KLL$^{\pm}$ Approximate Quantile Sketches over Dynamic Datasets

Fuheng Zhao
UC Santa Barbara
fuheng_zhao@ucsb.edu

Sujaya Maiyya
UC Santa Barbara
sujaya_maiyya@cs.ucsb.edu

Ryan Wiener
UC Santa Barbara
ryanlwiener@ucsb.edu

Divyakant Agrawal
UC Santa Barbara
agrawal@cs.ucsb.edu

Amr El Abbadi
UC Santa Barbara
amr@cs.ucsb.edu

ABSTRACT

Recently the long standing problem of optimal construction of quantile sketches was resolved by **K**arnin, **L**ang, and **L**iberty using the KLL sketch (FOCS 2016). The algorithm for KLL is restricted to online insert operations and no delete operations. For many real-world applications, it is necessary to support delete operations. When the data set is updated dynamically, i.e., when data elements are inserted and deleted, the quantile sketch should reflect the changes. In this paper, we propose *KLL*$^{\pm}$, the first quantile approximation algorithm to operate in the *bounded deletion* model to account for both inserts and deletes in a given data stream. KLL$^{\pm}$ extends the functionality of KLL sketches to support arbitrary updates with small space overhead. The space bound for KLL$^{\pm}$ is $O(\frac{\alpha^{1.5}}{\epsilon} log^2 log(\frac{1}{\epsilon\delta}))$, where ϵ and δ are constants that determine precision and failure probability, and α bounds the number of deletions with respect to insert operations. The experimental evaluation of KLL$^{\pm}$ highlights that with minimal space overhead, KLL$^{\pm}$ achieves comparable accuracy in quantile approximation to KLL.

PVLDB Reference Format:
Fuheng Zhao, Sujaya Maiyya, Ryan Wiener, Divyakant Agrawal, and Amr El Abbadi. KLL$^{\pm}$ Approximate Quantile Sketches over Dynamic Datasets. PVLDB, 14(7): 1215 - 1227, 2021.
doi:10.14778/3450980.3450990

1 INTRODUCTION

With the rise of big data in companies such as Google, Amazon, and Facebook, managing hundreds of terabytes to petabytes of data has become a necessity for day to day operations. To make use of this data, it is crucial to develop a deeper understanding of the underlying distributions of the data in real datasets. In particular, techniques such as quantile approximations, a non-parametric representation, are widely used to characterize data distributions. For large amounts of data, one-pass[1] algorithms are desirable, and many well-known data sketches are based on one-pass algorithms. For instance, the HyperLogLog [11] sketch[2] is a one-pass algorithm for the *count-distinct* problem; the Bloom Filter [4] data sketch is a one-pass algorithm for the *set membership* problem. Several one-pass quantile approximation algorithms [9, 12, 13, 20, 21, 27] have been proposed to guarantee high precision with small memory footprints. Since one-pass algorithms only read each element from the input once, these one-pass algorithms are both useful for large databases, and naturally applicable in the *streaming* data settings.

Approximate quantile problems are developed for a variety of settings. For example, approximate quantiles are considered in the streaming data settings [20, 21], sliding window settings [3], and distributed settings [27]. Moreover, approximate quantile sketches operate mainly within the *insertion-only* model and the *turnstile* model. The insertion-only model, also referred as the cash register model, consists of a stream of only insert operations; whereas the turnstile model consists of a stream of insert and delete operations such that deletes are performed on previously inserted items [28]. Quantile sketches such as the GK sketch [13] and Q-digest [27] operate in the insertion-only model; and quantile sketches such as Range Subset Sum [12], Dyadic Count-Min [9], and Dyadic Count Sketch [28] operate in the turnstile model. Turnstile model quantile summaries have significant usage in database and networking applications, and are used in most Database Management Systems (DBMSs) which monitor and maintain a variety of order statistics such as quantiles over the contents of database relations [25]; in value range partitioning for parallel database [26]; and in financial services [17]. Sketches proposed for the turnstile model assume a fixed universe in order to tolerate an arbitrary number of deletions and thus incur higher space and update complexity compared to data-driven sketch. The additional challenge of tolerating an arbitrary amount of deletion for the turnstile model is arguably an infeasible task for data-driven sketches which stores a subset of items chosen from the input. Several researchers [12, 28] have noted that if one first inserts n elements and then deletes all but one element, the data-driven sketch has no information about which element will survive since the data-driven sketch only keeps a subset of items. Wang et al. [28] conjecture the impossibility of any data-driven sketch to support the turnstile model where all elements can be deleted. Therefore, we focus on developing a data-driven sketch in the context of the *bounded deletion* model.

