{p}(Y - X\beta)^T (Y - X\beta) \\ &= \frac{1}{p}(Y^T Y - 2\beta^T X^T Y + \beta^T X^T X \beta) \\ &= \frac{1}{p}(Y^T Y - \beta^T X^T Y) \\ &= \frac{1}{p}(Y^T Y - Y^T X (X^T X)^{-1} X^T Y) = \frac{1}{p} Y^T A Y. \end{aligned} \tag{12}$$

Similarly, $\hat{\varepsilon} = \frac{1}{p} \hat{Y}^T A \hat{Y}$, and

$$\begin{aligned} |\varepsilon - \hat{\varepsilon}| &= \frac{1}{p} |(Y + \hat{Y})^T A (Y - \hat{Y})| \\ &\le \frac{1}{p} ||Y + \hat{Y}||_p \cdot ||A|| \cdot ||Y - \hat{Y}||_p \\ &\le \frac{2}{p} \max\{||Y||_p, ||\hat{Y}||_p\} \cdot ||A|| \cdot ||Y - \hat{Y}||_p. \end{aligned} \tag{13}$$

Hence, Equation 11 holds. □

The two theorems above show that once we control the estimation error of Y, then the estimation error of a_k and ε will also be controlled.

D. Time Cost of X-Sketch

Theorem 5. *For each item in the data stream, the time cost to handle it is $O(1)$.*

Proof. For each item e in the data stream, we first use hash function $h(e)$ to map it into Stage 2. If e is in bucket $\mathcal{B}[h(e)]$, we update its frequency and return, the time cost of which is $O(u)$. Otherwise, we use d hash functions $h_i(e)$ to map it into Stage 1 and increment the counter, with a time complexity of $O(d)$. Then, we query its frequency in the last s windows and start polynomial fitting. The query operation costs $O(sd)$ time and polynomial fitting through linear regression costs $O(sk^2)$ time. Also, we check the starting window of u items from bucket $\mathcal{B}[h(e)]$, which costs $O(u)$ time. Finally, we try to insert e into Stage 2 according to its weight, which costs $O(p)$ time. At the end of every window, we clear Stage 1 and perform linear regression to report every potential k-simplex item in Stage 2, for an amortized time cost of $O(pk^2)$. In conclusion, the time cost for every item only depends on these parameters. In data stream models, p, k, s, u, d are all very small (each of them is smaller than 10 in practice) to achieve high throughput, so the time cost to handle each item e can be viewed as $O(1)$. □

V. Experimental Results

In this section, we provide experimental results with X-Sketch. First, we describe the experimental setup in Section V-A. Then, we show how parameter settings affect X-Sketch's performance in Section V-B. Finally, we evaluate the performance of X-Sketch on different datasets in Section V-C.

A. Experimental Setup

Implementation: We implement X-Sketch and all other algorithms in C++, and use 32-bit Bob Hash (obtained from the open-source website [47]) with different initial seeds.

Computation Platform: We conduct all the experiments on a server with one 18-core processor (36 threads, Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz) and 128 GB DRAM memory. The processor has 64KB L1 cache, 1MB L2 cache for each core, and 24.75MB L3 cache shared by all cores.

Metrics:

1) Precision Rate (PR): The ratio of true positive instances to all reported instances.

2) Recall Rate (RR): The ratio of true positive instances to all actual instances.

3) F1 Score: $\frac{2*RR*PR}{RR+PR}$.

4) Average Relative Error (ARE): Let $\hat{t_1}, \hat{t_2}, \ldots, \hat{t_z}$ be the estimated lasting time of the reported items, and let $t_1, t_2, \ldots, t_z$ be the true lasting time of the reported items. ARE is defined as $\frac{1}{z} \cdot \sum_{j=1}^{z} \frac{|t_j - \hat{t_j}|}{t_j}$.

5) Throughput: We use Million of operations (insertions) per second (Mops) to measure the throughput.

Datasets: We use three real-world datasets and one Synthetic Dataset. For each dataset above, we divide it into 3000 windows, each containing 10000 items.

1) IP Trace Dataset. The IP Trace Dataset is streams of anonymized IP traces collected in 2016 by CAIDA [25].

2) MAWI Dataset: The MAWI Dataset is a set of anonymized traffic traces collected from trans-Pacific backbone link by MAWI Working Group [48].

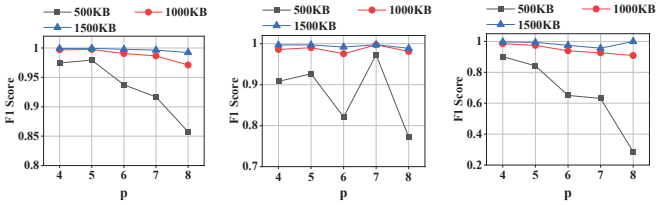
3) Data Center Dataset. The Data center dataset [49] contains traces collected from the data centers studied in [50].

4) Synthetic Dataset. We generate the Synthetic Datasets that follows the Zipf [51] distribution using Web Polygraph [52], an open-source performance testing tool. Here, we use the dataset with skewness of 1.5.

B. Experiments on Parameter Settings

In this section, we measure the effects of some key parameters of X-Sketch (XS-CM) on $k = 0, 1, 2$, namely, number of consecutive windows p in the definition, the number of cells u per bucket in Stage 2, the ratio r of the memory size of Stage 1 to the memory size of the whole X-Sketch, the number s of the latest windows recorded in Stage 1, the threshold G for Potential Λ, the threshold T for MSE ε, and the data structure used by Stage 1 mentioned in Section III-D1. We use the IP Trace Dataset in these experiments, and F1 Score to evaluate the effects. *We default* $d = 3$.

