Hyper-USS: Answering Subset Query Over Multi-Attribute Data Stream

Ruijie Miao[*][†]
Peking University
miaoruijie@pku.edu.cn

Yiyao Zhang[‡]
Nanjing University
zhangyiyao@smail.nju.edu.cn

Guanyu Qu[§][¶]
Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
quguanyu21@mails.ucas.ac.cn

Kaicheng Yang[*][†]
Peking University
ykc@pku.edu.cn

Tong Yang[*][†]
Peking University
yangtongemail@gmail.com

Bin Cui[*]
Peking University
bin.cui@pku.edu.cn

ABSTRACT

[1] Sketching algorithms are considered as promising solutions for answering approximate query on massive data stream. In real scenarios, a large number of problems can be abstracted as *subset query over multiple attributes*. Existing sketches are designed for query on single attributes, and therefore are inefficient for query on multiple attributes. In this work, we propose *Hyper-USS*, an innovative sketching algorithm that supports subset query over multiple attributes accurately and efficiently. To the best of our knowledge, this work is the first sketching algorithm designed to answer approximate query over multi-attribute data stream. We utilize the key technique, *Joint Variance Optimization*, to guarantee high estimation accuracy on all attributes. Experiment results show that, compared with the state-of-the-art (SOTA) sketches that support subset query on single attributes, Hyper-USS improves the accuracy by 16.67× and the throughput by 8.54×. The code is open-sourced at Github.

CCS CONCEPTS

• **Information systems → Data stream mining**; • **Theory of computation → Sketching and sampling**.

KEYWORDS

Sketch; Multi-attribute Data Stream; Subset Query

[*]National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University, Beijing, China
[†]Peng Cheng Laboratory, Shenzhen, China
[‡]Department of Computer Science and Technology, Nanjing university, Nanjing, China
[§]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[¶]University of Chinese Academy of Sciences, Beijing, China
[1]Corresponding author: Tong Yang (yangtongemail@gmail.com)

ACM Reference Format:
Ruijie Miao, Yiyao Zhang, Guanyu Qu, Kaicheng Yang, Tong Yang, and Bin Cui. 2023. Hyper-USS: Answering Subset Query Over Multi-Attribute Data Stream. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599383

1 INTRODUCTION

Approximate query for massive data streams has wide applications in data analysis, especially when processing efficiency is of high priority while errors can be tolerated to a certain degree. In real scenarios, the data stream usually has multiple attributes that users are interested in. That is, each item of the data stream can be denoted as $(e, v_1, v_2, \cdots, v_n)$, where e is the key and $v_1, v_2, \cdots, v_n$ are the attributes of interest. Users may query for the aggregated results over a subset of items on any attribute. If we regard each item in the data stream as one row in a table, the aggregated query can be expressed as:

```
SELECT SUM [,AVG] (Attribute)
FROM table
WHERE Key in Subset
```

We list some potential use cases for subset query over multiple attributes:

Use case 1: ad click analysis. In the ad click stream, each item represents one user viewing a specific ad and includes information such as whether they click the ad, user region and advertising company. For instance, (user_region, ad_id, company_id) is the key, and the number of views (in each item this is 1) and the number of clicks (in each item this is 0 or 1) are attributes. Operators may query for the total number of views for ads from the same company, or the total number of clicks of a specific ad from users in a specific country. These can be expressed as the subset sum over different attributes. Besides, users may also be interested in the click-through rate for users in a certain country, which is defined as the number of clicks over the number of views and can be expressed as subset average over the number of clicks.

Use case 2: Youtube video statistics. In Youtube video statistics, each watch record means a user plays a specific video, and the recorded information includes whether they like the video and their watch time. Operators may query for the total number of

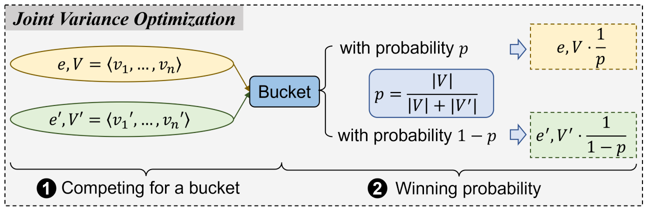

Joint Variance Optimization
$e, V = \langle v_1, \dots, v_n \rangle$
$e', V' = \langle v_1', \dots, v_n' \rangle$
Bucket
with probability p
$p = \frac{|V|}{|V| + |V'|}$
with probability $1 - p$
$e, V \cdot \frac{1}{p}$
$e', V' \cdot \frac{1}{1-p}$
1 Competing for a bucket
2 Winning probability

Figure 1: The key technique of Joint Variance Optimization in Hyper-USS.

plays or the total number of likes for videos in one channel (subset sum). They may also query for the average watch time for videos in one channel (subset average).

Use case 3: ecommerce data analytics. For ecommerce data, each shopping record means a buyer buys a specific product, and include information such as their rating of products. Users may query for the sum of sales volume for one store (subset sum), or the average rating of one product (subset average).

Use case 4: network measurement. In network measurement, the number of packets and the number of bytes are useful attributes in many applications [1, 2]. In traffic traces, each packet is sent from a source IP to a destination IP, with a specific packet size. Users may query for the total number of packets from a certain source IP prefix (subset sum), or the average number of bytes of each packet to one specific destination IP prefix (subset average).

To address approximate query for massive data streams, research community has developed many approximation techniques, and sketches are one of the most popular algorithms among them. Sketching algorithms are typically designed to record information in limited memory while minimizing querying errors. Sketches can be classified into two types. Most sketches [3–10] are designed for point query [2], and existing works [1, 11, 12] have shown that such algorithms are inefficient in supporting subset query. The other type of sketches achieve high accuracy and high processing speed in supporting subset query. However, they are designed for single-attribute data stream model. When applied for multi-attribute data streams, a strawman solution is to build one individual sketch for each attribute. This method wastes memory by repeatedly recording the information of keys in all sketches, and inserting one item requires updating all sketches. Therefore, it suffers from limited performance in both accuracy and insertion throughput.

In this paper, we propose our solution, namely Hyper-USS, which, to the best of our knowledge, is the first sketching algorithm designed to answer approximate query on multi-attribute data stream model. Hyper-USS records the information of keys and all attributes in one sketch. Besides, Hyper-USS provides for all attributes unbiased estimation and optimized estimation variance, which is well acknowledged as the golden principle for accurate subset query [11, 12]. Therefore, Hyper-USS achieves high accuracy and high insertion throughput for subset query over multiple attributes.

The challenge in sketch design lies in how to provide unbiased estimation while optimizing variance for all attributes. To address the problem, we propose our key technique, namely *Joint Variance Optimization*. In order to achieve high accuracy for all attributes, we set the optimization goal as minimizing the sum of estimation variance of all attributes. As a result, optimizing the sum of variance will jointly improve the accuracy on all attributes. Specifically, we generalize the idea of probability proportional to size sampling (PPS) to the multi-attribute scenarios, as shown in Figure 1. When two items compete for a bucket, each item has a probability to win the competition. The winning probability of two items is proportional to the L2 norm of the attributes. If one item successfully stays in the bucket, all its attributes are divided by its winning probability. The state-of-the-art sketches on subset query, USS and CocoSketch, can be regarded as a special case of Hyper-USS in the single-attribute scenarios. We will show in §4 that our technique maintains unbiased estimation while minimizing the sum of variance for all attributes.

While the Joint Variance Optimization technique works in common cases, it fails to perform uniformly well on all attributes when the values of different attributes vary by orders of magnitude. In response, we propose a technique named *Fair Evolution*, which achieves better performance under imbalanced attributes by normalizing attributes before calculating winning probability. For theoretical analysis, we not only provide formal proof on the unbiasedness and the variance optimization, but also study the error bound of Hyper-USS. Beside subset query, Hyper-USS can be applied for point query, identifying heavy hitters, detecting heavy change, finding heavy hitter subsets, and querying the distribution of values.

Our main contribution can be summarized as follows.

- We propose Hyper-USS, which supports subset query over multi-attribute data streams efficiently (§3.1).
- To achieve fairness among imbalanced attributes, we propose Fair Evolution by optimizing normalized errors (§3.2).
- We provide theoretical analysis for Hyper-USS, including the unbiasedness, variance minimization, and the error bound (§4).
- We conduct rich experiments on Hyper-USS. The results show that, compared with SOTA sketches for subset query on single attributes, Hyper-USS can reduce the estimation errors of subset query by 16.67 times, and speed up insertion by 8.54 times (§6).

2 BACKGROUND AND MOTIVATION

In this section, we first give a formal definition for the problem of subset query over multi-attribute data stream, and then we describe the related work.

2.1 Problem Definition

The single-attribute model successfully abstracts the data stream in many scenarios, which is widely studied in the research community. However, in more complicated scenarios, there exists requirements of *subset query over multiple attributes*. We provide the formal definition for the problem as follows.

