QSketch: An Efficient Sketch for Weighted Cardinality Estimation in Streams

Yiyan Qi*
International Digital Economy Academy (IDEA)
Shenzhen, China
qiyiyan@idea.edu.cn

Rundong Li*
MOE KLINNS Lab
Xi'an Jiaotong University
Xi'an, China
xjtulirundong@stu.xjtu.edu.cn

Pinghui Wang‡
MOE KLINNS Lab
Xi'an Jiaotong University
Xi'an, China
phwang@xjtu.edu.cn

Yufang Sun
MOE KLINNS Lab
Xi'an Jiaotong University
Xi'an, China
sunyufang00@stu.xjtu.edu.cn

Rui Xing
MOE KLINNS Lab
Xi'an Jiaotong University
Xi'an, China
xingrui128719@163.com

arXiv:2406.19143v1 [cs.DB] 27 Jun 2024

ABSTRACT

Estimating cardinality, i.e., the number of distinct elements, of a data stream is a fundamental problem in areas like databases, computer networks, and information retrieval. This study delves into a broader scenario where each element carries a positive weight. Unlike traditional cardinality estimation, limited research exists on weighted cardinality, with current methods requiring substantial memory and computational resources, challenging for devices with limited capabilities and real-time applications like anomaly detection. To address these issues, we propose QSketch, a memory-efficient sketch method for estimating weighted cardinality in streams. QSketch uses a quantization technique to condense continuous variables into a compact set of integer variables, with each variable requiring only 8 bits, making it 8 times smaller than previous methods. Furthermore, we leverage dynamic properties during QSketch generation to significantly enhance estimation accuracy and achieve a lower time complexity of $O(1)$ for updating estimations upon encountering a new element. Experimental results on synthetic and real-world datasets show that QSketch is approximately 30% more accurate and two orders of magnitude faster than the state-of-the-art, using only 1/8 of the memory.

CCS CONCEPTS

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

KEYWORDS

Streaming algorithms, Sketch, Weighted Cardinality Estimation

* Equal Contribution.
‡ Corresponding Author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

KDD '24, August 25–29, 2024, Barcelona, Spain

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671695

ACM Reference Format:
Yiyan Qi*, Rundong Li*, Pinghui Wang‡, Yufang Sun, and Rui Xing. 2024. QSketch: An Efficient Sketch for Weighted Cardinality Estimation in Streams. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3671695

1 INTRODUCTION

Real-world systems generate data in a streaming fashion. Examples range from financial transactions to Internet of Things (IoT) data, network traffic, call logs, trajectory logs, etc. Computing the cardinality, i.e., the number of distinct elements, of such a stream is fundamental in research areas like databases, machine learning, and information retrieval. For example, online games and mobile apps usually use *daily active users* (DAU), i.e., the number of distinct active users within a day, as a metric to measure the level of engagement. Other examples include network security monitoring [14] and connectivity analysis in the Internet graph [32].

Due to the unknown or even unlimited size and the high-speed nature of these data streams, it is infeasible to collect the entire data when the computation and memory resources of data collection devices (e.g., network routers) are limited. To solve this challenge, considerable attention has been paid to designing fast and memory-efficient cardinality estimating algorithms via sketching techniques [13, 17, 20, 40]. They build a compact data summary (i.e., sketch) on the fly and then estimate the cardinality from the generated sketch. The above cardinality computing problem can be generalized to a weighted scenario, where each element e in the data stream is associated with a positive weight $w \in \mathbb{R}_{+}$. In this new scenario, the goal is to compute the total sum of weights for all distinct elements, i.e., weighted cardinality. The weighted cardinality has various applications, including 1) In database systems, an example is a SQL query like "SELECT DISTINCT * FROM TABLE". In addition to the query's cardinality, understanding the total size of the resultant set aids in optimizing performance and managing resources [30, 31]. Here, the weighted cardinality represents the total size (in bytes) of the query result, calculated as the cumulative size of all distinct records, weighted by their row size. 2) In a voting system, each voter may have a weight based on their expertise and we need it to figure out the final voting result. 3) In an app or website, users with

more activity might be assigned higher weights. This metric allows for a more nuanced understanding of the app other than Daily Active User (DAU). Weighted cardinality computation is important in scenarios where individual contributions vary in importance.

Despite the plenty of works for estimating the regular cardinality, little attention has been paid to the problem of **W**eighted **C**ardinality **E**stimation (WCE). Lemiesz [26] conducts a formal study on the WCE problem. The proposed method maps each element in the data stream into m exponential variables concerning the element's weight. To guarantee estimation accuracy, m is set to hundreds or thousands, making it infeasible to deal with real-time streams. Zhang et al. [45] proposed a method *FastGM* to decrease the update time complexity of Lemiesz's method. Instead of generating m variables independently, FastGM generates those exponential variables in ascending order and early stops the generation when the value is greater than the maximal value stored in the current registers. A recent method FastExp Sketch [27] shares the same idea with FastGM. The above three methods use 64-bit floating-point registers to store these exponential variables. When a large value of m is employed for improved accuracy, it becomes memory-intensive for devices with limited computational and storage resources. Additionally, they require $O(m)$ operations to estimate weighted cardinality, making them computationally expensive when aiming to provide anytime-available estimation for real-time applications.

We develop a memory-efficient sketch, *QSketch*, to estimate the weighted cardinality of distinct elements in a data stream. QSketch generates m independent exponential variables for each incoming element in descending order. This process is employed to update the m **integer** registers, and an early termination occurs when a generated variable is smaller than the values in all registers. QSketch employs a novel mapping strategy that transforms continuous exponential variables into discrete variables, using a small set of integer registers to represent data streams with varying weighted cardinalities. Consequently, each register in our QSketch requires no more than 5 bits, making it up to 13× smaller than both Lemiesz's method and FastGM. To reduce time cost and estimation error, we propose an extension of QSketch, *QSketch-Dyn*, to monitor the weighted cardinality on the fly. QSketch-Dyn shares the same data structure as QSketch but only needs to compute one variable for each element. It utilizes the dynamic property of our QSketch to reduce estimation error significantly. We summarize our main contributions as:

- We propose a memory-efficient sketch method QSketch to estimate the weighted cardinality of distinct elements in a data stream. QSketch employs a novel mapping strategy that transforms continuous exponential variables into discrete variables, using a small set of integer registers to represent data streams with varying weighted cardinalities.
- We also present QSketch-Dyn, an advanced variant of QSketch, which leverages the inherent dynamic nature of QSketch, enabling real-time tracking of weighted cardinality.
- We conduct experiments on both synthetic and real-world datasets. The experimental results demonstrate that our new sketching method achieves approximately 30% more accurate and is two orders of magnitude faster than the state-of-the-art while requiring only 1/8 of the memory usage.

Algorithm 1: Pseudo-code of Lemiesz's method.

input : stream Π, m.
output: sketch R.

```
1 R ← [+∞, ..., +∞];
2 foreach (x, w) ∈ Π do
3     s for j = 1, ..., m do
4         r_j ← -ln h_j(x) / w ;
5         R[j] ← min(R[j], r_j);
```

The rest of this paper is organized as follows. Section 2 introduces the problem. Section 3 briefly discusses preliminaries. Section 4 presents our method QSketch and QSketch-Dyn. The performance evaluation and testing results are presented in Section 5. Section 6 summarizes related work. Concluding remarks then follow.

2 PROBLEM FORMULATION

We first introduce some notations. Let $\Pi = \mathrm{e}^{(1)} \cdots \mathrm{e}^{(t)} \cdots$ denote a data stream, where an element $\mathrm{e}^{(t)}$ arriving at time t corresponds to one of the elements $x_1, \ldots, x_n$ and each $x_i, 1 \le i \le n$ has a positive weight $w_i > 0$. Note that an element e may appear multiple times in the stream. Denote $N_\Pi^{(t)}$ as the set of distinct elements that occurred in stream Π before and including time t. Then, the weighted cardinality of stream Π at t is defined as

$$C^{(t)} = \sum_{x_i \in N_\Pi^{(t)}} w_i. \tag{1}$$

This paper aims to develop a sketch method to estimate the weighted cardinality $C^{(t)}$ accurately and efficiently. We omit the superscript (t) when no confusion arises.

3 PRELIMINARIES

In this section, we introduce two existing methods to estimate weighted cardinality and discuss their shortcomings.