Effects of p (Figure 3(a)-3(c)): *The experimental results show that the optimal value for p is from 4 to 7.* In this experiment, we vary p from 4 to 8 with a step size of 1. The

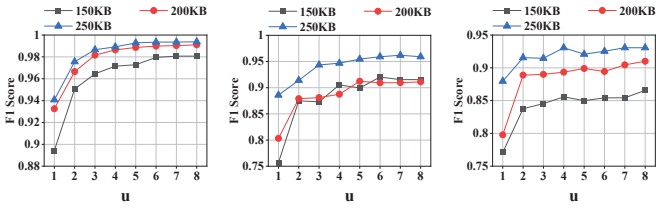


(a) $k = 0$ (b) $k = 1$ (c) $k = 2$

Fig. 3: Effects of the parameter: p

results show that the F1 Score mainly decreases as p increases, especially when the memory size is 500KB. However, the F1 Score peaks when $p = 7$ for $k = 1$ for this memory size. When the memory size is 1000KB/1500KB, the weakening of F1 Score becomes smaller with the increase of p. Considering that a larger p implies obtaining more ideal simplex items, we set $p = 7$ for $k = 0, 1, 2$. The reason for the trend in Figure 3 is that increasing p requires more memory overhead.

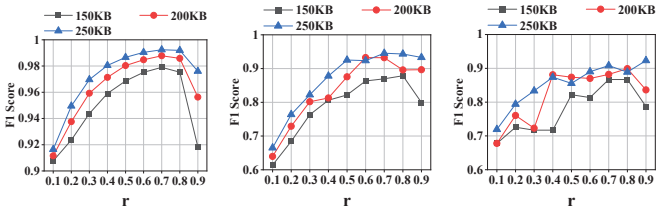
Effects of u (Figure 4(a)-4(c)): *The experimental results show that the optimal value for u is from 4 to 8.* In this experiment, we vary u from 1 to 8 with a step size of 1. The results show that when u is less than 3, the F1 Score increases as u increases, but when u exceeds 4, the F1 Score tends to be stable. Since larger u means smaller throughput, we set u to 4 for $k = 0, 1, 2$ in our experiments. The reason for the trend in Figure 4 is that as u increases, the least weighted item in the bucket is replaced more accurately based on the replacement strategy in Section III-D2, which eventually finds the simplex items more accurately.



(a) $k = 0$ (b) $k = 1$ (c) $k = 2$

Fig. 4: Effects of the parameter: u

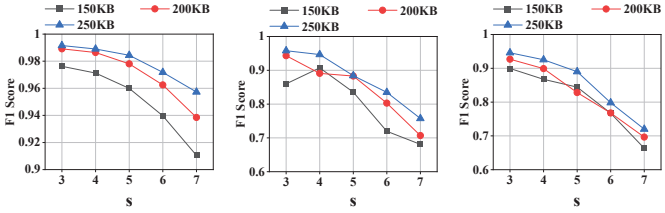
Effects of r (Figure 5(a)-5(c)): *The experimental results show that the optimal value for r is from 0.7 to 0.8.* In this experiment, we vary r from 0.1 to 0.9 with a step size of 0.1. We find that 0.7 or 0.8 is always the optimal value for r in 3 different memory cases. Thus, we set r to 0.8 for $k = 0, 1, 2$ in our experiments. The reason for the trend in Figure 5 is that Stage 1 filters non-simplex items more accurately as r increases. However, if r is too large, there will not be enough memory in Stage 2 to keep the final simplex items, resulting in a decrease in accuracy.



(a) $k = 0$ (b) $k = 1$ (c) $k = 2$

Fig. 5: Effects of the parameter: r

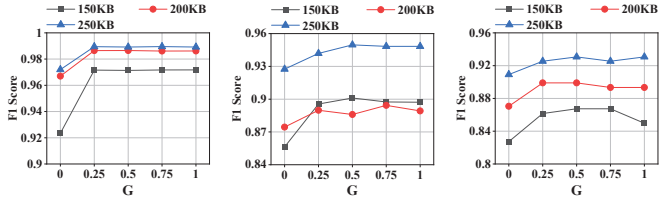
Effects of s (Figure 6(a)-6(c)): *The experimental results show that the optimal value for s is from 3 to 4.* In this experiment, we vary s from 3 to 7 with a step size of 1. The results show that the F1 Score mostly decreases as s increases. However, when the memory size is 150KB, the F1 Score peaks when $s = 4$ for $k = 1$. Since the F1 Scores for $s = 3$ and $s = 4$ are close, we set s to 4 for $k = 0, 1, 2$ to to ensure that Stage 1 filters out more items. The reason for the trend in Figure 6 is that the filtering in Stage 1 is more accurate but at the same time requires more memory overhead as s increases.



(a) $k = 0$ (b) $k = 1$ (c) $k = 2$

Fig. 6: Effects of the parameter: s

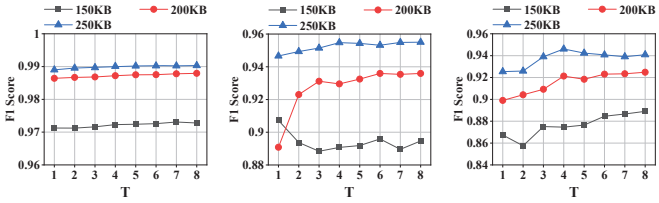
Effects of G (Figure 7(a)-7(c)): *The experimental results show that the optimal value for G is from 0.5 to 1.* In this experiment, we vary G from 0 to 1 with a step size of 0.25. We find that when G is less than 0.25, the F1 Score obviously increases as G increases, but when G exceeds 0.5, the F1 Score tends to be stable. Thus, we set G to 0.5 for $k = 0, 1, 2$. The reason for the trend in Figure 7 is that true simplex items may be missed when G is large, while too many non-simplex items may be passed to Stage 2 when G is small. However, for a large range of G, our replacement mechanism makes it work to detect true simplex items from items entering Stage 2.