Problem 1 (Subset Query Over Multiple Attributes). *Suppose the item in the multi-attribute data stream can be denoted as $(e, v_1, v_2, \cdots, v_n)$. Given a target attribute v_i, a subset of keys $\mathcal{S}_k$ and an operator f, we apply the operator on the attribute v_i for any key $k \in \mathcal{S}_k$.*

In this paper we aim to support two classic operators: *sum* and *average*.

- For the operator *sum* on i^{th} attribute over subset $\mathcal{S}_k$, we compute

$$\sum_{(e, v_1, \cdots, v_n), e \in \mathcal{S}_k} v_i$$

[2]Point query refers to querying the results of a single key.

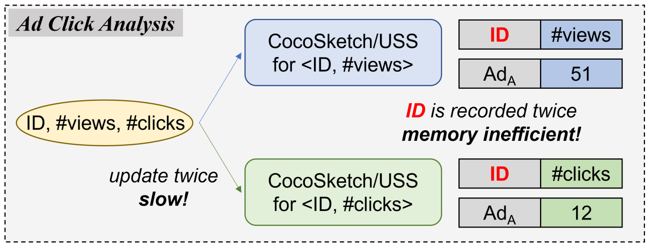


Figure 2: The limitations of SOTA sketches (CocoSketch/USS) on subset query over multiple attributes.

- For the operator *average* on i^{th} attribute over subset $\mathcal{S}_k$, we compute

$$\frac{\sum_{(e,v_1,\cdots,v_n),e\in\mathcal{S}_k} v_i}{|\{(e,v_1,\cdots,v_n) \mid e\in\mathcal{S}_k\}|}$$

2.2 Related Work

2.2.1 Sketching Algorithms.

Sketches are a class of compact data structures designed to answer approximate query with limited memory. Sketching algorithms have the advantages of high throughput and provable error bounds, and are therefore suitable for large data stream processing. Most existing sketching algorithms are discussed in the single-attribute data stream model and only support point query. Some recent works advanced in supporting subset query. Below we discuss these two kinds of sketches and their limitations for subset query over multiple attributes.

Sketches for Point Query. Research community has defined a lot of fundamental tasks and designed corresponding sketches in data stream processing, such as heavy hitter detection [5, 8, 13, 14], pattern mining [15, 16], and more [7, 17, 18]. Most tasks are defined over the single-attribute data stream model, and the corresponding sketches should only support point query, *i.e.*, query the aggregate result of a single key. For example, in heavy hitter detection of single keys, users are only interested in those single keys with large aggregated sum of value.

As pointed out by prior work [11, 12], sketches for point query suffer from inaccuracy and low throughput when applied to subset key query, due to the fact that these sketches support subset query by building one sketch for each queried subset. As a result, the memory of one specific sketch is reduced, causing drop in accuracy, while the insertion of one item requires updating multiple sketches and limits insertion throughput.

Sketches for Subset Query. Recently, researchers have noticed increased demand for subset query. Unbiased SpaceSaving (USS) [11] and CocoSketch [12] are the two representative sketch solutions. The key design for both USS and CocoSketch is similar. An incoming item will choose a bucket and compete with the recorded item in the bucket to decide which item stays in the bucket. The settings of winning probability in both algorithms follow the idea of probability proportional to size sampling (PPS) [19]. Suppose the incoming item (A, V_A) competes with the recorded item (B, V_B). With probability $\frac{V_A}{V_A+V_B}$, the incoming item wins and the bucket is updated to $(A, V_A + V_B)$. With probability $\frac{V_B}{V_A+V_B}$, the recorded item wins and the bucket is updated to $(B, V_A + V_B)$. The probabilistic substitution ensures one bucket providing unbiased estimation and minimized variance for two items. To locate the bucket for the incoming item, USS selects the bucket with the minimum value among all buckets, while CocoSketch selects the bucket with minimum value among the hashed buckets. CocoSketch improves USS in terms of throughput while maintaining high accuracy, therefore regarded as the state-of-the-art sketch solution on subset query.

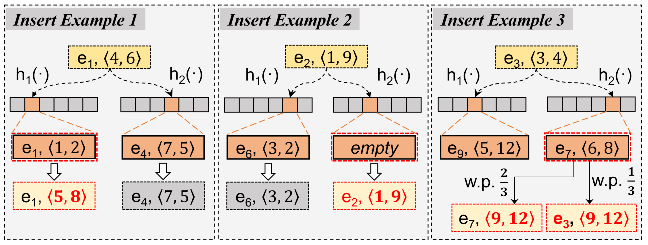


Figure 3: Three insertion examples for Hyper-USS ($n = 2, d = 2$).

However, directly applying CocoSketch for subset query over multiple attributes may suffer from inefficiency in terms of both memory and throughput. In order to support multiple attributes, we have to build one CocoSketch for each attribute. From the aspect of accuracy, as CocoSketch records keys in the bucket, multiple CocoSketches will keep multiple copies of keys, which drags down memory efficiency. From the aspect of throughput, each insertion should update multiple CocoSketches, leading to low throughput. Figure 2 shows an example of ad click analysis with two attributes: the number of views and the number of clicks. Building one sketch for each attribute requires updating two sketches for each incoming item, causing a halving of throughput. Besides, both sketches record the ID of ads, and the duplicated ID records lead to inefficient memory usage.

2.2.2 Multi-attribute Model.

The multi-attribute model arises naturally from many real-world scenarios, and therefore has been widely studied. Skyline query aims to find items no less than any other items on any attribute. Many algorithms [20, 21] are proposed to answer skyline query. Spatio-temporal data mining [22] provide analysis on items with at least one spatial attribute and at least one temporal attribute, which can be used in real-world applications such as traffic management and weather forecast. Research on multi-attributes model also involve multi-dimensional databases [23, 24], business application systems [25], and more [26, 27].

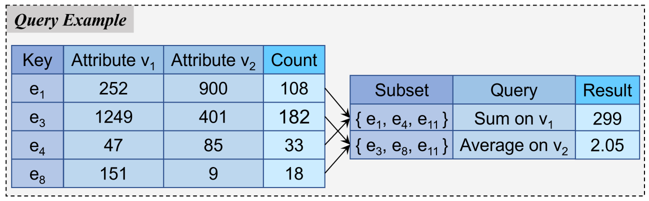
To the best of our knowledge, we are the first to introduce sketching algorithms under the multi-attribute model, with support for subset query on any attribute.

3 THE HYPER-USS ALGORITHM

3.1 Basic Design

Data Structure: The data structure consists of d bucket arrays, and each consisting of w buckets. Each bucket records a key and n attributes. Let $\mathcal{B}_i[j] (1 \leqslant i \leqslant d, 1 \leqslant j \leqslant w)$ be the j^{th} bucket in the i^{th} array. Let $\mathcal{B}_i[j].K$ be the recorded key in the bucket, and $\mathcal{B}_i[j].V[t] (1 \leqslant t \leqslant n)$ be the t^{th} recorded attribute. Each bucket array is associated with one hash function respectively, and we denote the corresponding hash function for i^{th} array as $h_i(\cdot)$.

Insertion: To insert the item $(e, v_1, v_2, \cdots, v_n)$, we first use d hash functions to hash the key e to d buckets $(\mathcal{B}_i[hash_i(e)], 1 \leqslant i \leqslant d)$


Query Example

Key	Attribute v_1	Attribute v_2	Count
e_1	252	900	108
e_3	1249	401	182
e_4	47	85	33
e_8	151	9	18

Subset	Query	Result
$\{e_1, e_4, e_{11}\}$	Sum on v_1	299
$\{e_3, e_8, e_{11}\}$	Average on v_2	2.05

Figure 4: A query example of Hyper-USS ($n = 2$).

in d arrays. In matching the key of incoming item with existing keys in the buckets, there are three possible cases:

Case 1: e is recorded in $\mathcal{B}_k[hash_k(e)]$, one of the d hashed buckets. In this case, we add the attributes $v_1, \cdots, v_n$ to the bucket. For each dimension t, the t^{th} attribute $\mathcal{B}_k[hash_k(e)].V[t]$ is increased by v_t.

Case 2: e is not recorded in any one of the d buckets, and at least one bucket is still empty. In such case, we select one empty bucket to record the incoming item.

Case 3: e is not recorded in any of the d buckets, and all buckets are occupied. In such cases, we look for the bucket with the smallest L2 norm of values. The incoming item e and the item in the bucket are considered as in competition, and we apply the technique of *Joint Variance Optimization*. Suppose the hashed bucket in the k^{th} array has the smallest L2 norm, we set the winning probability of the incoming item $\mathcal{P}$ according to the following formula:

$$\mathcal{P} = \frac{\sqrt{\sum_t v_t^2}}{\sqrt{\sum_t v_t^2} + \sqrt{\sum_t \mathcal{B}_k[hash_k(e)].V[t]^2}}$$

With probability of $\mathcal{P}$, the recorded item in $\mathcal{B}_k[hash_k(e)]$ is replaced by $(e, v_1, \cdots, v_n)$, and then each attribute is divided by $\mathcal{P}$. Otherwise, the item in the bucket wins the competition, and each recorded attribute is divided by $1 - \mathcal{P}$.