3.1 Existing Methods For WCE

• **Lemiesz's Method** [26] builds a sketch consisting of m registers $R[1], \ldots, R[m]$. Typically, m is set to be thousands to guarantee the desired accuracy. All m registers are initialized to $+\infty$. For each $j \in \{1, \ldots, m\}$, let $R^{(t)}[j]$ denote the value of $R[j]$ at time t. For each $\mathrm{e}^{(t)}$ arriving at time t, Lemiesz's method maps it into all m registers independently and each register is updated as

$$R^{(t)}[j] \leftarrow \min(R^{(t-1)}[j], r_j(\mathrm{e}^{(t)})).$$

Without loss of generality, we let $\mathrm{e}^{(t)} = x_i$ with weight w_i, $1 \le i \le n$. In the above equation, $r_j(\mathrm{e}^{(t)})$ is defined as

$$r_j(\mathrm{e}^{(t)}) = -\frac{\ln h_j(x_i)}{w_i},$$

where $h_j(x)$ is a hash function that maps x to $(0, 1)$ uniformly, i.e., $h_j(x) \sim \text{Uniform}(0, 1)$, $1 \le j \le m$. The pseudo-code of Lemiesz's method is shown in Algorithm 1. We note that $r_j(\mathrm{e}^{(t)})$ follows an exponential distribution $\text{EXP}(w^{(t)})$ and $R^{(t)}[j] = \min_{x_i \in N_\Pi^{(t)}} r_j(x_i)$

follows an exponential distribution $\text{EXP}(C^{(t)})$, in which $C^{(t)}$ is the weighted cardinality at time t. Thus, the summation variable $G_m = \sum_{j=1}^{m} R^{(t)}[j]$ follows a gamma distribution $G_m \sim \Gamma(m, C^{(t)})$. According to [41], the inverse of the summation variable G_m, i.e., $1/G_m$, has the inverse gamma distribution $1/G_m \sim \Gamma^{-1}(m, C^{(t)})$, and we have

$$\mathbb{E}[1/G_m] = \frac{C^{(t)}}{m-1}, \quad \mathbb{V}\text{ar}[1/G_m] = \frac{(C^{(t)})^2}{(m-2)(m-1)^2},$$

where the first equation holds for $m \geq 2$ and second for $m \geq 3$. Then at time t, Lemiesz gives an unbiased estimator of $C^{(t)}$ as

$$\hat{C}^{(t)} = \frac{m-1}{\sum_{j=1}^{m} R^{(t)}[j]}. \tag{2}$$

The above estimator's variance is computed as

$$\mathbb{V}\text{ar}[\hat{C}^{(t)}/C^{(t)}] = \frac{1}{m-2}.$$

• **FastGM.** Lemiesz's method needs $O(m)$ time to update an element In practice, m is usually thousands to achieve the expected accuracy and is infeasible for high-speed streams. To solve this problem, Zhang et al. [45] proposed FastGM, reducing the time complexity from $O(m)$ to $O(1)$. FastGM shares the same sketch structure as Lemiesz's method. The difference is that m random variables in FastGM $-\frac{\ln h_1(x_i)}{w_i}, \ldots, -\frac{\ln h_m(x_i)}{w_i}$ are generated in ascending order as $\left(-\frac{\ln h_{\pi_1}(x_i)}{w_i}, \pi_1\right), \ldots, \left(-\frac{\ln h_{\pi_m}(x_i)}{w_i}, \pi_m\right)$, where $-\frac{\ln h_{\pi_1}(x_i)}{w_i} < \cdots < -\frac{\ln h_{\pi_m}(x_i)}{w_i}$ and $\pi_1, \ldots, \pi_m$ is a random permutation of integers $1, \ldots, m$. Once the current obtained random variable $-\frac{\ln h_{j_1}(x_i)}{w_i}$ is larger than all values in registers $R^{(t)}[1], \ldots, R^{(t)}[m]$, there is no need to generate the following random variables because they have no chance to change the sketch. In detail, for each $\text{e}^{(t)} = x_i$, $1 \leq i \leq n$, FastGM generates the first exponential variable as

$$r_{\pi_1}(\text{e}^{(t)}) = -\frac{1}{m} \cdot \frac{\ln h_{\pi_1}(x_i)}{w_i}, \tag{3}$$

and the following exponential variables $r_{\pi_j}(\text{e}^{(t)}), 2 \leq j \leq m$ are generated in ascending order as

$$r_{\pi_j}(\text{e}^{(t)}) = r_{\pi_{j-1}}(\text{e}^{(t)}) - \frac{1}{m-j+1} \cdot \frac{\ln h_{\pi_j}(x_i)}{w_i}. \tag{4}$$

After generating a hash value $r_{\pi_j}(\text{e}^{(t)})$, $j \in \{1, \ldots, m\}$, FastGM uses the Fisher-Yates shuffle [15] to find its position $\pi_j \in \{1, \ldots, m\}$. Specially, let $(\pi_1, \ldots, \pi_m)$ be initialized to $(1, \ldots, m)$. To obtain the position of the j-th smallest hash value, FastGM randomly chooses a position $k \in \{j, j+1, \ldots, m\}$, swaps π_k and π_j, and updates $R^{(t)}[\pi_j]$ with $r_{\pi_j}(\text{e}^{(t)})$. FastGM uses an extra register r^* to perform an early stop to record the maximal values among m registers. Register values in FastGM follow the same distribution as LM. They share the same weighted cardinality estimator and the same estimation errors. Besides, we find that a recent method FastExpSketch [27] shares the same idea with FastGM.

3.2 Limitations of Existing Methods

• **Memory-consuming.** Lemiesz's method and FastGM use 32-bit or 64-bit floating-point registers to store hash variables. They require $32m$ or $64m$ bits of memory to store m hash variables for a stream Π. When we need a large m for better accuracy or there are many different streams, it is memory-intensive for devices (e.g. IoT devices or routers) with limited computational and storage resources. Therefore, reducing the number of bits for each register is practical, saving storage space and improving the computational efficiency for sketch operations [26, 29].

• **Time-consuming.** Lemiesz's method requires updating each register for element insertion, incurring an $O(m)$ time cost. While FastGM improves upon this by ordering register updates, it still demands $O(m)$ time in the worst-case scenario when element weights grow over time. Furthermore, the time complexity for the estimation process in both methods is quantified as $O(m)$. Such a computational demand poses challenges for providing consistent, real-time estimations in applications that require immediate data availability.

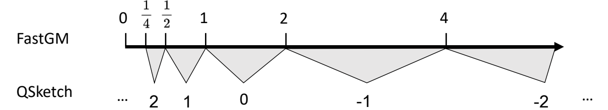


Figure 1: Basic idea of QSketch

4 OUR METHOD

In this section, we first introduce a compact sketch QSketch (**Q**uantization **Sketch**) which utilizes the quantization technique (i.e., mapping continuous infinite values to a small set of discrete values). We design a novel estimator to estimate weighted cardinality. Then, we exploit the dynamic properties of the register arrays over time to significantly improve the estimation accuracy and reduce the time cost to monitor the weighted cardinality on the fly.

4.1 Basic Idea

QSketch reduces the size of 32-bit or 64-bit floating-point registers to a smaller bit size (5 or 6 bits) through quantization. This process transforms an infinite range of continuous values into a limited set of discrete values. Figure 1 illustrates how QSketch's hash value mappings compare to those in FastGM and Lemiesz's method.

4.2 QSketch

Data structure and update procedure. Denote R as the sketch with m registers $R[1], \ldots, R[m]$, and $h_1, \ldots, h_m$ as m independent hash functions, each of them mapping x to a random value in range $(0, 1)$ uniformly, i.e., $h_j(x) \sim \text{Uniform}(0, 1)$, $1 \leq j \leq k$. When inserting e which is associated with i x and weight w, we generate m variables $y_1(\text{e}), \ldots, y_m(\text{e})$ as

$$y_j(\text{e}) = \lfloor -\log_2(r_j(\text{e})) \rfloor, \tag{5}$$

where $r_j(\text{e}) = -\frac{\ln h_j(x)}{w}$ is an exponential random variable and $\lfloor \cdot \rfloor$ is the round-down operation. Then QSketch updates as

$$R[j] \leftarrow \max(R[j], y_j(\text{e})). \tag{6}$$

Following [45], we generate m variables $r_{\pi_1}(\text{e}), \ldots, r_{\pi_m}(\text{e})$ in an ascending order, in which $(\pi_1, \ldots, \pi_m)$ is a random permutation of $(1, \ldots, m)$. As a result, $y_{\pi_1}(\text{e}), \ldots, y_{\pi_m}(\text{e})$ are generated in a descending order. The update procedure is shown in Algorithm 2. We use the Fisher-Yates shuffle [15] (Line 11-12 in Algorithm 2) to quickly find the position to be updated and use j^* to record

Algorithm 2: Update procedure of QSketch.

```
input  : stream Π, sketch size m, register size b.
output : sketch R.
1  r_min ← −2^{b−1} + 1; r_max ← 2^{b−1} − 1;
2  U ← [r_min, ..., r_min]; j* ← 1;
3  foreach e ∈ Π do
       /* element x with weight w.                    */
4      [π_1, ..., π_m] ← [1, ..., m] ;
5      r ← 0;
6      for j = 1, ..., m do
7          r ← r − ln h_j(x) / (w(m−j+1)) ;
8          y ← ⌊−log_2(r)⌋ ;
9          if y ≤ R[j*] then
               /* early stop                           */
10             break ;
11         k ← RandInt (j, m) ;
12         Swap (π_k, π_j);
13         if y > R[π_l] then
14             R[π_l] = min(max(y, r_min), r_max);
15             if π_l = j* then
16                 j* ← arg min_{j=1,...,m} R[j] ;
```

the register's index that records the sketch's minimum values. A brief introduction to the Fisher-Yates shuffle is in Appendix A.1. When the generated variable is smaller than $R[j^*]$, we early stop the generation procedure.