(a) $k = 0$ (b) $k = 1$ (c) $k = 2$

Fig. 7: Effects of the parameter: G

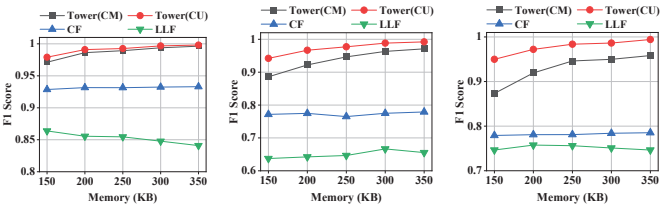
Effects of T (Figure 8(a)-8(c)): *The experimental results show that the optimal value of T is different for $k = 0, 1, 2$.* In this experiment, we vary T from 1 to 8 with a step size of 1. 1) The results show that the value of T has no significant effect on the F1 Score for $k = 0$. For simplicity, we set T to 1 for $k = 0$. 2) When the memory size is 200/250KB, the F1 Score increases as T increases as a whole for $k = 1$. However, when the memory is 150KB, the F1 Score decreases first and then fluctuates as T increases. For simplicity, we set T to 2 for $k = 1$. 3) When the memory size is 200/250KB, we find that the F1 Score peaks when $T = 4$ for $k = 2$. When the memory size is 150KB, the F1 Score increases slowly when T is greater than 4. Thus, we set T to 4 for $k = 2$. The reason for the trend in Figure 8 is that a larger T means greater tolerance for higher order polynomial fitting errors which contributes to a slight increase in accuracy.



(a) $k = 0$ (b) $k = 1$ (c) $k = 2$

Fig. 8: Effects of the parameter: T

Effects of Stage 1 data structure (Figure 9(a)-9(c)): *The experimental results show that the best data structure for Stage 1 is TowerSketch.* We choose Cold Filter (CF) [40], LogLog Filter (LLF) [41] and TowerSketch (CM and CU versions) [26] currently used for comparison. The results show that TowerSketch always outperforms the two state-of-the-art schemes in terms of filtering.



(a) $k = 0$ (b) $k = 1$ (c) $k = 2$

Fig. 9: Effects of Stage 1 data structure.

C. Experiments on Finding K-Simplex Items

In this section, we set $k = 0, 1, 2$, and compare the performance of X-Sketch (CM and CU versions, written as XS-CM and XS-CU) with the baseline solution on three real-world datasets and one Synthetic Dataset in the metrics below. *We default $p = 7$ and $d = 3$.*

1) Precision Rate (PR):

(1) k=0 (Figure 10(a)-10(d)): XS-CM and XS-CU achieve 75.8% and 76.0% higher PR than the baseline solution on average, respectively; **(2) k=1 (Figure 15(a)-15(d)):** XS-CM and XS-CU achieve 60.8% and 64.3% higher PR than the baseline solution on average, respectively; **(3) k=2 (Figure 20(a)-20(d)):** XS-CM and XS-CU achieve 46.0% and 49.0% higher PR than the baseline solution on average, respectively.

2) Recall Rate (RR):

(1) k=0 (Figure 11(a)-11(d)): XS-CM and XS-CU achieve 42.0% and 44.0% higher RR than the baseline solution on average, respectively; **(2) k=1 (Figure 16(a)-16(d)):** XS-CM and XS-CU achieve 43.3% and 45.2% higher RR than the baseline solution on average, respectively; **(3) k=2 (Figure 21(a)-21(d)):** XS-CM and XS-CU achieve 27.5% and 30.0% higher RR than the baseline solution on average, respectively.

3) F1 Score:

(1) k=0 (Figure 12(a)-12(d)): XS-CM and XS-CU achieve 67.5% and 68.6% higher F1 Score than the baseline solution on average, respectively; **(2) k=1 (Figure 17(a)-17(d)):** XS-CM and XS-CU achieve 55.2% and 57.9% higher F1 Score than the baseline solution on average, respectively; **(3) k=2 (Figure 22(a)-22(d)):** XS-CM and XS-CU achieve 39.5% and 42.2% higher F1 Score than the baseline solution on average, respectively.

4) Average Relative Error (ARE):

(1) k=0 (Figure 13(a)-13(d)): XS-CM and XS-CU achieve 371.5 and 442.0 times lower ARE than the baseline solution on average, respectively; **(2) k=1 (Figure 18(a)-18(d)):** XS-CM and XS-CU achieve 30.6 and 140.0 times lower ARE than the baseline solution on average, respectively; **(3) k=2 (Figure 23(a)-23(d)):** On IP Trace and MAWI datasets, the ARE of XS-CM and XS-CU is around 26.6 and 72.0 times lower than the one of the baseline solution on average, respectively[2].

5) Throughput:

(1) k=0 (Figure 14(a)-14(d)): XS-CM and XS-CU achieve 1.63 and 1.59 times higher throughput than the baseline solution on average, respectively; **(2) k=1 (Figure 19(a)-19(d)):** XS-CM and XS-CU achieve 1.79 and 1.75 times higher throughput than the baseline solution on average, respectively; **(3) k=2 (Figure 24(a)-24(d)):** The throughput of XS-CM and XS-CU is around 1.81 and 1.77 times lower than the one of the baseline solution on average, respectively.

6) Conclusion and Discussion on Higher K:

The higher the k, the fewer simplex items are obtained that satisfy the definition, provided that the other parameters do not change. As a result, the filtering effect of our key techniques, especially in Stage 1, becomes less evident and the advantage of accuracy with the baseline solution diminishes. We also provide experimental results for $k = 3$ in the Appendices [28].

VI. X-Sketch for ML

Machine learning (ML) models are often used to predict the frequency of a certain item. However, to the best of our knowledge, these models require massive training datasets and loops of training epochs to achieve high performance, which is challenging in real-time data stream processing with high speed and large scale. Also, not all items in reality follow a strict and predictable pattern. Simply predicting the frequency of all items in the datasets is inefficient, and most of these predictions are even false or inaccurate. Therefore, X-Sketch can be applied to "accelerate" ML algorithms: We first filter possible simplex items using X-Sketch and predict their frequency by the fitting polynomial. For other not-simplex items, ML can predict their frequency in the coming window if necessary. In this part, we compare the following three schemes on the accuracy and running time of prediction.

1) X-Sketch: We use X-Sketch to find all simplex items (e, w). If e is reported as a simplex item in window $w, \cdots, w+p-1$, we predict its frequency in window $w + p$.

2) Linear Regression Model: We use linear regressing model to predict all items in data streams whether an item is simplex or not.

3) Time Series Model: We use Autoregressive Integrated Moving Average Model (ARIMA) [18] to check the autocorrelation of every simplex item and predict their frequency.