Insertion Examples (Figure 3): In the first example, we aim to insert the item $(e_1, 4, 6)$, and we use two associated hash function to locate two buckets, respectively. The bucket in the first array matches the key, so the recorded attributes are updated. The hashed bucket in the second array is not changed. In the second example, we insert the item $(e_2, 1, 9)$. Neither of the two hashed buckets match e_2, and there exists an empty bucket, so the item is inserted to the empty bucket. In the third example, we insert the item $(e_3, 3, 4)$. Neither of the two hashed buckets match e_3, and the L2 norm of the second bucket is smaller, so we update the second bucket. The replacement probability $\mathcal{P}$ is set to $\sqrt{3^2+4^2}/(\sqrt{3^2+4^2}+\sqrt{6^2+8^2}) = 1/3$. With probability 1/3, the key is replaced by e_3, and each attribute is divided by 1/3. Otherwise, the key is unchanged, and all attributes are divided by 2/3.

Query: To query the subset sum, we extract all non-empty bucket in the result sketch and build a table with $n+1$ columns (one key and n attributes). Then we output the query result of the subset sum over the table.

Support for Subset Average: Subset average can be represented by the division of two subset sums: the subset sum of one attribute, and the frequency sum of the subset. We can estimate the latter by creating a virtual attribute v_{n+1}. For each item, we set the virtual attribute $v_{n+1} = 1$. The subset average over the attribute v_i is the subset sum of attribute v_i over the subset sum of attribute v_{n+1}.

Query Example (Figure 4): The left table shows an example of the table built from the result sketch. To query the sum over the subset $\{e_1, e_4, e_{11}\}$ on the attribute v_1, we apply the operations on the table. We aggregate the first and the third rows in the table. For e_{11}, as it does not appear in the table, its attributes are all estimated as 0. The query result is $252 + 47 + 0 = 299$. To query the average over the subset $\{e_3, e_8, e_{11}\}$ on the attribute v_2, we compute the subset sum on v_2 over the subset sum on the added virtual attribute (denoted as *Count*). The query result is $(401+9+0)/(182+18+0) = 2.05$.

3.2 Dealing with Imbalanced Attributes

In the basic version, the winning probability depends on the L2 norm of n equal-weighted attributes. However, in real scenarios, the values of different attributes may have different orders of magnitude. Such attributes with large values will dominate the L2 norm, thus dominate the update process and hurt the sketch performance on other attributes. To better illustrate the problem, consider video statistics with two attributes: the number of views and the number of likes. The number of views can be orders of magnitude larger than the number of likes. Therefore, when calculating the L2 norm of these two attributes, it is often dominated by the number of views. Such updating process will lead to unacceptable errors in the estimation of the number of likes.

To deal with imbalanced attributes, we propose the technique named *Fair Evolution*. The key design is to normalize n attributes before calculating the L2 norm for the winning probability. During the insertion, we compute the average of all past items on all n attributes, $A_i, 1 \leqslant i \leqslant n$. The i^{th} attribute is divided by A_i before calculating the L2 norm. Therefore, the winning probability of the incoming item is set according to the following adjusted formula,

$$\mathcal{P} = \frac{\sqrt{\sum_t (v_t/A_t)^2}}{\sqrt{\sum_t (v_t/A_t)^2} + \sqrt{\sum_t (\mathcal{B}_k[hash_k(e)].V[t]/A_t)^2}}$$

Besides, in the third case of the insertion when there is no matched bucket and no empty bucket, instead of looking for the bucket with smallest L2 norm, we look for the bucket with smallest normalized L2 norm. In other word, all L2 norm in the basic version is replaced by the normalized L2 norm, and the rest of the algorithm is unchanged.

3.3 The Rationale of Hyper-USS

As the subset average query is reduced to subset sum query, the high accuracy on subset sum estimation is the key problem. In this section, we provide the rationale behind the design of Hyper-USS.

Firstly, as pointed out by prior work [12], accurate subset sum estimation requires unbiased estimation. Applying a biased sketch, *e.g.*, CM sketch [3], to subset sum estimation, will result in unacceptably accumulated errors. The design of Hyper-USS ensures that, when two items compete for one bucket (case 3 in the insertion), Hyper-USS still provides unbiased estimation on any attribute for both items. To do so, Hyper-USS lets each item has a probability to win the competition, and scale all attributes of the winning item according to its winning probability. The detailed proof is shown in §4.1.

Given the unbiasedness, Hyper-USS achieves accurate subset sum estimation by optimizing the variance of the estimation. As shown in Table 1, the SOTA sketches on subset query over single

Sketch	Optimization Goal
USS/CocoSketch	minimize $\sum_e \left(S_i(e) - \widehat{S_i}(e)\right)^2$
The basic version	minimize $\sum_i \sum_e \left(S_i(e) - \widehat{S_i}(e)\right)^2$
The optimized version	minimize $\sum_i \frac{1}{W_i^2} \sum_e \left(S_i(e) - \widehat{S_i}(e)\right)^2$

Table 1: Optimization goals of different sketches. $S_i(\cdot), \widehat{S_i}(\cdot)$ denote the real and estimated sum on i^{th} attribute respectively. $W_i(\cdot)$ denotes the average value of all items on i^{th} attribute.

attributes, USS and CocoSketch, minimize the sum of estimation variance on all single keys. Hyper-USS aims to provide accurate estimation on all attributes, and therefore minimizes the sum of variance for all keys on all attributes. It is noticeable that, for the insertion of Hyper-USS, the specific choice of the winning probability $\mathcal{P}$ does not affect the unbiasedness property. By setting the winning probability proportional to the L2 norm of all attributes, Hyper-USS accomplishes the optimization goal. Besides, selecting the bucket with the smallest L2 norm for updating also targets at the optimization goal. The detailed proof is shown in §4.2.

From the view of the optimization goal, under imbalanced attributes, the larger attributes usually have larger variance. Therefore, simply optimizing the sum of variance of all attribute will be more beneficial for those larger attributes, as they have more room for optimization. A more reasonable optimization goal is to normalize the sum of variance on the i^{th} attribute according to the average value on i^{th} attribute W_i. During the processing of the data stream, the precise value of W_i is not known, as the future data cannot be accessed. Hyper-USS uses the average value A_i of historical items to approximate W_i. By normalizing attributes before calculating the winning probability, the optimized version accomplishes the new optimization goal, and the proof is shown in §4.2. Intuitively, the basic version can be regarded as optimizing "sum of errors", while the optimized version targets at "sum of normalized errors". The latter is thus more suitable for imbalanced attributes.

4 ANALYSIS

In this section, we provide mathematical analysis for Hyper-USS. We first prove the unbiasedness of subset sum estimation on any attribute in §4.1. Then we prove how the basic and the optimized version achieve their optimization goals of variance optimization in §4.2. We further provide the analysis of the error bound in §4.3.

4.1 Unbiasedness of Hyper-USS

Theorem 1. *For any attribute v_i, Hyper-USS provides unbiased sum estimation for any subset $\mathcal{S}$,*

$$\mathbb{E}\left[\widehat{S_i}(\mathcal{S})\right] = S_i(\mathcal{S})$$

Here $\widehat{S_i}(\cdot)$ denotes the estimated subset sum on i^{th} attribute, and $S_i(\cdot)$ denotes the real subset sum on i^{th} attribute.

Proof. We first prove that Hyper-USS gives unbiased sum estimation for any single key k on any attribute. Consider inserting the item $(e, w_1, w_2, \cdots, w_n)$. If one bucket matches e, the i^{th} attribute in the bucket will increase by v_i, and the increment of estimation for e is unbiased. If no matched bucket is found, the bucket with smallest L2 norm is updated, and we suppose the updated bucket is $\mathcal{B}_t[h_t(e)]$. After the insertion, the expected increment of estimation for the key e on attribute v_i is,

$$\frac{w_i}{\mathcal{P}} \cdot \mathcal{P} + 0 \cdot (1 - \mathcal{P}) = w_i$$

The expected increment for the key $\mathcal{B}_t[h_t(e)].K$ is,

$$\left(\frac{\mathcal{B}_t[h_t(e)].V[i]}{1 - \mathcal{P}} - \mathcal{B}_t[h_t(e)].V[i]\right)\cdot(1-\mathcal{P}) - \mathcal{B}_t[h_t(e)].V[i]\cdot\mathcal{P} = 0$$

As a result, during the insertion the estimated sum of any key is unbiased. For any subset S, we have

$$\mathbb{E}\left[\widehat{S_i}(\mathcal{S})\right] = \sum_{e\in\mathcal{S}} \mathbb{E}\left[\widehat{S_i}(e)\right] = \sum_{e\in\mathcal{S}} S_i(e) = S_i(\mathcal{S})$$

□

4.2 Variance Optimization in Hyper-USS

Analysis for the choice of $\mathcal{P}$. We first consider the basic version of Hyper-USS with only one array and one associated hash function, and discuss why the choice of winning probability $\mathcal{P}$ is theoretically optimal for the optimization goal.