Weighted Cardinality Estimation. Direct use of the estimator in Equation (2) yields a large estimation error since the quantization leads to a loss of precision. Instead, we design a new estimator with the help of Maximum Likelihood Estimation (MLE). Before that, we first derive the probability distribution of a single register $R[j], 1 \leq j \leq m$. It has been proved that a register value R (without quantization) follows an exponential distribution $\text{EXP}(C_\Pi)$ [26, 45], in which C_Π is the weighted cardinality of stream Π. In QSketch, according to Equation (5), we quantize the continuous register value to discrete values. Particularly, continuous register values in the range $(2^{-(r+1)}, 2^{-r}]$ will be compressed into a discrete value r. Then the probability distribution of a single register value $R[i]$ is

$$Pr(R[j] = r|C_\Pi) = \int_{2^{-(r+1)}}^{2^{-r}} C_\Pi e^{-C_\Pi x} dx = e^{-C_\Pi \cdot 2^{-(r+1)}} - e^{-C_\Pi \cdot 2^{-r}}. \tag{7}$$

Given the specific register values $R[1], \ldots, R[m]$, we compute the likelihood function for C_Π is

$$L(C_\Pi) = \prod_{j=1}^{m} Pr(R[j]|C_\Pi) = \prod_{j=1}^{m} \left(e^{-C_\Pi \cdot 2^{-(R[j]+1)}} - e^{-C_\Pi \cdot 2^{-R[j]}} \right), \tag{8}$$

and the derivative of the likelihood function's logarithm is:

$$\frac{d(\ln L(C_\Pi))}{dC_\Pi} = \sum_{j=1}^{m} 2^{-(R[j]+1)} \cdot \frac{2 - e^{C_\Pi \cdot 2^{-(R[j]+1)}}}{e^{C_\Pi \cdot 2^{-(R[j]+1)}} - 1}. \tag{9}$$

To compute the MLE of C_Π, let the above formula equal 0. Unfortunately, this equation is too complicated to be solved directly. So we use the Newton-Raphson method to obtain $\hat{C}_\Pi$. Specially, let

$$f(C_\Pi) = \sum_{j=1}^{m} 2^{-(R[j]+1)} \cdot \frac{2 - e^{C_\Pi \cdot 2^{-(R[j]+1)}}}{e^{C_\Pi \cdot 2^{-(R[j]+1)}} - 1}, \tag{10}$$

The Newton-Raphson method [5] starts from an initial estimation $\hat{C}_\Pi^{(0)}$ and then repeats the following steps:

$$\hat{C}_\Pi^{(l+1)} \leftarrow \hat{C}_\Pi^{(l)} - \frac{f(\hat{C}_\Pi^{(l)})}{f'(\hat{C}_\Pi^{(l)})}, \tag{11}$$

until $\hat{C}_\Pi$ converges, where $f'(\hat{C}_\Pi^{(l)})$ is the derivative of $f(C_\Pi)$ at point $C_\Pi = \hat{C}_\Pi^{(l)}$. To start the iteration, we initialize $\hat{C}_\Pi^{(0)}$ as

$$\hat{C}_\Pi^{(0)} = \frac{m-1}{\sum_{1 \leq j \leq m} 2^{-R[j]}}.$$

Since the register values in QSketch are the quantization of register values in FastGM, $\hat{C}_\Pi^{(0)}$ can be viewed as an approximation of $\hat{C}_\Pi$, which is reasonable as an initial value to guarantee convergence.

The above MLE-based estimator provides an asymptotically unbiased estimation. The approximate variance of the above estimator is based on the Cramér-Rao bound [12]. Specifically, we have $\text{Var}[\hat{C}] \approx \frac{1}{I_\Pi(\hat{C})}$, where $I_\Pi(\hat{C})$ is the observed fisher information given stream data Π, i.e., the negative of the second derivative of the log-likelihood function at $\hat{C}$. Formally, we have

$$\text{Var}[\hat{C}] \approx -\frac{1}{f'(\hat{C}_\Pi^{(l)})}.$$

Through the quantization, all possible values of $y(\text{e})$ are integers, i.e., $y(\text{e}) \in \mathbb{Z}$. In practice, we notice that most values of $y(\text{e})$ are concentrated in a small range and we can truncate these generated variables by $y'(\text{e}) = \min(\max(y(\text{e}), r_{\min}), r_{\max})$. As a result, adjusting the probability distribution of the truncated value in each register is necessary, as shown in Equation (7).

$$Pr(R[j]|C_\Pi) = \begin{cases} e^{-C_\Pi \cdot 2^{-(r_{\min}+1)}}, & R[j] \leq r_{\min}; \\ 1 - e^{-C_\Pi \cdot 2^{-r_{\max}}}, & R[j] \geq r_{\max}; \\ e^{-C_\Pi \cdot 2^{-(R[j]+1)}} - e^{-C_\Pi \cdot 2^{-R[j]}}, & \text{otherwise.} \end{cases}$$

By substituting the above probability to Equation (8), we get the weighted cardinality estimator under truncated values. Note that the estimator only fails to give an unbiased estimation when all register values equal $r_{\min}$ or $r_{\max}$ as the likelihood function $L(C_\Pi)$ becomes monotonous without an extremum. Fortunately, in the following theorem, we show that by properly setting $r_{\min}$ and $r_{\max}$, the failure probability is extremely low.

Theorem 1. *Let $0 < \varepsilon \ll 1$ be a small positive value. Given a sketch of m registers with minimal value r_{min} and maximal value r_{max}, when $-2^{(r_{min}+1)} \cdot \ln \epsilon < C_\Pi < -2^{r_{max}} \ln(1-\epsilon)$, the register*

values are not in the discrete set $\{r_{min}, ..., r_{max}\}$ *within a maximum probability of* 2ϵ.

The proof is in Appendix A.2. When $r_{\min} = -127$, $r_{\max} = 127$, and $\varepsilon = 0.001$, each register value at time t is not tuncatated with a probability of more than 0.998 when $8.1 \times 10^{-38} \le C_\Pi \le 3.4 \times 10^{35}$. In this case, we need just 8 bits to store the register values.

4.3 QSketch-Dyn

QSketch has the same expected time complexity of $O(m \cdot \ln m + n)$ as FastGM [33, 45]. However, it still suffers from the worst time complexity $O(m \cdot n)$ under scenarios in which the weights of the elements increase as time progresses. In addition, we have to solve the MLE problem whenever figuring out the latest weighted cardinality, which costs for real-time estimation. To improve the estimation efficiency, we utilize the dynamic property of the sketch and propose QSketch-Dyn to keep track of the weighted cardinality on the fly.

Data structure and update procedure. QSketch-Dyn shares the same data structure, i.e., a bit array of size m, and the same set of hash functions $h_1, \ldots, h_m$ as Lemiesz's method [26]. The main differences are: 1) QSketch-Dyn introduces another hash function $g(x)$, which uniformly maps x to some integer in set $\{1, \ldots, m\}$ at random. 2) QSketch-Dyn maintains a tabular T to record the frequency of values in the sketch. Specifically, the tabular T consists of 2^b counters, in which b is the number of bits used by a register.

Next, we introduce how to update an element $\mathrm{e}^{(t)}$ corresponding to x and w. Instead of updating multiple register values as in other methods, QSketch-Dyn first randomly chooses one register $R[j]$ with $j = g(x)$, $1 \le j \le m$ from the sketch, computes its quantized hash value $y_j(\mathrm{e}) = \lfloor -\log_2(r_j(\mathrm{e})) \rfloor$ (Equation 5), and update the register value $R[j] \leftarrow \max(R[j], y_j(\mathrm{e}))$ (Equation 6). Once the register value $R[j]$ changes, we then update the tabular T as $T[R[j]] \leftarrow T[R[j]] - 1$ and $T[y_j(\mathrm{e})] \leftarrow T[y_j(\mathrm{e})] + 1$.

Weighted Cardinality Estimation. Denote $q_R^{(t)}$ as the probability of $\mathrm{e}^{(t)}$ changing a register among $R[1], \ldots, R[m]$ at time t,

$$\begin{aligned} q_R^{(t)} &\triangleq \sum_{1 \le j \le m} Pr(y > R[j] \wedge g(x) = j \mid R[j]) \\ &= \sum_{1 \le j \le m} Pr(y \ge R[j] + 1 \mid R[j]) \cdot Pr(g(x) = j) \\ &= \frac{1}{m} \cdot \sum_{1 \le j \le m} \int_0^{2^{-(R[j]+1)}} w \cdot e^{-wx} dx \\ &= 1 - \frac{1}{m} \sum_{1 \le j \le m} e^{-w \cdot (2^{-(R[j]+1)})}. \end{aligned}$$

Let $\hat{C}_\Pi^{(t)}$ denote the weighted cardinality estimate of stream Π at time t. When element e arrives at time t, we update the weighted cardinality estimate as

$$\hat{C}_\Pi^{(t)} \leftarrow \hat{C}_\Pi^{(t-1)} + \frac{\mathbf{1}(R[j]^{(t)} \neq R[j]^{(t-1)})}{q_R^{(t)}} \cdot w, \tag{12}$$

in which $\mathbf{1}(R[j]^{(t)} \neq R[j]^{(t-1)})$ is an indicator variable which equals to 1 if element e changes the register value $R[j]$ (remind that $j = g(x)$) and equals to 0 otherwise. In later analysis, we will prove that $\hat{C}_\Pi^{(t)}$ is an unbiased estimation of $C_\Pi^{(t)}$.