A. Experimental Setup

Implementation: We have implemented linear regression model and time series model in Python, and X-Sketch in both C++ and Python. We first run X-Sketch to get all k-simplex items. Then, we test the performance of three algorithms on these simplex items. We allocate 300KB memory for X-Sketch (C++/Python), while others use as large memory as they can.

Metrics: *1) Accuracy:* The ratio of the number of accurate predictions to the number of predictions. For every item, if the predicted result does not deviate from the ground truth too much, we classify this prediction as accurate. *2) Running Time:* The total running time of finishing all predictions.

Datasets: We use the four datasets mentioned in Section V-A (only the results of IP Trace are shown, see the Appendices [28] for others) plus a synthetic dataset called Transactional Dataset and generated by IBM Almaden Quest research group [53]. For each dataset, we use the first 300000 items and divide them into 30 windows, each containing 10000 items.

TABLE II: Experiments on the IP Trace Dataset.

	Model	Accuracy (%)	Running Time (s)
$k = 0$	X-Sketch (C++ / py)	99.97	0.05 / 0.248
	Linear Regression	99.34	26.5
	Time Series	99.62	1158
$k = 1$	X-Sketch (C++ / py)	92.75	0.04 / 0.244
	Linear Regression	86.31	26.4
	Time Series	96.23	53.8
$k = 2$	X-Sketch (C++ / py)	92.95	0.04 / 0.246
	Linear Regression	93.60	26.8
	Time Series	97.90	433

TABLE III: Experiments on the Transactional Dataset.

	Model	Accuracy (%)	Running Time (s)
$k = 0$	X-Sketch (C++ / py)	98.49	0.014 / 0.225
	Linear Regression	98.38	2.14
	Time Series	98.47	71
$k = 1$	X-Sketch (C++ / py)	90.71	0.013 / 0.222
	Linear Regression	91.67	2.11
	Time Series	93.67	37.6
$k = 2$	X-Sketch (C++ / py)	85.31	0.015 / 0.228
	Linear Regression	85.67	2.13
	Time Series	95.36	71.8

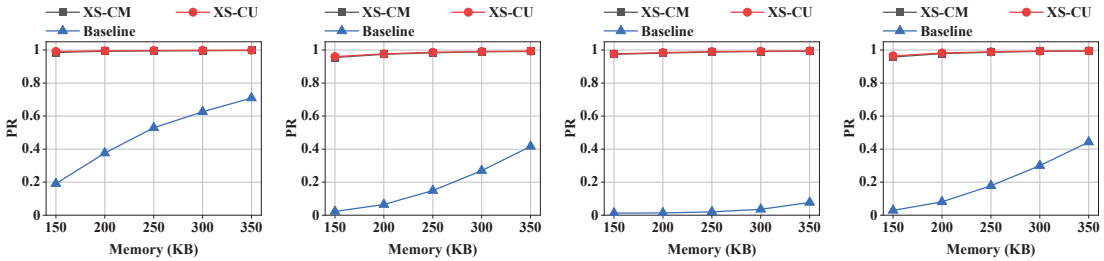
B. Experiments on Different Datasets

Our experiments show that X-Sketch can improve the running time of predictions by at least a factor of 100/9× (i.e., C++/Python) and still maintain a high level of accuracy.

Results on the IP Trace Dataset. X-Sketch's running times are 530/106.9×, 660/108.2×, 670/108.9× and 23160/4669.4×, 1345/220.5×, 10825/1760.2× faster than the linear regression and time series for $k = 0, 1, 2$, respectively. The accuracy of X-Sketch all exceed 90% within 0.05/0.25 ($i.e.$, C++/Python) seconds, while linear regression model takes nearly 30 seconds to achieve similar accuracy. Although time series model forecast these items more accurately, it usually takes several minutes to get a satisfactory result.

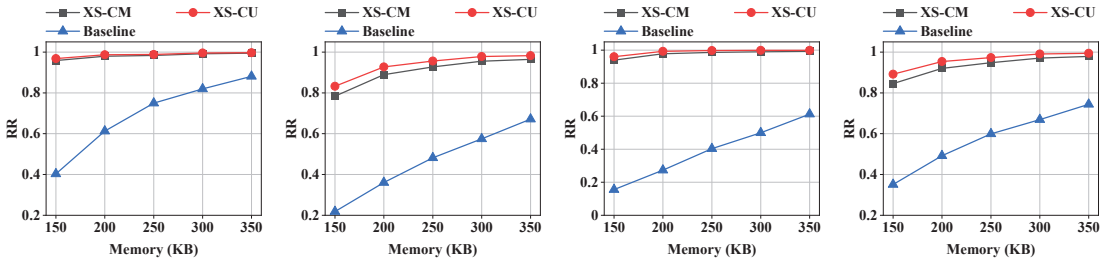
Results on the Transactional Dataset. X-Sketch's running times are 152.9/9.5×, 162.3/9.5×, 142/9.3× and 5071.4/315.6×, 2892.3/169.4×, 4786.7/314.9× faster than the linear regression and time series for $k = 0, 1, 2$, respectively. It achieves 85% accuracy in just 0.02/0.22 seconds, while the linear regression takes more than 2 seconds to achieve the same performance. The running time of time series model largely depends on the number of items we want to predict, and its throughput is always inferior to X-Sketch for $k = 0, 1, 2$.

[2]For $k = 2$, the AREs of XS-CM on the Synthetic are all 0, and the AREs of XS-CU on both the Synthetic and Data Center datasets are all 0.



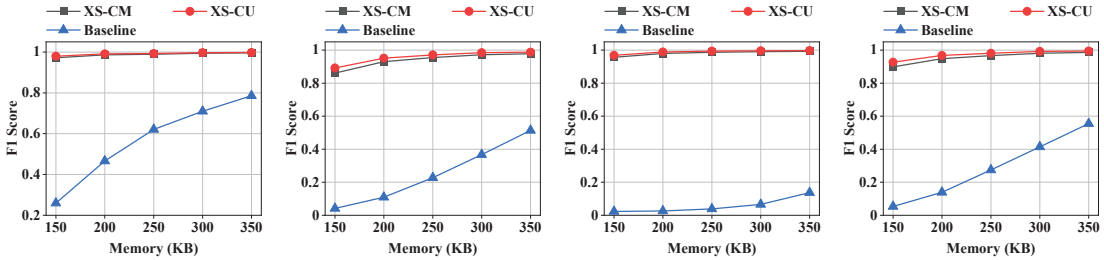
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 10: Precision Rate (PR) on finding 0-simplex items.