Theorem 2. *In the basic version with $d = 1$, Hyper-USS minimizes the sum of variance of all keys on all attributes, shown as follows.*

$$minimize \quad \sum_{i=1}^{n}\sum_{e}\left(S_i(e) - \widehat{S_i}(e)\right)^2 \quad (1)$$

Proof. We consider the increment of Eq. (1) for the insertion of each item $(e, w_1, \cdots, w_n)$. If one bucket matches e or there is at least one empty bucket, i^{th} attribute of item e has an increment of w_i, and the increment of Eq. (1) is 0. Otherwise, suppose the updated bucket is $\mathcal{B}[h(e)]$, and the record attributes are $u_i, 1 \leqslant i \leqslant n$. The increment variance only involves the keys of e and $\mathcal{B}[h(e)].K$, and we have,

$$\begin{aligned}\sum_{i=1}^{n}\sum_{e}\Delta\left(S_i(e) - \widehat{S_i}(e)\right)^2 &= \sum_{i=1}^{n}\mathcal{P}\cdot\left(\left(\frac{w_i}{\mathcal{P}} - w_i\right)^2 + u_i^2\right)\\ &+ (1-\mathcal{P})\cdot\left(w_i^2 + \left(\frac{u_i}{1-\mathcal{P}} - u_i\right)^2\right)\\ &= \frac{\sum_{i=1}^{n} w_i^2}{\mathcal{P}} + \frac{\sum_{i=1}^{n} u_i^2}{1-\mathcal{P}} - \sum_{i=1}^{n} w_i^2 - \sum_{i=1}^{n} u_i^2\end{aligned}$$

When setting

$$\mathcal{P} = \frac{\sqrt{\sum_i w_i^2}}{\sqrt{\sum_i w_i^2} + \sqrt{\sum_i u_i^2}}$$

the variance increment is minimized. □

Analysis for selecting the bucket with minimal L2 norm. We then analyze the general case of the basic version, and discuss why we choose to update the bucket with the minimal L2 norm.

Theorem 3. *In the basic version with $d > 0$, Hyper-USS minimizes the sum of variance of all keys on all attributes, shown as follows.*

$$minimize \quad \sum_{i=1}^{n}\sum_{e}\left(S_i(e) - \widehat{S_i}(e)\right)^2 \quad (2)$$

Proof. Suppose the incoming item is $(e, w_1, \cdots, w_n)$. According the the proof in Theorem 2, when there is no matched bucket and no empty bucket, suppose the updated bucket is $\mathcal{B}_t[h_t(e)]$, and its attributes are $u_i, 1 \leqslant i \leqslant n$. The minimal increment of Eq. (2) will be,

$$\sum_{i=1}^{n}\sum_{e}\Delta\left(S_i(e)-\widehat{S_i}(e)\right)^2 = \frac{\sum_{i=1}^{n} w_i^2}{\mathcal{P}} + \frac{\sum_{i=1}^{n} u_i^2}{1-\mathcal{P}} - \sum_{i=1}^{n} w_i^2 - \sum_{i=1}^{n} u_i^2$$
$$= 2\cdot\sqrt{\sum_{i=1}^{n} w_i^2}\cdot\sqrt{\sum_{i=1}^{n} u_i^2}$$

For the insertion of Hyper-USS, we look for the bucket with the minimal L2 norm among d hashed buckets as the updated bucket. Therefore, when $d > 0$, Eq. (2) is also minimized. □

Analysis for the optimized version. Then we show that the optimized version minimizes the normalized sum of errors.

Theorem 4. *Suppose the average of value on i^{th} attribute is fixed, denoted as W_i. In the optimized version, Hyper-USS minimizes the normalized sum of variance, shown as follows.*

$$\textit{minimize} \quad \sum_{i=1}^{n}\frac{1}{W_i^2}\sum_{e}\left(S_i(e)-\widehat{S_i}(e)\right)^2 \tag{3}$$

Proof. Since W_i is fixed throughout the insertion, the A_i maintained by the optimized version is exactly equal to W_i. We can divide the i^{th} attribute of each item by W_i, and then the algorithm and the optimization goal become the same as those of the basic version. The proof is the same as the proof of Theorem 2 and Theorem 3. □

4.3 Error Bound

In this section, we provide analysis on the error bound for the basic version of Hyper-USS. The proof is deferred to Appendix A.

Theorem 5. *In the basic version, the error of Hyper-USS's estimation on an arbitrary attribute of an arbitrary key e can be bounded as follows, where U is the total number of inserted items and L is the upper bound of L2 norm for each inserted item.*

$$\Pr\left[|\hat{S}_j(e) - S_j(e)| \geqslant \epsilon\right] \leqslant \frac{4U^2L^2}{w^2\epsilon^2}$$

Corollary 1. *In the basic version, the error of Hyper-USS's estimation on an arbitrary attribute of a subset $\mathcal{T}$ can be bounded as follows.*

$$\Pr\left[|\hat{S}_j(\mathcal{T}) - S_j(\mathcal{T})| \geqslant \epsilon\right] \leqslant \frac{4\,|\mathcal{T}|\,U^2L^2}{w^2\epsilon^2}$$

5 DISCUSSION

Supporting negative attributes. We propose different schemes to support negative attributes for following two different cases.

- Negative attribute value $-x$ should have the same winning probability with positive attribute value $+x$. In such cases, we can treat negative attributes in the same manner as we treat non-negative attributes, because the winning probability depends on the square of x.
- Negative attribute value $-x$ should have lower winning probability compared with positive attribute value $+x$, and suppose the attribute has a lower bound $-R$. In such cases, for the incoming item with attribute $-x$, we will consider the attribute as $-x + R$ and insert the item, and subtract $R \cdot \textit{EstimatedCount}$ from the results when querying.

Supporting other applications. In addition to subset sum query and subset average query, Hyper-USS also supports:

- *Point query.* Point query is the special case of subset query where the subset consists only one key.
- *Heavy hitter [8, 13, 14, 28].* Heavy hitters on attribute v_i refers to keys whose aggregated sum on attribute v_i are above a threshold, which can be reduced to point query.
- *Heavy change [29].* For two adjacent windows, heavy change refers to the items whose frequency change is larger than a threshold. By detecting heavy hitters in both windows and compare the difference, Hyper-USS can support heavy change detection.
- *Heavy hitter subset (hierarchical heavy hitter [30–32]).* Finding heavy hitter subsets is an extension of finding heavy hitters. Suppose all keys are partitioned into several subsets, heavy hitter subset query on attribute v_i aims to find those subsets whose subset sum on v_i is above a threshold. Heavy hitter subset query can be reduced to subset sum query and thus be supported By walking through the different levels of the hierarchy and finding heavy hitter subsets, Hyper-USS supports hierarchical heavy hitter.
- *Distribution of values.* Given a key/subset of keys, Hyper-USS supports query for the distribution of values in i^{th} attribute: how many items come with value v in the i^{th} attribute, for all different v? To do so, we can duplicate the i^{th} attribute as an additional field of the key, and the problem reduces to the subset sum query.

Hyper-USS cannot support arbitrary user-defined functions, as it is primarily designed for subset sum estimation. However, we believe an efficient support of subset sum query covers a wide range of real-world applications.

6 EVALUATION

We conduct rich experiments to compare Hyper-USS with the SOTA sketches on subset query over single attributes, USS [11] and CocoSketch [12], and show that:

- In multi-attribute scenarios, how accurate Hyper-USS can achieve on different tasks compared with USS and CocoSketch;
- How fast Hyper-USS processes items compared with USS and CocoSketch;
- How parameters of Hyper-USS and the datasets influence the sketch performance;
- How the optimized version of Hyper-USS outperforms the basic version under imbalanced attributions.

6.1 Experiment Setup

Datasets:

(1) *The synthetic dataset.* The keys in the synthetic dataset are randomly generated integers. To simulate the heavy-tailed distribution in real world workload, the frequency of keys follows the Zipf distribution [33] with skewness 1.5 according to prior works [7, 34].

For each item, the value of each attribute follows exponential distribution. We choose exponential distribution to generate positive integers. The synthetic dataset contains 50M items, and each item consists of a 40-byte key and 10 attributes. For 10 attributes we select values between 1 and 16 as the mean of exponential distributions.

(2) *The Criteo dataset.* The Criteo dataset [35] contains feature values and click feedbacks for millions of display ads over a period of 24 days. The Criteo dataset contains 13 integer features and 26 categorical features. USS [11] uses this dataset to evaluate subset query over single attributes. We select 5 categorical features as the key and 5 count features as attributes, and use the first 50M items.