Algorithm 3: Update procedure of QSketch-Dyn.

input : stream Π, sketch size m, register size b.
output: Estimated weighted cardinality $\hat{C}$.

```
1  r ← 0; q_R ← 0; Ĉ ← 0;
2  R ← [r_min, ..., r_min]; T ← [0, 0, ..., 0] ;
3  foreach e ∈ Π do
       /* corresponding to element x with weight w. */
4      j ← RandInt (1, m);
5      r ← − ln h_j(x) / w ;
6      y ← ⌊−log_2(r)⌋ ;
7      if y > R[j] then
8          if T[R[j] − r_min] > 0 then
9              T[R[j] − r_min] ← T[R[j] − r_min] − 1 ;
10             T[y − r_min] ← T[y − r_min] + 1;
11         else
12             T[y − r_min] ← T[y − r_min] + 1;
13         R[j] ← min(y, r_max);
14         q ← 0;
15         for k = 0, ..., 2^b − 1 do
16             q ← q + T[k] · e^(−w·2^(−(k+r_min+1)));
17         q_R = 1 − q/m;
18         Ĉ ← Ĉ + w/q_R;
```

The computation of probability $q_R^{(t)}$ needs summation over all m registers, which is time-consuming when m is set to a large value. To save time, we additionally maintain a tabular T recording the histogram of register values as mentioned above, where $T[R[j]]$ tracks the count of value $R[j]$ in the current sketch. As discussed previously, each register occupies b bits and the number of different values is at most 2^b. Then, $q_R^{(t)}$ is expressed as

$$q_R^{(t)} = 1 - \frac{1}{m} \sum_{1 \le j \le 2^b} T[R[j]] \cdot e^{-w \cdot (2^{-(R[j]+1)})}.$$

We summarize the pseudo-code of the update and estimation procedure of QSketch-Dyn in Algorithm 3.

Complexity Analysis. The time complexity of updating an element for QSketch-Dyn is $O(1)$ since it only chooses one register to update its value. Then it costs $O(2^b)$ time for QSketch-Dyn to compute $q_R^{(t)}$. Considering that b is small, this part costs little time. Finally, QSketch-Dyn tracks the estimated cardinality over time, and it costs no time for estimation compared with QSketch. For space complexity, QSketch uses m registers, in which each register occupies b bits. Besides, QSketch-Dyn maintains a tabular T with 2^b counters. Since there are m registers in the sketch, each counter of tabular T occupies at most $\log_2(m)$ bits. Therefore, the total space complexity is $m \cdot b + 2^b \cdot \log_2(m)$.

Error Analysis. For the estimated weighted cardinality $\hat{C}_\Pi^{(t)}$ from QSketch-Dyn, we first prove that our estimator is unbiased and then derive the variance of $\hat{C}_\Pi^{(t)}$.

Table 1: Statistics of real-world datasets.

Dataset	#Documents	Vector Size
Real-sim [39]	72,309	20,958
Rcv1 [28]	20,242	47,236
Webspam [36]	350,000	16,609,143
News20 [24]	19,996	1,355,191
Libimseti [25]	220,970	220,970

Theorem 2. *The expectation and variance of* $\hat{C}_{\Pi}^{(t)}$ *are*

$$\mathbb{E}[\hat{C}_{\Pi}^{(t)}] = C_{\Pi}^{(t)},$$

$$Var[\hat{C}_{\Pi}^{(t)}] = \sum_{i \in T_s^{(t)}} (w^{(i)})^2 \mathbb{E}[\frac{1 - q_R^{(i)}}{q_R^{(i)}}],$$

where $T_s^{(t)}$ *is the set of timestamps that each element appears in the stream for the first time.*

Proof. See Appendix A.3 □

5 EVALUATION

All algorithms are implemented in C++ and run on a processor with a Quad-Core Intel(R) Xeon(R) CPU E3-1226 v3 CPU 3.30GHz processor. Our source code is available [4].

5.1 Datasets

We conduct experiments on both synthetic and real-world datasets.

• **Synthetic Datasets.** We generate synthetic datasets with the following distributions: Uniform distribution $U(0,1)$, Gauss distribution $N(1,0.1)$, and Gamma distribution $\gamma(1,2)$ For each type of distribution, we generate datasets with different sizes of elements, respectively. The name of the dataset is represented as "distribution-#elements". For example, Uniform-1k represents the dataset with $1,000$ elements, and the weight of each element follows the Uniform distribution $N(0,1)$. Each dataset is considered a single stream.

• **Real-world Datasets.** We use the following real-world datasets: **Twitter** [1], **Real-sim** [39], **Rcv1** [28], **Webspam** [36], **News20** [24] **Libimseti** [25]. Twitter [1] is a dataset of "following" relationships between Twitter users. The above two datasets are treated as single-stream datasets. Real-sim [39], Rcv1 [28], Webspam [36] and News20 [24] are datasets of web documents from different resources, where each vector represents a document and each entry in the vector refers to the TF-IDF score of a specific word for the document. Libimseti [25] is a dataset of ratings of different users, where each vector refers to a user and each entry records the user's rating. These six datasets are considered multi-stream datasets, and each vector within the dataset is a single stream of elements with a weight. The details of these datasets are described in Table 1.

5.2 Baselines

We compare QSketch and QSketch-Dyn, with state-of-the-art methods, Lemiesz's method [26] (represented as **LM**), FastGM [45] and FastExp Sketch [27]. All baseline methods maintain a sketch with m 64-bit registers. QSketch and QSketch-Dyn are truncated with $r_{\min} = -127$ and $r_{\max} = 127$, i.e., they all use 8-bit integer registers by default. We assign each algorithm the same number of registers, which means QSketch and QSketch-Dyn use about 1/8 of the memory space of baseline methods. Following [3], we use a 32-bit word to hold multiple short-bit registers. For example, with each register set to 8 bits, a 32-bit word can hold $\lfloor \frac{32}{8} \rfloor = 4$ registers.

5.3 Metrics

• **Accuracy.** We use *Relative Root Mean Square Error* (RRMSE) and *Average Absolute Relative Error* (AARE) to evaluate the estimation accuracy on single-stream datasets and multi-stream datasets, respectively. In detail, the RRMSE of estimation $\hat{C}$ is defined as

$$RRMSE(\hat{C}) = \frac{\sqrt{\mathbb{E}[(\hat{C} - C)^2]}}{C},$$

and the AARE is defined as

$$AARE = \frac{1}{n} \sum_{i=1}^{n} \frac{|\hat{C}_i - C_i|}{|C_i|}.$$

• **Efficiency.** We use **Throughput** (Million updates per second, Mops) to evaluate the update speed for incoming elements, and **Esimation time** to assess the time taken to calculate the weighted cardinality from the sketch. All experimental results are empirically computed from 100 independent runs by default.

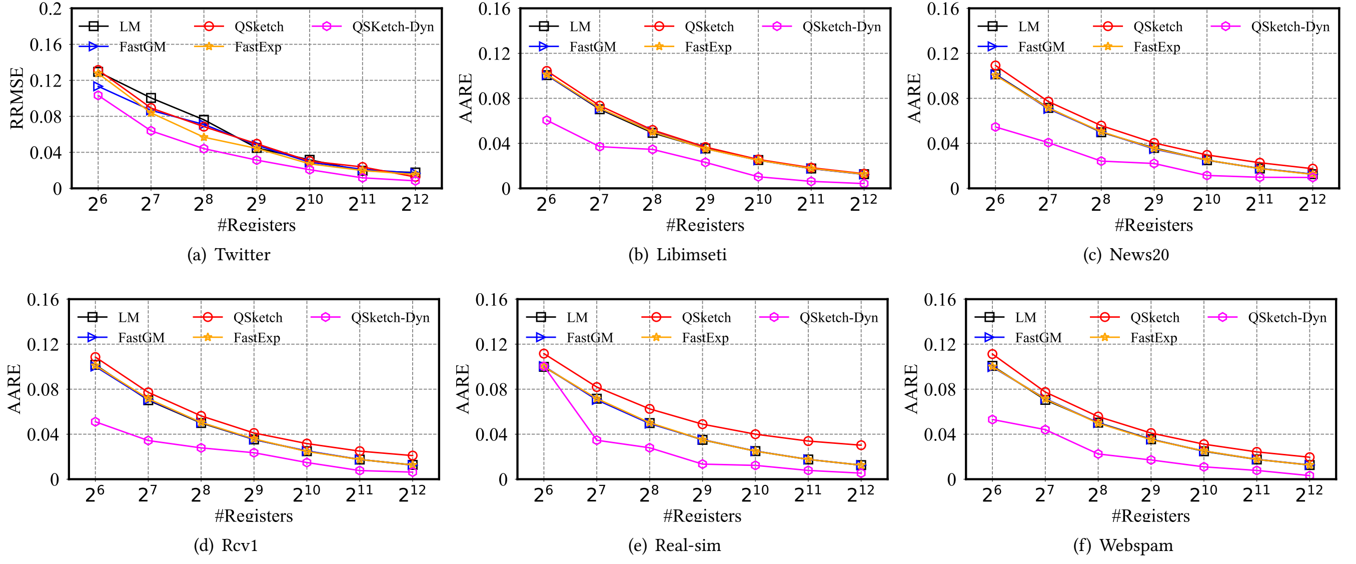
5.4 Accuracy Analysis

5.4.1 Results on Real-World Datasets. Figure 2 shows the results on accuracy concerning the number of registers in a sketch on real-world datasets. Specially, we vary the number of registers in each sketch $m \in \{2^6, 2^7, 2^8, 2^9, 2^{10}, 2^{11}, 2^{12}\}$. For dataset Twitter and the other 3 document datasets, we evaluate RRMSE and AARE w.r.t. the number of registers, respectively. Notably, QSketch demonstrates comparable performance to other baseline methods across all datasets. Conversely, QSketch-Dyn outperforms its competitors, leveraging the dynamic nature of the sketch. For example, on the dataset Twitter, our method QSketch-Dyn is 30% more accurate than alternative methods with the number of registers $m = 2^8$. It is imperative to emphasize that QSketch and QSketch-Dyn utilize only 1/8 of the memory compared to LM and FastGM.