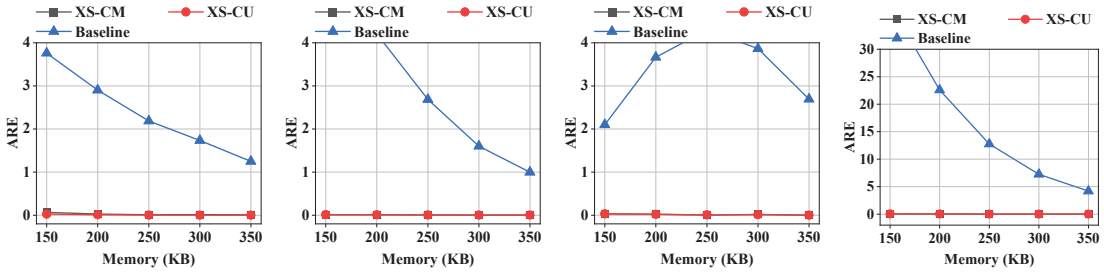
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 11: Recall Rate (RR) on finding 0-simplex items.



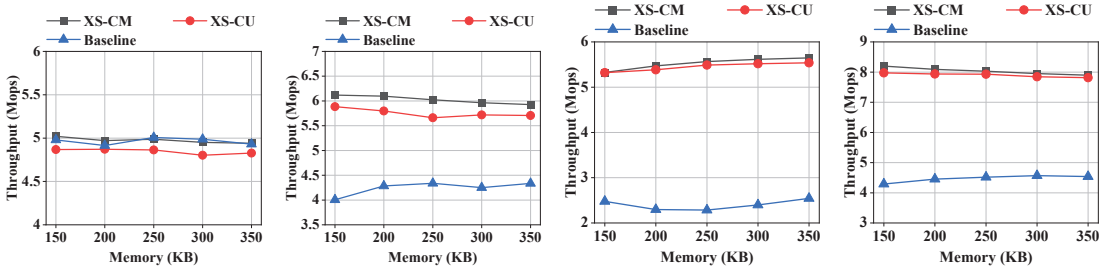
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 12: F1 Score on finding 0-simplex items.



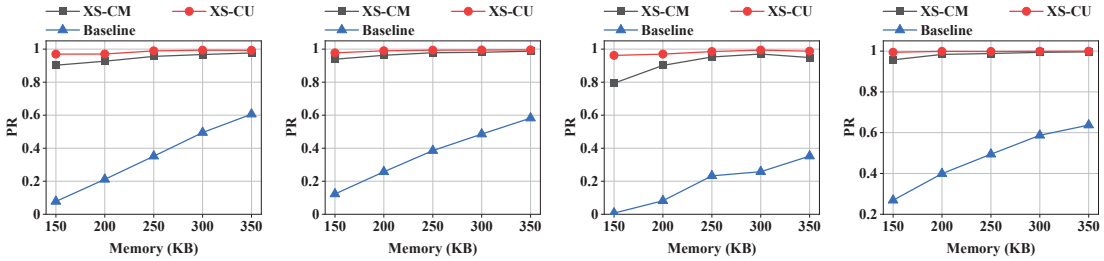
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 13: ARE on finding 0-simplex items.



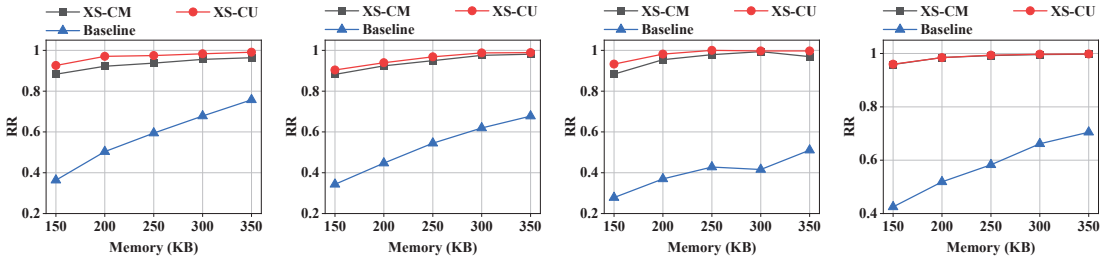
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 14: Throughput on finding 0-simplex items.



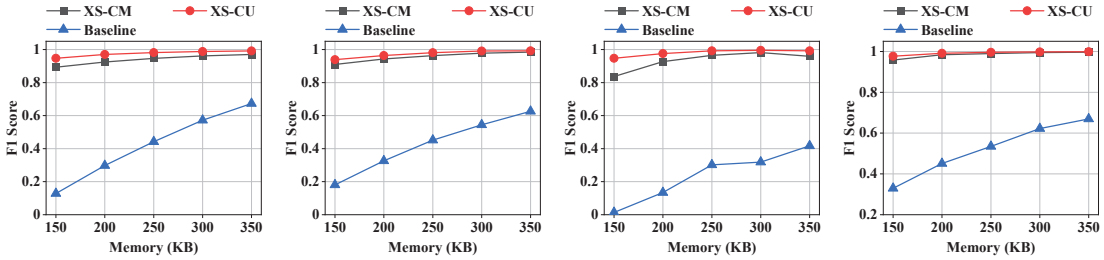
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 15: Precision Rate (PR) on finding 1-simplex items.