(3) *The NBA dataset.* The NBA dataset [36] contains player statistics from the 2004 season onwards. Each player is identified by 4 keys. For each game, there are 19 statistical indicators provided for each player, and we select all 19 fields as attributes.

(4) *The CAIDA dataset.* The CAIDA dataset [37] contains one hour of anonymous network traces collected from the Equinix-Chicago monitor in 2018. We use the source IP and destination IP as the ID of items, and use two attributes: the packet size and the packet interval. We use 1-minute interval, which contains around 27M items and 85K distinct items.

Metrics:

- *F1 Score:* F1 Score is $\frac{2 \cdot RR \cdot PR}{RR+PR}$. Here, RR is the recall rate, defined as the ratio of the number of correctly reported instances to the number of ground truths. PR is the precision rate, defined as the ratio of the number of correctly reported instances to the number of reported items. F1 score is used to evaluate the accuracy on finding heavy hitters and heavy hitter subsets.
- *Average Absolute Error (AAE):* $\frac{\sum_{\mathcal{S}\in\Psi}\left|f(\mathcal{S})-\widehat{f}(\mathcal{S})\right|}{|\Psi|}$, where $f(\mathcal{S})$ is the ground truth, $\widehat{f}(\mathcal{S})$ is the output query result, and Ψ is the query set.
- *Average Relative Error (ARE):* $\frac{1}{|\Psi|}\sum_{\mathcal{S}\in\Psi}\frac{|f(\mathcal{S})-\widehat{f}(\mathcal{S})|}{f(\mathcal{S})}$.
- *Throughput*: Million items per second (Mips). The throughput numbers are the median value among 5 independent trials.

CPU Implementation: We implement Hyper-USS and its competitors in C++, and the code is open-sourced at Github [38]. As in previous work [12], we implement a throughput-enhanced USS with a hash table and a double link list, because the naive USS is too slow. In the optimized USS, the hash table is used to accelerate the process of checking whether and where a key is stored in the data structure, and the double link list is used to accelerate finding the minimal bucket by sorting buckets. For CocoSketch we use the recommended parameters in paper [12]. For Hyper-USS, we set $d = 2$ by default.

FPGA Implementation: We also implement Hyper-USS on the Xilinx XC7VX690T FPGA. The evaluation of the FPGA implementation is shown in Appendix D.

6.2 Accuracy

6.2.1 *Experiments on the Synthetic Dataset.*

We evaluate the accuracy of subset query and point query on the synthetic dataset. For subset query, we randomly select 1000 subsets, each of size 1000, and compute their average metric for all trials on all attributes. For point query, we compute the average metric for all attributes on 1000 random items.

Subset query (Figure 5): The experimental results show that, Hyper-USS is more accurate for subset query on the synthetic dataset. Compared with USS and CocoSketch, for subset sum estimation, Hyper-USS reduces the estimation AAE by up to 16.69×, 5.65×, and reduces ARE by up to 13.83×, 4.84×, respectively. For subset average estimation, Hyper-USS reduces AAE by up to 32.15×, 16.67×, and reduces ARE by up to 28.13×, 12.20×, respectively.

Point query (Figure 6): The experimental results show that, Hyper-USS achieves higher accuracy for point query on the synthetic dataset. Compared with USS and CocoSketch, for point sum estimation, Hyper-USS reduces AAE by up to 55.26×, 21.13×, and reduces ARE by up to 20.75×, 13.07×, respectively. For point average estimation, Hyper-USS reduces AAE by up to 21.72×, 21.36×, and reduces ARE by up to 19.52×, 18.84×, respectively. Besides, Hyper-USS requires much more memory for accurate point query than subset query. This is because Hyper-USS provides unbiased estimation for a single key, and thus it can achieve accurate subset query even when the point query is not accurate enough.

6.2.2 *Experiments on the Real Datasets.*

We evaluate the accuracy for subset query and point query on the Criteo dataset. We evaluate the subset query on the heavy hitter subsets and the point query on the heavy hitters. For each attribute, we set the heavy hitter threshold to the 10^{-5} of the total sum on that attribute. Recall that in the Criteo dataset, the key of each item consists of 5 features. For heavy hitter subsets, a natural way is to partition according to values in a certain subset of the 5 features (*e.g.*, when partitioning according to the first 3 features, the result subsets are $(1, 2, 3, *, *)$, $(5, 3, 4, *, *)$, *etc.*). By doing so we detect heavy hitters on a specific subset of the 5 features. To prove universality, we go through all $C_5^1 + C_5^2 + C_5^3 + C_5^4 = 30$ subsets of features for partitioning.

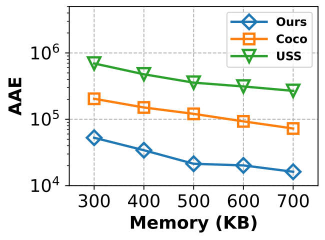
We also conduct evaluations on the NBA dataset in Appendix B, and evaluate the performance of Hyper-USS on other tasks in Appendix C, including heavy hitter, heavy hitter subset, heavy change and distribution of values.

Subset query (Figure 7): The experimental results show that, Hyper-USS is more accurate for subset query in the Criteo dataset. Compared with USS and CocoSketch, for subset sum estimation, Hyper-USS reduces AAE by up to 30.91×, 7.51×, and reduces ARE by up to 14.78×, 2.91×, respectively. For subset average estimation, Hyper-USS reduces AAE by up to 12.70×, 10.28×, and reduces ARE by up to 13.02×, 8.63×, respectively.

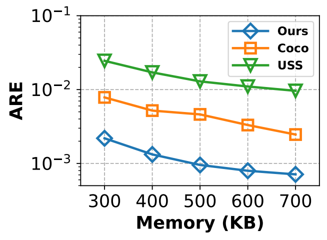
Point query (Figure 8): The experimental results show that, Hyper-USS also achieves higher accuracy for point query on the Criteo dataset. Compared with USS and CocoSketch, for point sum estimation, Hyper-USS reduces AAE by up to 275.87×, 14.62×, and reduces ARE by up to 74.06×, 6.07×, respectively. For point average estimation, Hyper-USS reduces AAE by up to 16.88×, 9.99×, and reduces ARE by up to 32.26×, 13.73×, respectively.

6.3 Throughput

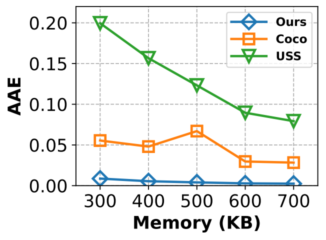
We evaluate the insertion throughput of Hyper-USS, and compare it with USS and CocoSketch on both the synthetic dataset and the Criteo dataset.



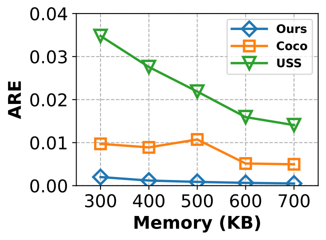
(a) Subset sum - AAE



(b) Subset sum - ARE

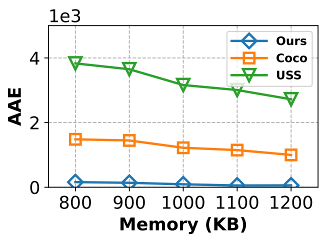


(c) Subset average - AAE

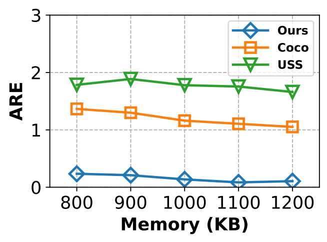


(d) Subset average - ARE

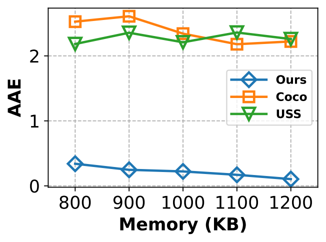
Figure 5: Experiments on the subset query on the synthetic dataset.



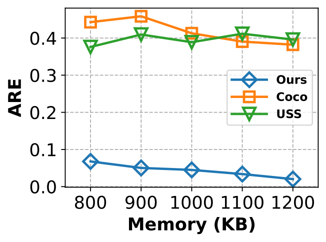
(a) Point sum - AAE



(b) Point sum - ARE

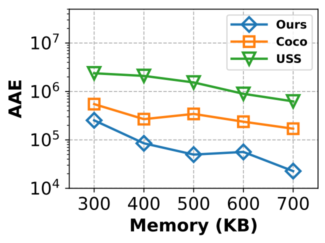


(c) Point average - AAE

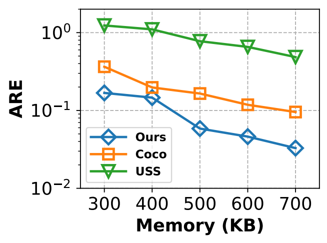


(d) Point average - ARE

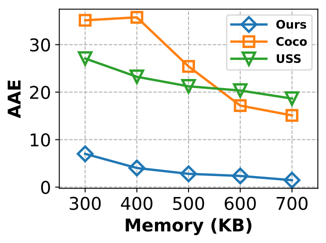
Figure 6: Experiments on the point query on the synthetic dataset.