5.4.2 Results on Synthetic Datasets. To comprehensively assess the efficacy of QSketch across diverse scenarios, we conduct a thorough evaluation comparing its performance with that of other baseline methods. This evaluation encompasses a range of factors including data distribution, dataset scale, register count, and register size.

Performance under different data distribution. We compare our methods QSketch and QSketch-Dyn with other methods on synthetic datasets from different distributions. Figure 3 illustrates the comparative performance. Remarkably, QSketch-Dyn consistently outperforms other methods across all distributions, the same as real-world dataset results.

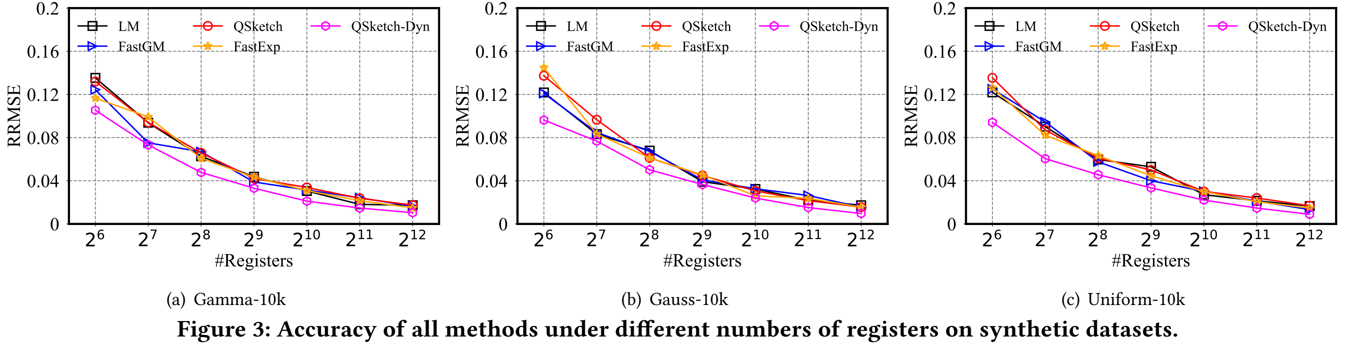
Performance under different dataset sizes. Next, we explore the performance of our methodologies across varying dataset sizes. We generate datasets from three distributions at different scales ranging from 10^2 to 10^6. The results of the remaining two distributions are summarized in the Appendix. The number of registers for all methods is fixed at 2^8. As shown in Figure 4, QSketch, LM, FastGM, and FastExp Sketch estimation errors remain consistent across all dataset scales. However, the performance of QSketch-Dyn



(a) Twitter (b) Libimseti (c) News20

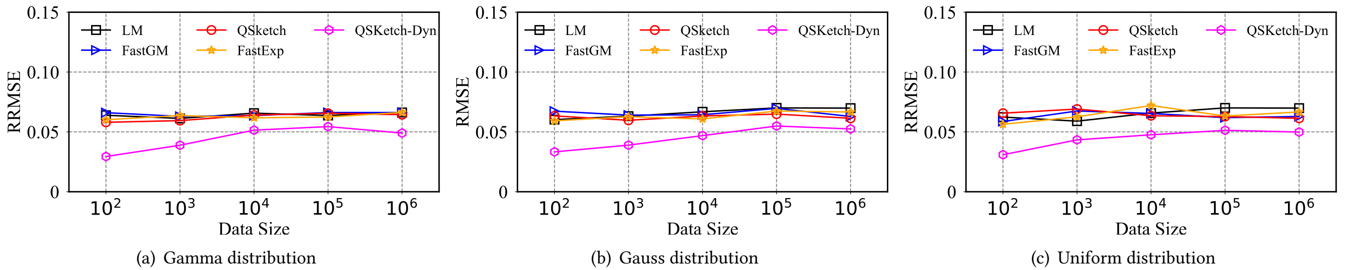
(d) Rcv1 (e) Real-sim (f) Webspam

Figure 2: Accuracy of all methods under different numbers of registers on real-world datasets.



(a) Gamma-10k (b) Gauss-10k (c) Uniform-10k

Figure 3: Accuracy of all methods under different numbers of registers on synthetic datasets.



(a) Gamma distribution (b) Gauss distribution (c) Uniform distribution

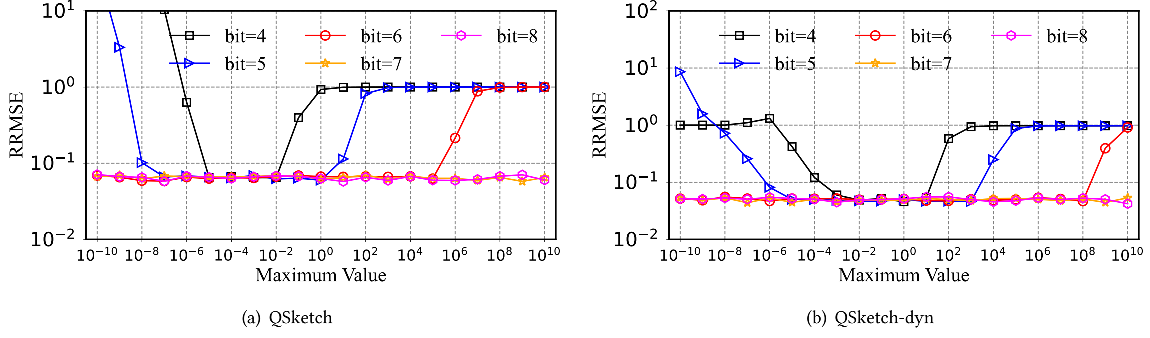
Figure 4: Accuracy of all methods under different data sizes on synthetic datasets.

shows a slight improvement with increasing dataset scale, with the estimation error stabilizing around 0.05 for dataset sizes exceeding 10^4. This phenomenon primarily stems from the fact that, with smaller dataset sizes, most registers in QSketch-Dyn are populated by only one element, resulting in nearly exact counting.

Performance under different register size. As mentioned in Theorem 1, the bit size of the sketch's registers also influences its performance. Figure 5 illustrates the estimation error of QSketch and QSketch-Dyn on the Uniform-10k distribution, considering the maximum value of the distribution ranging from 10^{-10} to 10^{10} (i.e., weighted cardinality ranging from 5×10^{-7} to 5×10^{13}). The number of registers of both methods is set to 2^8. It is evident that when employing 4- or 5-bit registers, both QSketch and QSketch-Dyn offer accurate estimations within a limited range. However, with a bit size increase to 7 or 8, both methods consistently perform well across all values, aligning with the findings of Theorem 1.

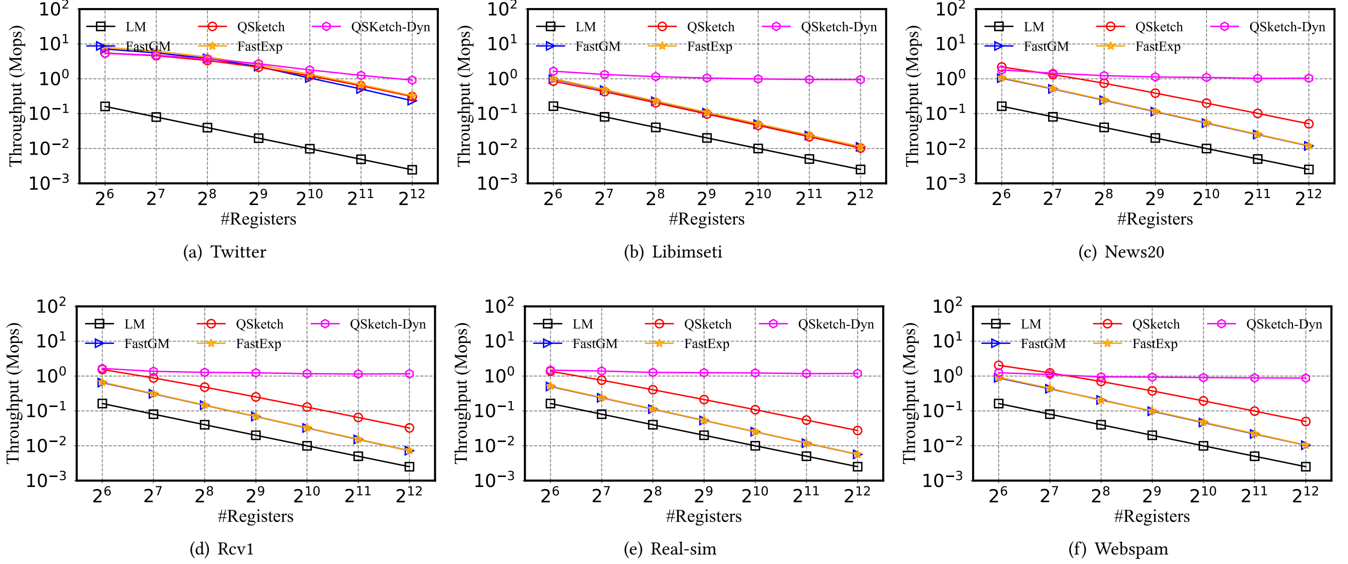
5.5 Efficiency Analysis

For efficiency, we measure the **Throughput** (Millions of updates per second) as the update speed of a sketch for incoming elements, and the **Estimation time** as the time taken to calculate the weighted cardinality from the sketch.