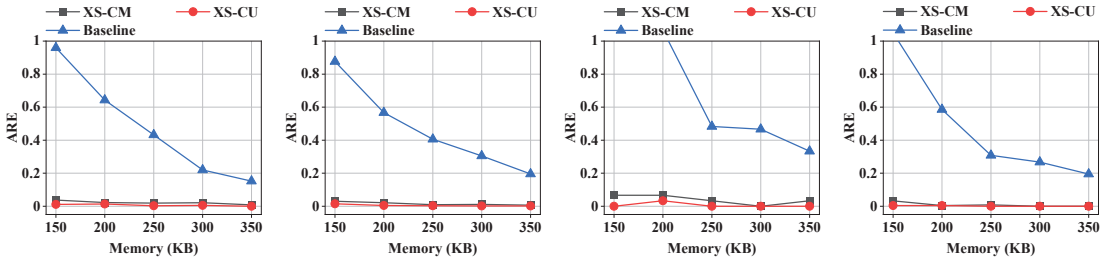
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 16: Recall Rate (RR) on finding 1-simplex items.



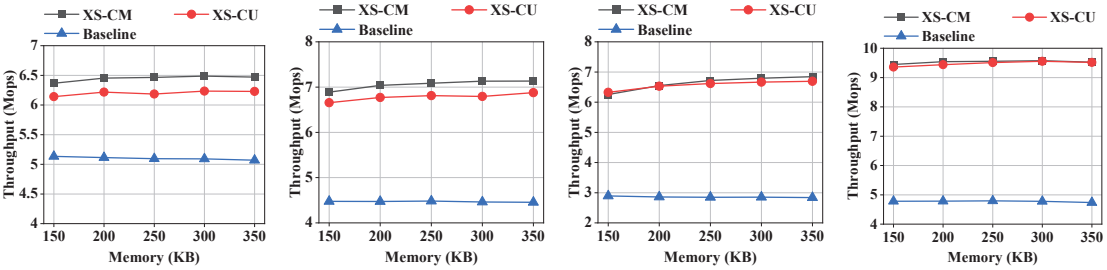
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 17: F1 Score on finding 1-simplex items.



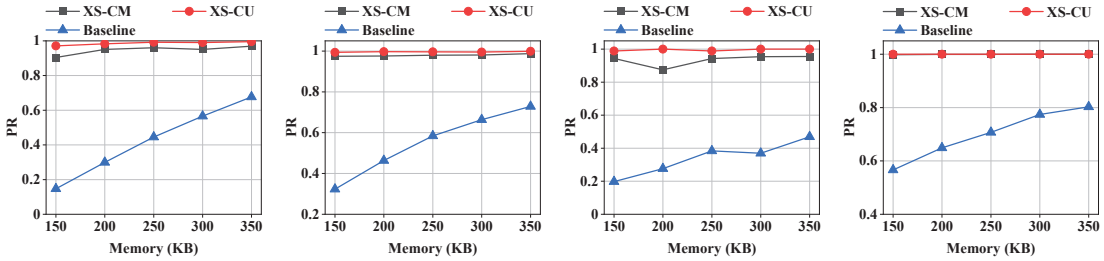
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 18: ARE on finding 1-simplex items.



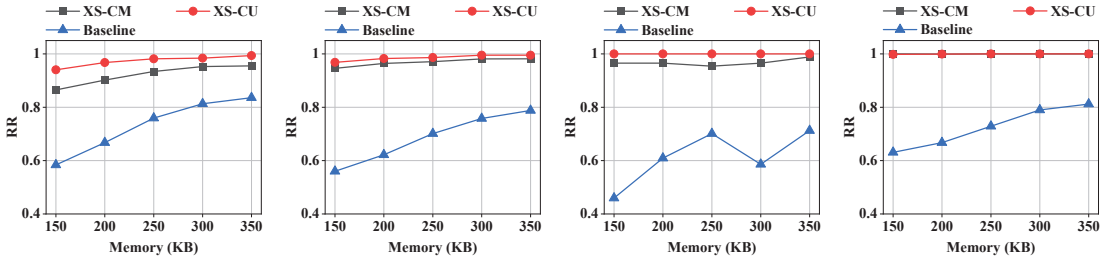
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 19: Throughput on finding 1-simplex items.



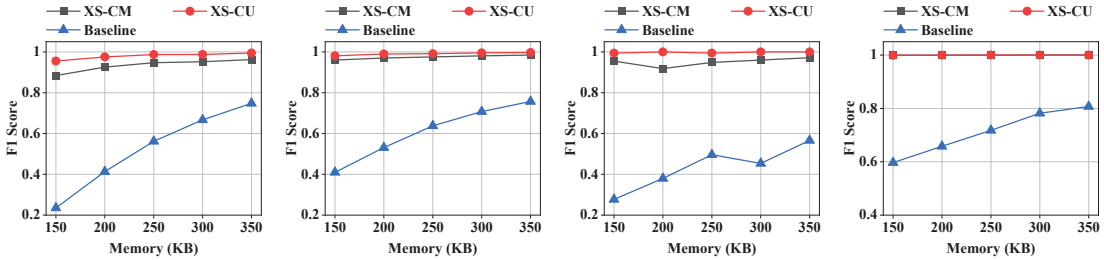
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 20: Precision Rate (PR) on finding 2-simplex items.



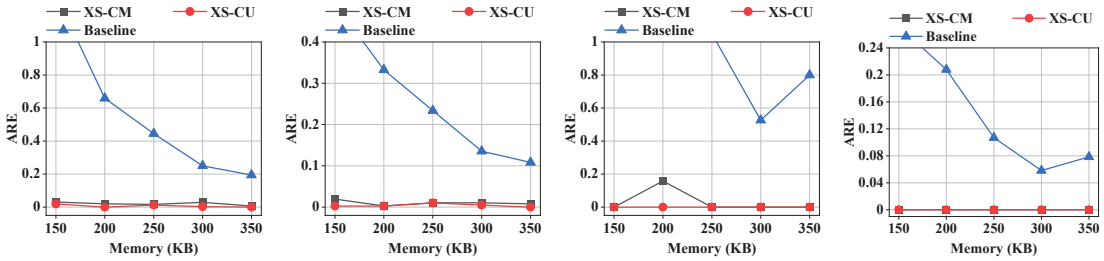
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 21: Recall Rate (RR) on finding 2-simplex items.