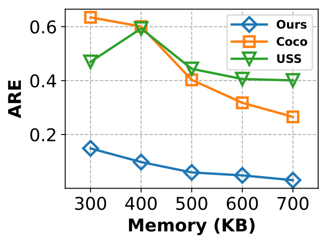
(a) Subset sum - AAE



(b) Subset sum - ARE

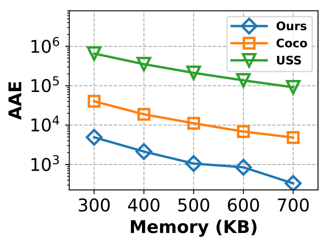


(c) Subset average - AAE

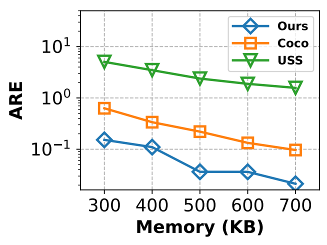


(d) Subset average - ARE

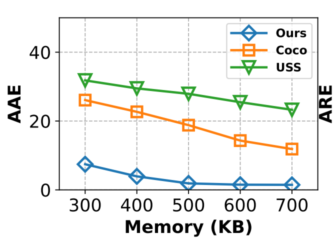
Figure 7: Experiments on the subset query on the Criteo dataset.



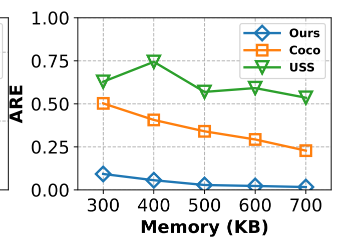
(a) Point sum - AAE



(b) Point sum - ARE

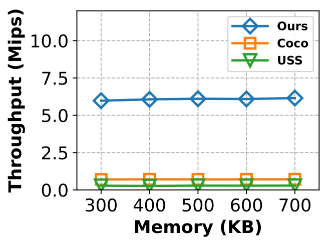


(c) Point average - AAE

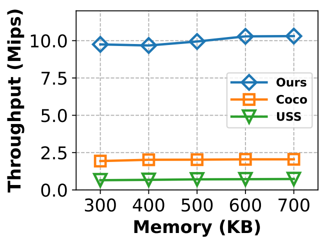


(d) Point average - ARE

Figure 8: Experiments on the point query on the Criteo dataset.



(a) The synthetic dataset



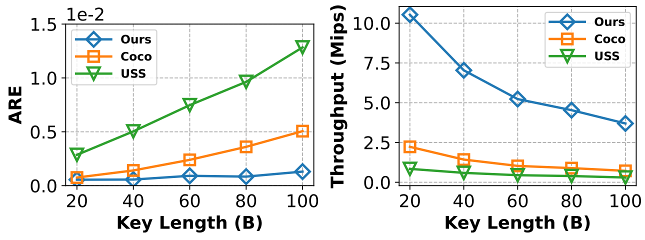
(b) The Criteo dataset

Figure 9: Experiments on the insertion throughput.

Experimental results (Figure 9): The results show that, Hyper-USS greatly speeds up the insertion process. Hyper-USS provides more than 21.01×, 8.54× speed up on the synthetic dataset, and more than 14.04×, 4.79× speed up on the Criteo dataset. Hyper-USS achieves around 6.08Mips on the synthetic dataset, and around 9.99Mips on the Criteo dataset. It achieves higher insertion throughput on the Criteo dataset, because in the Criteo dataset the length of key and the number of attributes is smaller.

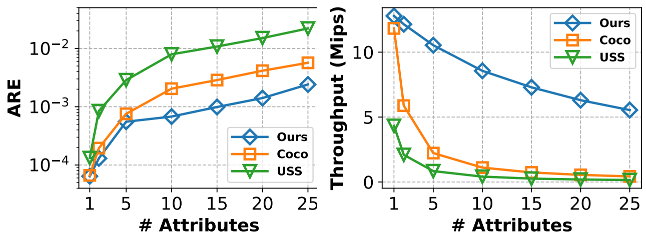
6.4 Microbenchmark

In this section, we evaluate the effect of different parameters in the dataset and the algorithm.



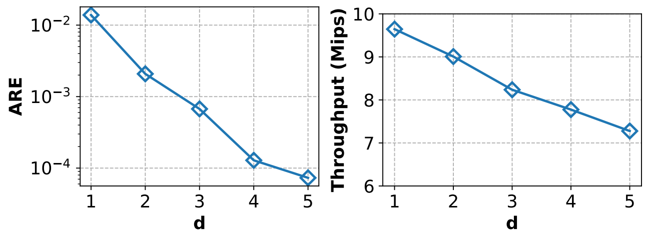
(a) ARE (b) Throughput

Figure 10: Experiments on the key length.



(a) ARE (b) Throughput

Figure 11: Experiments on the number of attributes.

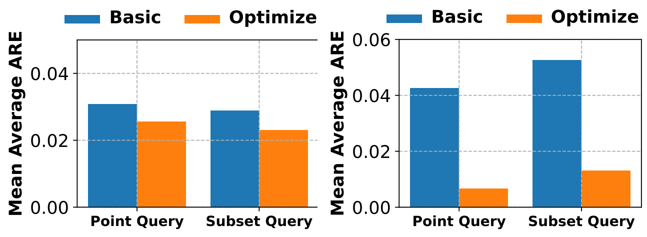


(a) ARE (b) Throughput

Figure 12: Experiments on the number of arrays *d*.

The effect of the key length (Figure 10): We fixed the number of attributes to be 5, and vary the key length in the synthetic dataset from 20 bytes to 100 bytes. The experimental results show that, as the key length increases, the effect on the accuracy of Hyper-USS is more significant. When the key length is 100 bytes, the ARE of Hyper-USS is 0.0013, while those of CocoSketch and USS are 0.0050, 0.0128, respectively. The throughput of Hyper-USS drops as key length grows, but is still higher than 3.70Mips. The drop rate is higher than other algorithms because other algorithms are too slow.

The effect of the attribute number (Figure 11): We fixed the key length to be 20 bytes, and vary the number of attributes in the synthetic dataset from 1 to 25. The experimental results show that, as the number of attribute increases, the effect on the accuracy of Hyper-USS is more significant. When the number of attribute is 25, the ARE of Hyper-USS is 0.0024, while those of CocoSketch and USS are 0.0056, 0.0220, respectively. The throughput of Hyper-USS decreases as the number of attributes grows, but maintains higher than 5.54Mips. When the number of attribute is 1, Hyper-USS degenerates to CocoSketch. However, it is worth noting that even with only two attributes, Hyper-USS still outperforms CocoSketch, achieving 1.50× lower ARE and 2.07× higher throughput.



(a) The Criteo dataset (b) The CAIDA dataset

Figure 13: Experiments on the optimized version.

The effect of *d* (Figure 12): We vary the number of arrays from 1 to 5 in the Hyper-USS. The experimental results show that, the parameter d in Hyper-USS serves as a trade-off between accuracy and throughput. When d increases from 1 to 5, the accuracy improves as the ARE decreases from 0.013 to 0.00007, and the insertion throughput decreases from 9.64Mips to 7.28Mips. Setting $d = 2$ is a reasonable choice that balances the accuracy and the insertion throughput.

6.5 The Optimization

In this section, we compare the basic version and the optimized version of Hyper-USS on the Criteo dataset and the CAIDA dataset. For the Criteo dataset, to simulate imbalanced attributes, we scale one attribute of each item by 10^5. For the CAIDA dataset, the value of packet interval is around 10^{-6}, and the value of packet size is usually 10^3, and thus two attributes are already imbalanced. With imbalanced attributes, ARE is more suitable metric for accuracy. In each trial, we perform subset query and point query respectively on heavy hitter subsets / heavy hitters, and then record the average ARE. We conduct 1000 trials and calculate the mean of average ARE.

Experimental results (Figure 13): The results show that, the optimized version performs better when attributes are imbalanced. In both point query and subset query, the mean ARE is reduced by up to 21% on the Criteo dataset and 75% on the CAIDA dataset, which indicates the optimized version has improved accuracy.

7 CONCLUSIONS

A wide range of practical problems can be abstracted as subset query over multi-attribute data stream. While sketching algorithms are considered as promising solutions for processing large data, few are designed to adequately support query over multiple attributes. We propose Hyper-USS, the first sketching solution that provides subset query over multiple attributes accurately and efficiently. With Joint Variance Optimization, Hyper-USS provides unbiased estimation and optimizes estimation variance jointly, addressing the challenge of accurately estimating multiple attributes in the sketch design. Our evaluations show that Hyper-USS outperforms existing solutions in both accuracy and insertion throughput in multi-attribute scenarios. The code is available at Github [38].