(a) QSketch (b) QSketch-dyn

Figure 5: Accuracy of our methods QSketch and QSketch-Dyn under different register sizes on synthetic datasets.



(a) Twitter (b) Libimseti (c) News20

(d) Rcv1 (e) Real-sim (f) Webspam

Figure 6: Update throughput of all methods under different numbers of registers on real-world datasets.

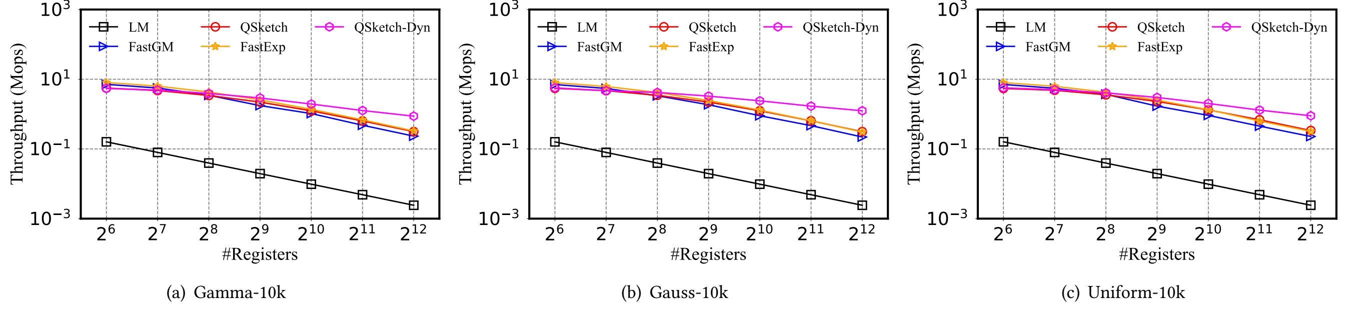
5.5.1 Results on Real-World Datasets. Figure 6 shows the results of the update throughput on real-world datasets. The throughput for LM, FastGM, FastExp Sketch, and QSketch demonstrates a decrease with more registers in the sketch. Moreover, the update throughput for FastGM and FastExp Sketch exhibits similarity and is faster than LM. For QSketch, since it uses packed integers for implementation, it needs fewer memory accesses than FastGM and FastExp, which leads to a large update throughput on most datasets. The update throughput for QSketch-Dyn remains nearly consistent across varying numbers of registers. Specifically, on dataset Rcv1, the update time for QSketch-Dyn is approximately 2 to 3 orders of magnitude shorter compared to FastGM and LM, respectively.

5.5.2 Results on Synthetic Datasets. Figure 7 shows the experimental results of update throughput on synthetic datasets with three different distributions. Remarkably, the update throughput exhibits similar trends across all three distributions. Overall, QSketch-Dyn emerges as the superior performer among all competitors, a trend consistent with the results observed on real-world datasets. Specifically, the update throughput for QSketch-Dyn is approximately 10 and 100 times shorter compared to FastGM and LM, respectively. Figure 8 shows the estimation time of all methods on three synthetic datasets. We omit similar results on other datasets. The estimation time of LM, FastGM, and FastExp Sketch is only related to the number of registers in the sketch. QSketch needs several iterations for convergence, which costs more time. Fortunately, in practical application scenarios, the estimation procedure may happen much less frequently than the update procedure, and the absolute estimation time of QSketch is only 0.01s when using 4,096 registers, which is acceptable. Besides, QSketch-Dyn keeps track of the weighted cardinality on the fly, and it does not need an estimation procedure.

6 RELATED WORK

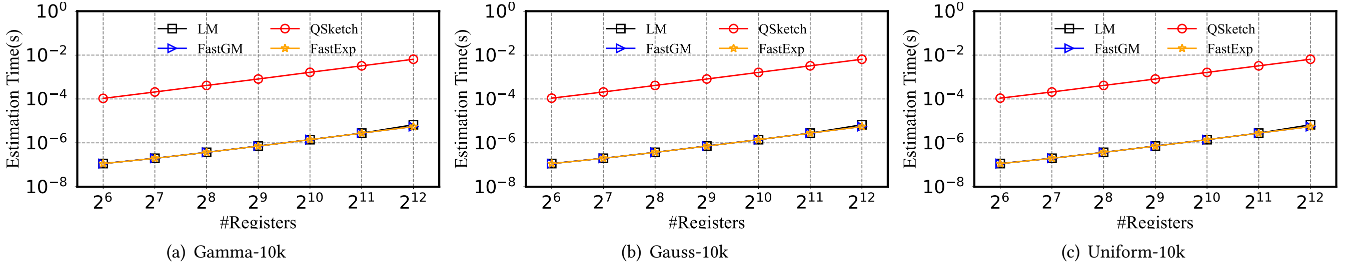
6.1 Cardinality Estimation

Harmouch et al. [21] give a comprehensive review of existing sketch methods of estimating the cardinality. Whang et al. [40] introduce the LPC sketch using random hash functions for element mapping. Various enhancements to LPC's estimation range were later proposed [6, 14]. Flajolet and Martin [18] develop the FM sketch, which was subsequently refined through methods like LogLog [13], HyperLogLog [17], RoughEstimator [22], and HLL-TailCut+ [43], reducing register size and employing multiple registers. Giroire et al. [20] develop a sketching method *MinCount* (also known as bottom-k sketch [8]) which stores the k minimum hash values of



(a) Gamma-10k (b) Gauss-10k (c) Uniform-10k

Figure 7: Update throughput of all methods under different numbers of registers on synthetic datasets.



(a) Gamma-10k (b) Gauss-10k (c) Uniform-10k

Figure 8: Estimation time of all methods under different numbers of registers on synthetic datasets.

elements in the set. Ting [34] develops a martingale-based estimator to improve the accuracy of the above sketch methods such as LPC, HyperLogLog, and MinCount. Chen et al. [7] extend HyperLogLog to sliding windows. Besides sketch methods, two sampling methods [16, 19] are also proposed for cardinality estimation. Recently, considerable attention [37, 42, 44, 46] has been given to developing fast sketch methods for monitoring the cardinalities of network hosts over high-speed links. Ting [35] developed methods to estimate the cardinality of set unions and intersections from MinCount sketches. Cohen et al. [10] developed a method combining MinHash and HyperLogLog to estimate set intersection cardinalities. Karppa et al. [23] optimized HyperLogLog by decomposing register values into a base value and an offset vector, leading to more efficient storage. Very recently, Wang et al. [38] proposed a method named Half-Xor to handle element deletions in cardinality estimation.

6.2 Weighted Cardinality Estimation

To estimate the weighted cardinality, Considine et al. [11] used binary representations to represent integer weights, which is not efficient for elements with large weights. Cohen et al. [9] proposed a weighted estimator based on bottom-k sketches. However, bottom-k sketches require maintaining a sorted list of the k smallest values, which needs more updating time and memory usage. Recently, Lemiesz [26] presented a method that maps each element to m exponential distributed variables. Thus it needs $O(m)$ time to process an incoming element, which is infeasible for high-speed streams. Therefore, Zhang et al. [45] proposed FastGM to accelerate [26]. FastGM generates these exponential variables in ascending order and stops the generation in advance if the generated value is greater than the maximal value in current registers. As a recent simultaneous work, FastExpSketch [27] shares the same idea with FastGM. However, these methods store generated values with 32-bit or 64-bit floating-point registers. When we need a large m to achieve better accuracy or there are many different streams, it is memory-intensive for devices with limited computational and storage resources. In addition, these methods need $O(m)$ to estimate weighted cardinality, and it is not efficient for them to provide anytime-available estimation for real-time applications.

7 CONCLUSIONS AND FUTURE WORK

This paper introduces QSketch, a memory-efficient sketching technique that leverages quantization methods to transform continuous register values into discrete integers. Unlike traditional sketching approaches which allocate 64 bits per register, our QSketch achieves comparable performance using only 8 bits per register. The QSketch experiences a worst-case time of $O(m \cdot n)$ where the weights of elements increase over time, and it needs to solve an MLE problem through the Newton-Raphson method, which introduces extra computational overhead. Therefore, we further capitalize on the dynamic nature of sketches by proposing QSketch-Dyn, which enables real-time monitoring of weighted cardinality. This enhanced method reduces estimation errors and maintains a constant time complexity for updates. We validate our approach through experiments conducted on both synthetic and real-world datasets. The results demonstrate that our novel sketching approach outperforms existing methods by approximately 30% while consuming only one-eighth of the memory. In the future, we aim to explore weighted cardinality in streaming scenarios, particularly focusing on handling element deletions and elements with negative weights.