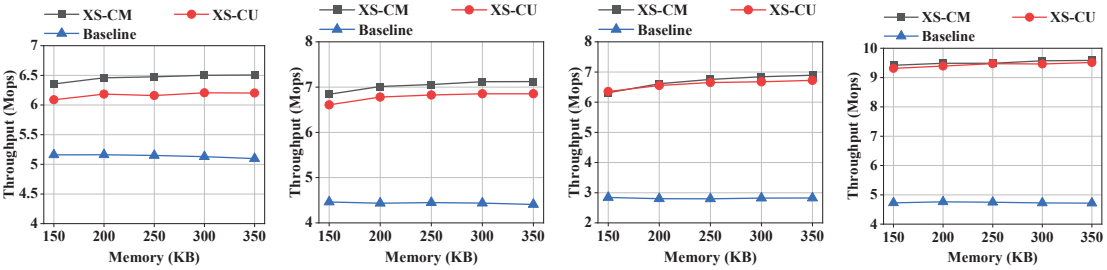
(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 22: F1 Score on finding 2-simplex items.



(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 23: ARE on finding 2-simplex items.



(a) IP Trace (b) MAWI (c) Data Center (d) Synthetic

Fig. 24: Throughput on finding 2-simplex items.

VII. Conclusion

In this paper, we define a new pattern in data streams: k-simplex items, whose frequencies in several consecutive windows can be fitted by a polynomial of k-degree ($k = 0, 1, 2$). We proposes a novel sketch algorithm called X-Sketch for finding k-degree simplex items in real time, which is memory-efficient and accurate. Our experimental results show that the F1 Score of X-Sketch is on average 68.6%, 57.9%, and 42.2% higher than the baseline solution for $k = 0, 1, 2$, respectively. In addition, our case-specific experiments validate that it can also be applied to reduce the running time of the linear regression and time series models mostly by more than 100×.

Acknowledgment

We would like to thank Yiyao Zhang for participating in the experiments of implementing X-Sketch in Python in Section VI. We would like to thank the anonymous reviewers for their thoughtful feedback. This work is supported by Key-Area Research and Development Program of Guangdong Province 2020B0101390001, and National Natural Science Foundation of China (NSFC) (No. U20A20179, 61832001).

References

[1] G. Liu, H. Lu, W. Lou, and J. X. Yu, "On computing, storing and querying frequent patterns," in *Proc. KDD*, 2003, pp. 607–612.

[2] Y. Tong, L. Chen, Y. Cheng, and P. S. Yu, "Mining frequent itemsets over uncertain databases," *Proc. VLDB Endow.*, vol. 5, no. 11, pp. 1650–1661, 2012.

[3] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, and K. Ren, "Heavy hitter estimation over set-valued data with local differential privacy," in *Proc. CCS*, 2016, pp. 192–203.

[4] A. Bhattacharyya, P. Dey, and D. P. Woodruff, "An optimal algorithm for ℓ_1-heavy hitters in insertion streams and related problems," *ACM Transactions on Algorithms (TALG)*, vol. 15, no. 1, pp. 1–27, 2018.

[5] K. Li and G. Li, "Approximate query processing: What is new and where to go? a survey on approximate query processing," *Data Science and Engineering*, vol. 3, pp. 379–397, 2018.

[6] Z. Fan, Z. Hu, Y. Wu, J. Guo, W. Liu, T. Yang, H. Wang, Y. Xu, S. Uhlig, and Y. Tu, "Pisketch: finding persistent and infrequent flows," in *Proc. FFSPIN*, 2022, pp. 8–14.

[7] Y. Zhang, X. Lin, J. Xu, F. Korn, and W. Wang, "Space-efficient relative error order sketch over data streams," in *Proc. ICDE*, 2006, pp. 51–51.

[8] A. Gkoulalas-Divanis, D. Vatsalan, D. Karapiperis, and M. Kantarcioglu, "Modern privacy-preserving record linkage techniques: An overview," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4966–4987, 2021.

[9] B. Shi, Z. Zhao, Y. Peng, F. Li, and J. M. Phillips, "At-the-time and back-in-time persistent sketches," in *Proc. SIGMOD*, 2021, pp. 1623–1636.

[10] Z. Fan, Y. Zhang, T. Yang, M. Yan, G. Wen, Y. Wu, H. Li, and B. Cui, "Periodicsketch: Finding periodic items in data streams," in *Proc. ICDE*, 2022, pp. 96–109.

[11] P. Pandey, A. Conway, J. Durie, M. A. Bender, M. Farach-Colton, and R. Johnson, "Vector quotient filters: Overcoming the time/space trade-off in filter design," in *Proc. SIGMOD*, 2021, pp. 1386–1399.

[12] Z. Shi, A. Jain, K. Swersky, M. Hashemi, P. Ranganathan, and C. Lin, "A hierarchical neural model of data prefetching," in *Proc. ASPLOS*, 2021, pp. 861–873.

[13] C. Ortega, V. Garcia, M. Moreto, M. Casas, and R. Rusitoru, "Data prefetching on in-order processors," in *Proc. HPCS*, 2018, pp. 322–329.

[14] H. Shi, Y. Cui, X. Wang, Y. Hu, M. Dai, F. Wang, and K. Zheng, "Stms: Improving mptcp throughput under heterogeneous networks," in *Proc. USENIX ATC*, 2018, pp. 719–730.

[15] B. Liu, L. Zhang, F. Wang, M. Liu, Y. Mao, L. Zhao, T. Sun, and X. Xin, "Adaptive dynamic wavelength and bandwidth allocation algorithm based on error-back-propagation neural network prediction," *Optics Communications*, vol. 437, pp. 276–284, 2019.

[16] J. Cao, Z. Ma, J. Xie, X. Zhu, F. Dong, and B. Liu, "Towards tenant demand-aware bandwidth allocation strategy in cloud datacenter," *Future Generation Computer Systems*, vol. 105, pp. 904–915, 2020.

[17] G. A. Seber and A. J. Lee, *Linear regression analysis*. John Wiley & Sons, 2012.

[18] G. Box and G. Jenkins, "Time series analysis: Forecasting and control," *Holden-Day San Francisco, California*, 1970.

[19] D. S. Yeung, S. Jin, and X. Wang, "Covariance-matrix modeling and detecting various flooding attacks," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 37, no. 2, pp. 157–169, 2007.