ACKNOWLEDGMENT

We thank all anonymous reviewers for their help in improving this paper. We also thank Yikai Zhao for his valuable suggestions in the algorithm design and theoretical analysis. 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 and 61832001).

REFERENCES

[1] Ran Ben-Basat, Gil Einziger, Roy Friedman, Marcelo Caggiani Luizelli, and Erez Waisbard. Constant time updates in hierarchical heavy hitters. In *SIGCOMM 2017*. ACM, 2017.

[2] Michel Cukier, Robin Berthier, Susmit Panjwani, and Stephanie Tan. A statistical analysis of attack data to separate attacks. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 383–392. IEEE, 2006.

[3] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 2005.

[4] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 2004.

[5] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In Thomas Eiter and Leonid Libkin, editors, *ICDT 2005*, Lecture Notes in Computer Science. Springer, 2005.

[6] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: adaptive and fast network-wide measurements. In *SIGCOMM 2018*. ACM, 2018.

[7] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1449–1463, 2016.

[8] Jizhou Li, Zikun Li, Yifei Xu, Shiqi Jiang, Tong Yang, Bin Cui, Yafei Dai, and Gong Zhang. Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams. In *KDD '20*, pages 1574–1584. ACM, 2020.

[9] Daniel Ting. Count-min: Optimal estimation and tight error bounds using empirical error distributions. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2319–2328, 2018.

[10] Ran Ben Basat, Gil Einziger, Michael Mitzenmacher, and Shay Vargaftik. Salsa: self-adjusting lean streaming analytics. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 864–875. IEEE, 2021.

[11] Daniel Ting. Data sketches for disaggregated subset sum and frequent item estimation. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *SIGMOD 2018*. ACM, 2018.

[12] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. Cocosketch: high-performance sketch-based measurement over arbitrary partial key query. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 207–222, 2021.

[13] Bohan Zhao, Xiang Li, Boyu Tian, Zhiyu Mei, and Wenfei Wu. Dhs: Adaptive memory layout organization of sketch slots for fast and accurate data stream processing. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2285–2293, 2021.

[14] Prashant Pandey, Shikha Singh, Michael A Bender, Jonathan W Berry, Martín Farach-Colton, Rob Johnson, Thomas M Kroeger, and Cynthia A Phillips. Timely reporting of heavy hitters using external memory. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1431–1446, 2020.

[15] Haipeng Dai, Muhammad Shahzad, Alex X Liu, and Yuankun Zhong. Finding persistent items in data streams. *Proceedings of the VLDB Endowment*, 10(4):289–300, 2016.

[16] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003.

[17] Zhewei Wei, Ge Luo, Ke Yi, Xiaoyong Du, and Ji-Rong Wen. Persistent data sketching. In *Proceedings of the 2015 ACM SIGMOD international conference on Management of Data*, pages 795–810, 2015.

[18] Benwei Shi, Zhuoyue Zhao, Yanqing Peng, Feifei Li, and Jeff M Phillips. At-the-time and back-in-time persistent sketches. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1623–1636, 2021.

[19] Nick G. Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *J. ACM*, 2007.

[20] Mehdi Sharifzadeh and Cyrus Shahabi. The spatial skyline queries. In *Proceedings of the 32nd international conference on Very large data bases*, pages 751–762, 2006.

[21] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 467–478, 2003.

[22] Gowtham Atluri, Anuj Karpatne, and Vipin Kumar. Spatio-temporal data mining: A survey of problems and methods. *ACM Computing Surveys (CSUR)*, 51(4):1–41, 2018.

[23] Marc Gyssens and Laks VS Lakshmanan. A foundation for multi-dimensional databases. In *VLDB*, volume 97, pages 106–115. Citeseer, 1997.

[24] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. Learning multi-dimensional indexes. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 985–1000, 2020.

[25] Jelena Lukić, Miloš Radenković, Marijana Despotović-Zrakić, Aleksandra Labus, and Zorica Bogdanović. A hybrid approach to building a multi-dimensional business intelligence system for electricity grid operators. *Utilities Policy*, 41:95–106, 2016.

[26] Jinbao Wang, Sai Wu, Hong Gao, Jianzhong Li, and Beng Chin Ooi. Indexing multi-dimensional data in a cloud system. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 591–602, 2010.

[27] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Statistical change detection for multi-dimensional data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 667–676, 2007.

[28] Katsiaryna Mirylenka, Graham Cormode, Themis Palpanas, and Divesh Srivastava. Conditional heavy hitters: detecting interesting correlations in data streams. *The VLDB Journal*, 24:395–414, 2015.

[29] Robert T. Schweller, Ashish Gupta, Elliot Parsons, and Yan Chen. Reversible sketches for efficient and accurate change detection over network data streams. In Alfio Lombardo and James F. Kurose, editors, *IMC 2004*. ACM, 2004.

[30] Yin Zhang, Sumeet Singh, Subhabrata Sen, Nick G. Duffield, and Carsten Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *IMC 2004*. ACM, 2004.

[31] Graham Cormode, Flip Korn, Shanmugavelayutham Muthukrishnan, and Divesh Srivastava. Finding hierarchical heavy hitters in data streams. In *Proceedings 2003 VLDB Conference*, pages 464–475. Elsevier, 2003.

[32] Graham Cormode, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 155–166, 2004.

[33] David MW Powers. Applications and explanations of zipf's law. In *New methods in language processing and computational natural language learning*, 1998.

[34] Xiangyang Gou, Long He, Yinda Zhang, Ke Wang, Xilai Liu, Tong Yang, Yi Wang, and Bin Cui. Sliding sketches: A framework using time zones for data stream processing in sliding windows. In *KDD '20*. ACM, 2020.

[35] The Criteo 1TB Click Logs dataset. https://ailab.criteo.com/download-criteo-1tb-click-logs-dataset/.

[36] The NBA dataset. https://www.kaggle.com/datasets/nathanlauga/nba-games/.

[37] Anonymized internet traces 2018. https://catalog.caida.org/details/dataset/passive_2018_pcap. Accessed: 2022-6-29.

[38] Source code related to Hyper-USS. https://github.com/HyperUSS/HyperUSS.

A PROOFS

In this section, we prove Theorem 5 in §4.3. First of all, we make some assumptions as follows.

- The total number of items inserted into Hyper-USS is U.
- For every item inserted, its L2 norm has an upper bound L.

Lemma 6. *During the insertion of Hyper-USS, for the updated bucket, the increment of L2 norm is no larger than the L2 norm of the incoming item. Furthermore, for a bucket which has been updated for k times, the L2 norm of the values of the bucket has an upper bound kL.*

Proof. Suppose the incoming item $(e, w_1, \cdots, w_n)$ updates the bucket $\mathcal{B}_t[h_t(e)]$ and $u_i = \mathcal{B}_t[h_t(e)].V[i]$.
If the key in the bucket matches e, the increment of L2 norm is,

$$\sqrt{\sum_{i=1}^{n}(u_i + w_i)^2} - \sqrt{\sum_{i=1}^{n} u_i^2} \leq \sqrt{\sum_{i=1}^{n} w_i^2}$$

Otherwise, recall that we have the winning probability as follows.

$$\mathcal{P} = \frac{\sqrt{\sum_{i=1}^{n} w_i^2}}{\sqrt{\sum_{i=1}^{n} w_i^2} + \sqrt{\sum_{i=1}^{n} u_i^2}}$$

If e wins the competition, the increment of L2 norm is,

$$\sqrt{\frac{\sum_{i=1}^{n} w_i^2}{\mathcal{P}^2}} - \sqrt{\sum_{i=1}^{n} u_i^2} = \sqrt{\sum_{i=1}^{n} w_i^2}$$

If the key in the bucket wins, the result L2 norm is,

$$\sqrt{\frac{\sum_{i=1}^{n} u_i^2}{(1-\mathcal{P})^2}} - \sqrt{\sum_{i=1}^{n} u_i^2} = \sqrt{\sum_{i=1}^{n} w_i^2}$$

Therefore, the increment of L2 norm is no larger than the L2 norm of the incoming item and it is obvious that after k updates, the L2 norm of the bucket has an upper bound kL. □

Lemma 7. *During the insertion of Hyper-USS, for an update to a bucket which has been updated for k times, the increment of total variance of items in the bucket has an upper bound $2kL^2$.*

Proof. The L2 norm of the bucket before the update has an upper bound,

$$\sqrt{\sum_{i=1}^{n} u_i^2} \leq kL$$

The L2 norm of the inserted item has an upper bound,

$$\sqrt{\sum_{i=1}^{n} w_i^2} \leq L$$

By Theorem 3, the increment of total variance is,

$$\sum_{i=1}^{n}\sum_{e} \Delta\left(S_i(e) - \widehat{S}_i(e)\right)^2 = 2 \cdot \sqrt{\sum_{i=1}^{n} w_i^2} \cdot \sqrt{\sum_{i=1}^{n} u_i^2} \leq 2kL^2$$