ACKNOWLEDGMENT

The authors thank the reviewers for their comments and suggestions. This work was supported by the National Natural Science Foundation of China (U22B2019, 62372362, 62272372, 62272379).

REFERENCES

[1] 2010. Twitter. https://anlab-kaist.github.io/traces/
[2] 2018. CAIDA. https://www.caida.org/
[3] 2022. https://github.com/mkarppa/hyperlogloglog/blob/master/hyperlogloglog/PackedVector.hpp#L130
[4] 2024. https://github.com/Rezerolird/QSketch
[5] Saba Akram and Quarrat Ul Ann. 2015. Newton raphson method. *International Journal of Scientific & Engineering Research* 6, 7 (2015), 1748–1752.
[6] Aiyou Chen, Jin Cao, Larry Shepp, and Tuan Nguyen. 2011. Distinct counting with a self-learning bitmap. *JASA* 106, 495 (2011), 879–890.
[7] Wenji Chen, Yang Liu, and Yong Guan. 2013. Cardinality change-based early detection of large-scale cyber-attacks. In *IEEE INFOCOM*. 1788–1796.
[8] Edith Cohen and Haim Kaplan. 2007. Summarizing data using bottom-k sketches. In *PODC*. 225–234.
[9] Edith Cohen and Haim Kaplan. 2008. Tighter estimation using bottom k sketches. *PVLDB* 1, 1 (2008), 213–224.
[10] Reuven Cohen, Liran Katzir, and Aviv Yehezkel. 2017. A minimal variance estimator for the cardinality of big data set intersection. In *ACM SIGKDD*. 95–103.
[11] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. 2004. Approximate aggregation techniques for sensor databases. In *IEEE ICDE*. 449–460.
[12] Harald Cramér. 1999. *Mathematical methods of statistics*. Vol. 26. Princeton university press.
[13] Marianne Durand and Philippe Flajolet. 2003. Loglog counting of large cardinalities. In *ESA*. 605–617.
[14] Cristian Estan, George Varghese, and Mike Fisk. 2003. Bitmap algorithms for counting active flows on high speed links. In *SIGCOMM*. 153–166.
[15] Ronald A Fisher and Frank Yates. 1938. *Statistical tables: For biological, agricultural and medical research*. Oliver and Boyd.
[16] P. Flajolet. 1990. On Adaptive Sampling. *Computing* 43, 4 (1990), 391–400.
[17] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *DMTCS* Proceedings (2007).
[18] Philippe Flajolet and G Nigel Martin. 1985. Probabilistic counting algorithms for data base applications. *JCSS* 31, 2 (1985), 182–209.
[19] Phillip B Gibbons. 2001. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB*, Vol. 1. 541–550.
[20] Frédéric Giroire. 2009. Order statistics and estimating cardinalities of massive data sets. *DAM* 157, 2 (2009), 406–427.
[21] Hazar Harmouch and Felix Naumann. 2017. Cardinality estimation: An experimental survey. *PVLDB* 11, 4 (2017), 499–512.
[22] Daniel M Kane, Jelani Nelson, and David P Woodruff. 2010. An optimal algorithm for the distinct elements problem. In *PODS*. 41–52.
[23] Matti Karppa and Rasmus Pagh. 2022. HyperLogLogLog: Cardinality Estimation With One Log More. In *ACM SIGKDD*. 753–761.
[24] S Sathiya Keerthi, Dennis DeCoste, and Thorsten Joachims. 2005. A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research* 6, 3 (2005).
[25] Jérôme Kunegis, Gerd Gröner, and Thomas Gottron. 2012. Online dating recommender systems: The split-complex number approach. In *Proceedings of the 4th ACM RecSys workshop on Recommender systems and the social web*. 37–44.
[26] Jakub Lemiesz. 2021. On the algebra of data sketches. *PVLDB* 14, 9 (2021), 1655–1667.
[27] Jakub Lemiesz. 2023. Efficient framework for operating on data sketches. *PVLDB* 16, 8 (2023), 1967–1978.
[28] David D Lewis, Yiming Yang, Tony Russell-Rose, and Fan Li. 2004. Rcv1: A new benchmark collection for text categorization research. *JMLR* 5, Apr (2004), 361–397.
[29] Ping Li and Christian König. 2010. b-Bit minwise hashing. In *WWW*. 671–680.
[30] Xiaohui Long and Torsten Suel. 2005. Three-level caching for efficient query processing in large web search engines. In *WebConf*. 257–266.
[31] Rifat Ozcan, Ismail Sengor Altingovde, and Özgür Ulusoy. 2011. Cost-aware strategies for query result caching in web search engines. *ACM TWEB* 5, 2 (2011), 1–25.
[32] Christopher R Palmer, Georgos Siganos, Michalis Faloutsos, Christos Faloutsos, and Phillip B Gibbons. 2001. The connectivity and fault-tolerance of the Internet topology. (2001).
[33] Yiyan Qi, Pinghui Wang, Yuanming Zhang, Junzhou Zhao, Guangjian Tian, and Xiaohong Guan. 2020. Fast generating a large number of gumbel-max variables. In *WebConf*. 796–807.
[34] Daniel Ting. 2014. Streamed Approximate Counting of Distinct Elements: Beating Optimal Batch Methods. In *ACM SIGKDD*. 442–451.
[35] Daniel Ting. 2016. Towards optimal cardinality estimation of unions and intersections with sketches. In *ACM SIGKDD*. 1195–1204.
[36] De Wang, Danesh Irani, and Calton Pu. 2012. Evolutionary study of web spam: Webb spam corpus 2011 versus webb spam corpus 2006. In *IEEE CollaborateCom*. 40–49.
[37] Pinghui Wang, Xiaohong Guan, Tao Qin, and Qiuzhen Huang. 2011. A data streaming method for monitoring host connection degrees of high-speed links. *IEEE TIFS* 6, 3 (2011), 1086–1098.
[38] Pinghui Wang, Dongdong Xie, Junzhou Zhao, Jinsong Li, Zhicheng Li, Rundong Li, and Yang Ren. 2024. Half-Xor: A Fully-Dynamic Sketch for Estimating the Number of Distinct Values in Big Tables. *IEEE Transactions on Knowledge and Data Engineering* (2024).
[39] Wu Wei, Bin Li, Chen Ling, and Chengqi Zhang. 2017. Consistent Weighted Sampling Made More Practical. In *WebConf*. 1035–1043.
[40] Kyu-Young Whang, Brad T Vander-Zanden, and Howard M Taylor. 1990. A linear-time probabilistic counting algorithm for database applications. *ACM TODS* 15, 2 (1990), 208–229.
[41] Viktor Witkovskỳ. 2001. Computing the distribution of a linear combination of inverted gamma variables. *Science Direct Working Paper* S1574-0358 (2001), 04.
[42] Qingjun Xiao, Shigang Chen, Min Chen, and Yibei Ling. 2015. Hyper-compact virtual estimators for big network data based on register sharing. In *ACM SIGMETRICS*. 417–428.
[43] Qingjun Xiao, You Zhou, and Shigang Chen. 2017. Better with fewer bits: Improving the performance of cardinality estimation of large data streams. In *IEEE INFOCOM*. 1–9.
[44] M Yoon, Tao Li, Shigang Chen, and J-K Peir. 2009. Fit a spread estimator in small memory. In *IEEE INFOCOM*. 504–512.
[45] Yuanming Zhang, Pinghui Wang, Yiyan Qi, Kuankuan Cheng, Junzhou Zhao, Guangjian Tian, and Xiaohong Guan. 2023. Fast Gumbel-Max Sketch and its Applications. *IEEE TKDE* (2023).
[46] Qi Zhao, Abhishek Kumar, and Jun Xu. 2005. Joint Data Streaming and Sampling Techniques for Detection of Super Sources and Destinations.. In *ACM SIGCOMM*. 77–90.

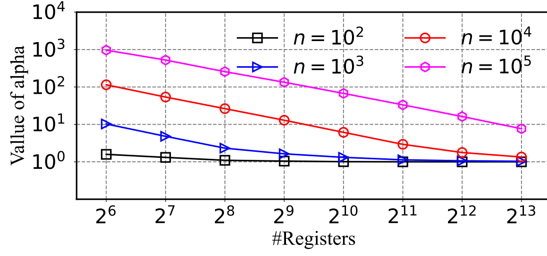


Figure 9: The value of alpha with different n and m on uniform distribution.

A APPENDXIX

A.1 Fisher-Yates shuffle

The Fisher-Yates shuffle is commonly utilized to generate an unbiased permutation step-by-step. The pseudocode of the shuffle is summarized in Algorithm 4. Note that after the i-th step of the Fisher-Yates shuffle, the first i positions of the permutation are computed and fixed.