[20] Z. Tan, A. Jamdagni, X. He, P. Nanda, and R. P. Liu, "A system for denial-of-service attack detection based on multivariate correlation analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 447–456, 2013.

[21] Y. Chen, X. Ma, and X. Wu, "Ddos detection algorithm based on preprocessing network traffic predicted method and chaos theory," *IEEE Communications Letters*, vol. 17, no. 5, pp. 1052–1054, 2013.

[22] X. Cao, J. Chen, Y. Cheng, X. S. Shen, and Y. Sun, "An analytical mac model for ieee 802.15.4 enabled wireless networks with periodic traffic," *IEEE Transactions on Wireless Communications*, vol. 14, no. 10, pp. 5261–5273, 2015.

[23] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[24] Y. Wu, Z. Liu, X. Yu, J. Gui, H. Gan, Y. Han, T. Li, O. Rottenstreich, and T. Yang, "Mapembed: Perfect hashing with high load factor and fast update," in *Proc. KDD*, 2021, pp. 1863–1872.

[25] "The CAIDA Anonymized Internet Traces." [Online]. Available: http://www.caida.org/data/overview/

[26] K. Yang, Y. Li, Z. Liu, T. Yang, Y. Zhou, J. He, T. Zhao, Z. Jia, Y. Yang *et al.*, "Sketchint: Empowering int with towersketch for per-flow per-switch measurement," in *Proc. ICNP*, 2021, pp. 1–12.

[27] "The source codes related to X-Sketch." [Online]. Available: https://github.com/SimpleX-Sketch/X-Sketch

[28] "The appendices related to X-Sketch." [Online]. Available: https://github.com/SimpleX-Sketch/X-Sketch/blob/master/X_Sketch_Appendices.pdf

[29] B. Lahiri, S. Tirthapura, and J. Chandrashekar, "Space-efficient tracking of persistent items in a massive data stream," *The ASA Data Science Journal*, vol. 7, no. 1, pp. 70–92, 2014.

[30] H. Dai, M. Shahzad, A. X. Liu, and Y. Zhong, "Finding persistent items in data streams," *Proc. VLDB Endow.*, vol. 10, no. 4, pp. 289–300, 2016.

[31] H. Dai, M. Shahzad, A. X. Liu, M. Li, Y. Zhong, and G. Chen, "Identifying and estimating persistent items in data streams," *IEEE/ACM Transactions on Networking (ToN)*, vol. 26, no. 6, pp. 2429–2442, 2018.

[32] H. Dai, M. Li, A. X. Liu, J. Zheng, and G. Chen, "Finding persistent items in distributed datasets," *IEEE/ACM Transactions on Networking (ToN)*, vol. 28, no. 1, pp. 1–14, 2020.

[33] Y. Zhang, J. Li, Y. Lei, T. Yang, Z. Li, G. Zhang, and B. Cui, "On-off sketch: a fast and accurate sketch on persistence," *Proc. VLDB Endow.*, vol. 14, no. 2, pp. 128–140, 2021.

[34] H. Huang, Y.-E. Sun, S. Chen, S. Tang, K. Han, J. Yuan, and W. Yang, "You can drop but you can't hide: k-persistent spread estimation in high-speed networks," in *Proc. INFOCOM*, 2018, pp. 1889–1897.

[35] H. Huang, Y.-E. Sun, C. Ma, S. Chen, Y. Zhou, W. Yang, S. Tang, H. Xu, and Y. Qiao, "An efficient k-persistent spread estimator for traffic measurement in high-speed networks," *IEEE/ACM Transactions on Networking (ToN)*, vol. 28, no. 4, pp. 1463–1476, 2020.

[36] Y.-E. Sun, H. Huang, S. Chen, Y. Zhou, K. Han, and W. Yang, "Privacy-preserving estimation of k - persistent traffic in vehicular cyber-physical systems," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8296–8309, 2019.

[37] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, 2003.

[38] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theoretical Computer Science*, vol. 312, no. 1, pp. 3–15, 2002.

[39] T. Li, S. Chen, and Y. Ling, "Per-flow traffic measurement through randomized counter sharing," *IEEE/ACM Transactions on Networking (ToN)*, vol. 20, no. 5, pp. 1622 – 1634, 2012.

[40] Y. Zhou, T. Yang, J. Jiang, B. Cui, M. Yu, X. Li, and S. Uhlig, "Cold filter: A meta-framework for faster and more accurate stream processing," in *Proc. SIGMOD*, 2018, pp. 741–756.
[41] P. Jia, P. Wang, J. Zhao, Y. Yuan, J. Tao, and X. Guan, "Loglog filter: Filtering cold items within a large range over high speed data streams," in *Proc. ICDE*, 2021, pp. 804–815.
[42] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *Discrete Mathematics and Theoretical Computer Science*, 2007, pp. 137–156.
[43] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. EDBT*, 2013, pp. 683–692.
[44] T. Yang, Y. Zhou, H. Jin, S. Chen, and X. Li, "Pyramid sketch: A sketch framework for frequency estimation of data streams," *Proc. VLDB Endow.*, vol. 10, pp. 1442–1453, 2017.
[45] L. Tang, Q. Huang, and P. P. C. Lee, "Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams," in *Proc. INFOCOM*, 2019, pp. 2026–2034.
[46] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proc. SIGCOMM*, 2018, pp. 561–575.
[47] "The source code of Bob Hash." [Online]. Available: http://burtleburtle.net/bob/hash/evahash.html
[48] "MAWI Working Group Traffic Archive." [Online]. Available: http://mawi.wide.ad.jp/mawi/
[49] "The Data Center Dataset." [Online]. Available: http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html
[50] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. IMC*, 2010, pp. 267–280.
[51] D. M. Powers, "Applications and explanations of zipf's law," in *Proc. NeMLaP3/CoNLL*, 1998, pp. 151–160.
[52] A. Rousskov and D. Wessels, "High-performance benchmarking with web polygraph," *Software: Practice and Experience*, vol. 34, no. 2, pp. 187–211, 2004.
[53] "Frequent Itemset Mining Dataset Repository." [Online]. Available: http://fimi.uantwerpen.be/data/