□

Lemma 8. *For a bucket which has been updated for M times, the variance of the estimated value $\hat{S}$ for an arbitrary key mapped to the certain bucket and an arbitrary attribute has an upper bound $(M+1)ML^2$.*

Proof. For key e and an arbitrary attribute j, the estimated value of Hyper-USS is $\hat{S}_j(e)$. By Lemma 7, the variance can be bounded by summing up all increments of variance for each update,

$$\begin{aligned}\operatorname{Var}\left[\hat{S}_j(e) \mid M\right] &\leq \sum_{e':h(e')=h(e)} \sum_{i=1}^{n} \operatorname{Var}\left[\hat{S}_i(e') \mid M\right] \\ &\leq \sum_{k=1}^{M} 2kL^2 \\ &= (M+1)ML^2\end{aligned}$$

□

Theorem 5. *In the basic version, the error of Hyper-USS's estimation on an arbitrary attribute of an arbitrary key can be bounded as follows, where U is the total number of inserted items and L is the upper bound of L2 norm for each inserted item.*

$$\Pr\left[|\hat{S}_j(e) - S_j(e)| \geqslant \epsilon\right] \leqslant \frac{4U^2L^2}{w^2\epsilon^2}$$

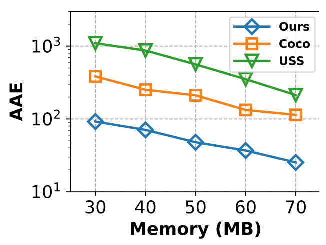
Proof. Recall that the estimated value by Hyper-USS is unbiased by Theorem 1. For each update, the probability that the inserted item is hashed to the same bucket is $p = \frac{1}{w}$. By Lemma 8, the variance of estimated value $\hat{S}_j(e)$ can by bounded,

$$\begin{aligned}\operatorname{Var}\left[\hat{S}_j(e)\right] &= \operatorname{E}\left[\operatorname{Var}\left[\hat{S}_j(e) \mid M\right]\right] + \operatorname{Var}\left[\operatorname{E}\left[\hat{S}_j(e) \mid M\right]\right] \\ &= \operatorname{E}\left[\operatorname{Var}\left[\hat{S}_j(e) \mid M\right]\right] \\ &\leq \sum_{M=0}^{U} \binom{U}{M} p^M (1-p)^{U-M} (M+1)ML^2 \\ &\leq 2UpL^2 + 3\sum_{M=2}^{U} \binom{U}{M} p^M (1-p)^{U-M} M(M-1)L^2 \\ &\leq 2UpL^2 + 3U^2p^2L^2 \sum_{M=2}^{U} \binom{U-2}{M-2} p^{M-2} (1-p)^{U-M} \\ &\leq \frac{4U^2L^2}{w^2}\end{aligned}$$

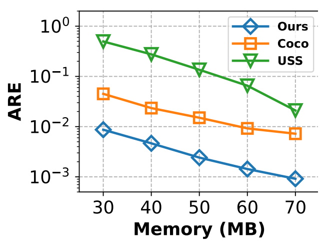
According to Chebyshev's inequality, we have

$$\Pr\left[|\hat{S}_j(e) - S_j(e)| \geqslant \epsilon\right] \leq \frac{\operatorname{Var}\left[\hat{S}_j(e)\right]}{\epsilon^2} \leq \frac{4U^2L^2}{w^2\epsilon^2}$$

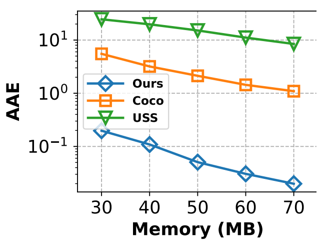
□



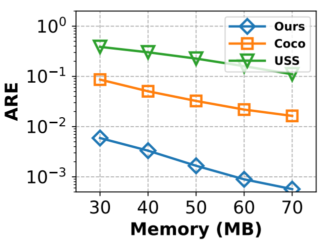
(a) Subset sum - AAE



(b) Subset sum - ARE

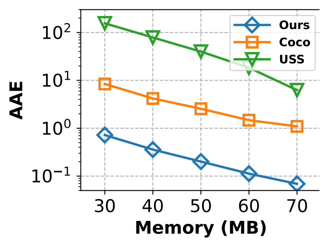


(c) Subset average - AAE

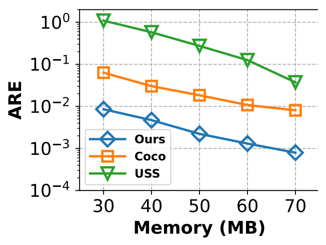


(d) Subset average - ARE

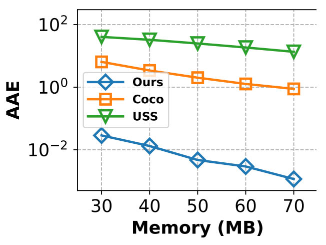
Figure 14: Experiments on the subset query on the NBA dataset.



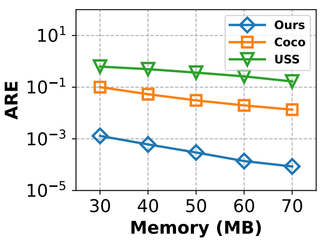
(a) Point sum - AAE



(b) Point sum - ARE

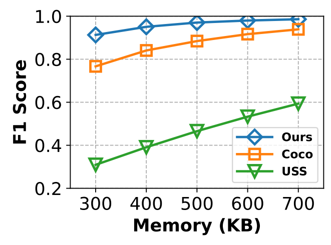


(c) Point average - AAE

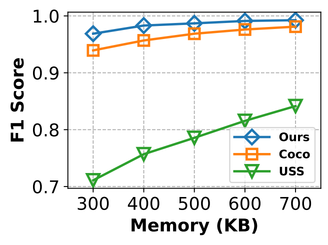


(d) Point average - ARE

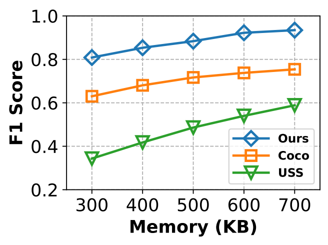
Figure 15: Experiments on the point query on the NBA dataset.



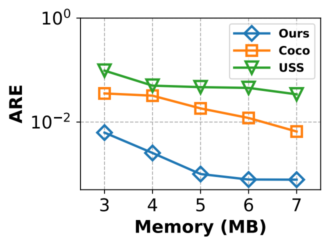
(a) Heavy hitter



(b) Heavy hitter subset



(c) Heavy change



(d) Distribution of values

Figure 16: Evaluations on other tasks.

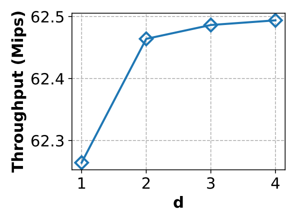


Figure 17: Experiments on the FPGA platfrom.

B EVALUATIONS ON THE NBA DATASET

Subset query (Figure 14): The experimental results show that, Hyper-USS is more accurate for subset query in the NBA dataset. Compared with USS and CocoSketch, for subset sum estimation, Hyper-USS reduces AAE by up to 4.49×, 11.71×, and reduces ARE by up to 6.22×, 56.14×, respectively. For subset average estimation, Hyper-USS reduces AAE by up to 41.99×, 299.00×, and reduces ARE by up to 19.56×, 136.35×, respectively.

Point query (Figure 15): The experimental results show that, Hyper-USS also achieves higher accuracy for point query on the NBA dataset. Compared with USS and CocoSketch, for point sum estimation, Hyper-USS reduces AAE by up to 12.70×, 198.28×, and reduces ARE by up to 8.28×, 123.56×, respectively. For point average estimation, Hyper-USS reduces AAE by up to 426.46×, 5280.64×, and reduces ARE by up to 103.96×, 1233.80×, respectively.

C EVALUATIONS ON OTHER TASKS

Experimental results (Figure 16): For heavy hitters and heavy hitter subsets, we conduct evaluation as described in §6.2.2. For heavy change, we divide the Criteo dataset into two equal parts as two windows and evaluate heavy change detection. For distribution of value, we evaluate the performance on the attribute that indicating whether or not the ad is clicked. The experimental results show that, Hyper-USS achieves higher accuracy than CocoSketch and USS in tasks including heavy hitter, heavy hitter subset, heavy change and distribution of values.

D EVALUATIONS ON THE FPGA PLATFORM

Experimental results (Figure 17): We fix the memory consumption to 44KB, and vary the d from 1 to 4. The results show that, the throughput drops as the d grows. When d ranges between 1 and 4, the throughput maintains higher than 62.26Mips, which is much higher than the throughput of the CPU implementation.