Algorithm 4: Fisher-Yates shuffle.

```
input  : An initial permutation [π_1, ..., π_m].
output : Shuffled permutation [π'_1, ..., π'_m].
1 [π_1, ..., π_m] ← [1, ..., m] ;
2 foreach i ∈ {1, ..., m} do
3 |   k ← RandInt (i, m) ;
4 |   Swap (π_k, π_i);
```

A.2 Proof of Theorem 1

According to Equation (7), the probability that a register value $R[j] \leq r_{\min}$ or $R[j] \geq r_{\max}$ is computed as,

$$\begin{aligned} Pr(R[j] \leq r_{\min}) &= \sum_{r=-\infty}^{r_{\min}} Pr(R[j] = r) \\ &= \int_{2^{-(r_{\min}+1)}}^{+\infty} C_\Pi \cdot e^{-C_\Pi x} dx \\ &= e^{-C_\Pi \cdot 2^{-(r_{\min}+1)}}, \end{aligned}$$

and

$$\begin{aligned} Pr(R[j] \geq r_{\max}) &= \sum_{r_{\min}}^{r=+\infty} Pr(R[j] = r) \\ &= \int_{0}^{2^{-r_{\max}}} C_\Pi \cdot e^{-C_\Pi x} dx \\ &= 1 - e^{-c_\Pi \cdot 2^{-r_{\max}}}. \end{aligned}$$

By setting $Pr(R[j] \leq r_{\min}) < \epsilon$ and $Pr(R[j] \geq r_{\max}) < \epsilon$ simultaneously for above formulas, we have

$$-2^{(r_{\min}+1)} \cdot \ln \epsilon < C_\Pi < -2^{r_{\max}} \cdot \ln(1-\epsilon),$$

which is the conclusion of the theorem.

A.3 Proof of Theorem 2

Given the data stream Π the set of timestamps that each element appears in the stream for the first time $T_s^{(t)}$, we first derive the expectation. Let $\mathbf{1}(R^{(t)} \neq R^{(t-1)})$ demote an indicator of whether the state of sketch R has changed at time t, i.e., $\mathbf{1}(R^{(t)} \neq R^{(t-1)}) = 1$ for $R^{(t)} \neq R^{(t-1)}$ and 0 otherwise. We note that $q_R^{(t)}$ only depends on $R^{(t-1)}$ and then we have

$$\mathbb{E}[\mathbf{1}(R^{(t)} \neq R^{(t-1)})|q_R^{(t)}] = \mathbb{E}[\mathbf{1}(R^{(t)} \neq R^{(t-1)})|R^{(t-1)}] = q_R^{(t)}.$$

From the linearity of expectation and the law of total expectation, we have

$$\begin{aligned} \mathbb{E}[\hat{C}_\Pi^{(t)}] &= \mathbb{E}[\mathbb{E}[\hat{C}_\Pi^{(t)}|R^{(t-1)}]] \\ &= \mathbb{E}[\mathbb{E}[\hat{C}_\Pi^{(t-1)}|R^{(t-1)}] + \mathbb{E}[\frac{\mathbf{1}(R^{(t)} \neq R^{(t-1)})}{q_R^{(t)}} w^{(t)}|R^{(t-1)}]] \\ &= \mathbb{E}[\hat{C}_\Pi^{(t-1)}] + w^{(t)} = C_\Pi^{(t)}. \end{aligned}$$

For the variance, we have

$$\mathrm{Var}[\hat{C}_\Pi^{(t)}] = \mathbb{E}[(\hat{C}_\Pi^{(t)})^2] - \mathbb{E}[\hat{C}_\Pi^{(t)}]^2 = \mathbb{E}[(\hat{C}_\Pi^{(t)})^2] - (C_\Pi^{(t)})^2.$$

Following a similar derivation with the expectation, we have

$$\mathbb{E}[(\hat{C}_\Pi^{(t)})^2] = \mathbb{E}[\mathbb{E}[(\hat{C}_\Pi^{(t)})^2|R^{(t-1)}]],$$

where $\mathbb{E}[(\hat{C}_\Pi^{(t)})^2|R^{(t-1)}]$ is computed as

$$\begin{aligned} &\mathbb{E}[(\hat{C}_\Pi^{(t)})^2|R^{(t-1)}] \\ =&(\hat{C}_\Pi^{(t-1)})^2 + \frac{2w^{(t)}\hat{C}_\Pi^{(t-1)}}{q_R^{(t)}}\mathbb{E}[\mathbf{1}(R^{(t)} \neq R^{(t-1)})|R^{(t-1)}] + \\ &(\frac{w^{(t)}}{q_R^{(t)}})^2\mathbb{E}[(\mathbf{1}(R^{(t)} \neq R^{(t-1)}))^2|R^{(t-1)}] \\ =&(\hat{C}_\Pi^{(t-1)})^2 + 2w^{(t)}\hat{C}_\Pi^{(t-1)} + \frac{(w^{(t)})^2}{q_R^{(t)}}. \end{aligned}$$

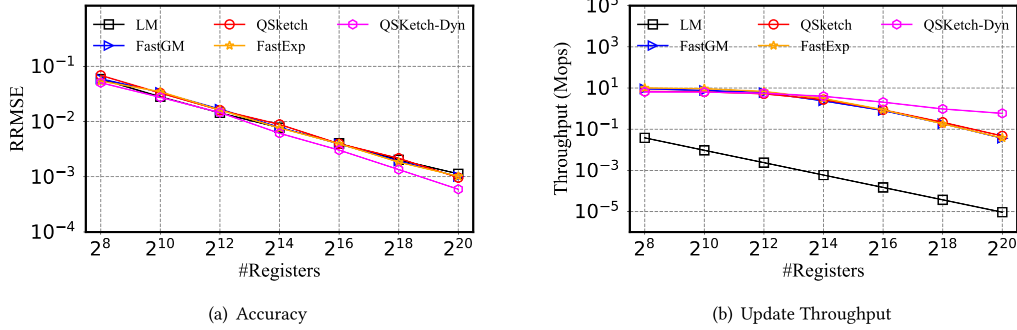
Then, we have

$$\begin{aligned} \mathbb{E}[(\hat{C}_\Pi^{(t)})^2] &= \mathbb{E}[(\hat{C}_\Pi^{(t-1)})^2 + 2w^{(t)}\hat{C}_\Pi^{(t-1)} + \frac{(w^{(t)})^2}{q_R^{(t)}}] \\ &= \mathbb{E}[(\hat{C}_\Pi^{(t-1)})^2] + 2w^{(t)}\mathbb{E}[\hat{C}_\Pi^{(t-1)}] + \mathbb{E}[\frac{(w^{(t)})^2}{q_R^{(t)}}] \\ &= \sum_{i \in T_s^{(t)}} \mathbb{E}[\frac{(w^{(i)})^2}{q_R^{(i)}}] + 2 \sum_{i,j \in T_s^{(t)} \wedge i \neq j} w^{(i)} w^{(j)}. \end{aligned}$$

Finally, we obtain

$$\mathrm{Var}[\hat{C}_\Pi^{(t)}] = \mathbb{E}[(\hat{C}_\Pi^{(t)})^2] - \mathbb{E}[\hat{C}_\Pi^{(t)}]^2 = \sum_{i \in T_s^{(t)}} (w^{(i)})^2 \mathbb{E}[\frac{1 - q_R^{(i)}}{q_R^{(i)}}].$$

An exact expression for $\mathbb{E}[\frac{1-q_R^{(i)}}{q_R^{(i)}}]$ can be derived with the probability $P(R^{(t)}[1] = r_1, \ldots, R^{(t)}[m] = r_m|\Pi)$. However, it is too



(a) Accuracy (b) Update Throughput

Figure 10: Experiments on CAIDA dataset under a different number of registers.

complex to analyze. Hence, we try to approximate the variance via the empirical analysis.

$$\begin{aligned}\mathrm{Var}[\hat{C}_\Pi^{(t)}] <& w_{\max}^{(t)} \sum_{i \in T_s^{(t)}} \mathbb{E}[\frac{w^{(i)}}{q_R^{(i)}}] - w_{\max}^{(t)} C_\Pi^{(t)} \\ =& (\alpha_{dis}(n,m) - 1) w_{\max}^{(t)} C_\Pi^{(t)},\end{aligned}$$

where $w_{\max}^{(t)} = \max_{i \in T_s^{(t)}} w^{(i)}$ and $\alpha_{dis}(n,m)$ is a function of number of elements n and number of registers m.

As an illustration, we consider a uniform distribution from the interval $(0, 1)$. Figure 9 depicts the variation in the function's value across different values of m and n. As a result, we can get an upper bound of the variance together with the weighted cardinality estimation.

A.4 Results on Large-Scale Dataset CAIDA

We further conduct experiments on the CAIDA [2] dataset, which consists of streams of anonymized IP items collected from high-speed monitors by CAIDA in 2018. A 1-minute CAIDA network traffic trace contains about 27M packets. For each packet, we consider the tuple (source IP, target IP) as the identifier of the element e, and the packet size as the weight of the element e. Experimental results are summarized in Figure 10. We vary the number of registers in each sketch $m \in \{2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}\}$, and evaluate the RRMSE as well as the update throughput. All methods achieve better estimation accuracy when using more registers at the cost of lower update throughput. Besides, we observe that:

- **QSketch achieves similar estimation accuracy as LM, FastGM, and FastGM, and QSketch-Dyn performs best among all methods.** This is consistent with previous results. For example, when using $m = 2^{20}$ registers, the estimation accuracy of QSketch-Dyn is about twice that of other methods.
- **The update throughput for QSketch-Dyn remains nearly consistent across varying numbers of registers and is higher than other methods.** This is also consistent with previous results. For example, when use $m = 2^{19}$ registers, the update throughput of QSketch-Dyn is about 1 Mops, while the update throughput of FastGM, QSketch, and FastExp Sketch is only about 0.2 Mops, which means that QSketch-Dyn is about 5× faster.