Recently, Jayaram et al. [19] observed that in practice many turnstile models only incur a fraction of deletions and thus proposed an intermediate model, called the *bounded deletions* model, which naturally lies in between the insertion-only stream model and the turnstile model. The authors [19] use parameter α to denote different degrees of deletions such that at most $1 - \frac{1}{\alpha}$ of prior insertions are deleted; when $\alpha = 1$ the input dataset is in the insertion-only model. The authors [19] show significant space improvements for

Proceedings of the VLDB Endowment, Vol. 14, No. 7 ISSN 2150-8097.
doi:10.14778/3450980.3450990

[1] Where input data is read only once.
[2] The term *sketch* refers to the algorithm and data structures that can extract valuable information through one pass of the data.

many fundamental streaming problems within the bounded deletion model over the turnstile model, such as heavy hitters and inner product estimation, and identify the bounded deletion model as being particularly useful for applications such as computing differences between network traffic patterns, L_0 estimation of moving sensors (monitoring wildlife or water flow pattern), and completing synchronization in database analytical context.

Furthermore the bounded deletion model is important for maintaining approximate quantiles in real-world applications such as summarizing product sales in electronic commerce platforms and ranking in standardized testings. Many companies need accurate quantile information to identify current status and forecast future demands [15]. After customers purchase some products, a certain percentage of the customers may return the product and submit refund requests, hence the quantile summaries should reflect these changes. However, it is highly unlikely that the majority of these customers will return their purchases and in most cases a bounded deletion model can be assumed. In the context of standardized testing such as SAT, GRE, and LSAT, quantiles are seen as a more descriptive measurement compared to the test scores, since the difficulty of the exam varies. In these standardized testing settings, students may request to regrade their exam only once to verify any machine errors in scanning answers or human errors in grading the essay. The updated grades may change the underlying score distribution. Thus, a quantile summary with $\alpha = 2$, is sufficient to reflect all changes even if all students require a regrade[3]. In Section 4.3, we also show how to leverage KLL$^{\pm}$ with $\alpha = 2$ to maintain fixed-size sliding window quantile approximation over datasets.

In this paper, we extend the original optimal quantile sketch KLL [20], which operates in the insertion-only model, and present KLL$^{\pm}$ quantile sketch that supports bounded deletions in which the total number of deleted items are less than a certain threshold of the total inserted items, and the deleted items are previously inserted into the data set. The KLL algorithm is online updatable for insert operations, but not for delete operations. Without delete operations, a KLL sketch cannot be maintained for a dynamic dataset that is constantly updated as an expensive scan operation will be needed to recompute the KLL-sketch, when deletions are applied. Our KLL$^{\pm}$ expands the functionality of the original KLL sketch by supporting bounded deletions; and if the administrator of the big data, knows a priori that deletions are not insensitively large compared to the insertions, such as some corrections or adjustments on previously inserted items, then KLL$^{\pm}$ can approximate quantiles with small space and high accuracy. In summary, the main contributions in this paper are: (i) Presenting the KLL$^{\pm}$ sketch, the first data-driven quantile sketch that operates in the bounded deletion model; (ii) Providing thorough mathematical proofs and analysis to guarantee the correctness of KLL$^{\pm}$ sketches; (iii) Applying the KLL$^{\pm}$ sketches over datasets to maintain approximate quantile estimations in the fixed-size sliding window; (iv) Evaluating KLL$^{\pm}$ and comparing it with KLL, two parallel KLL strategy, and DCS sketch [28] through various experiments; and (v) Incorporating several optimization strategies [18] proposed for the original KLL sketch [20], such as implementing the algorithm with fixed memory size.

[3]https://collegereadiness.collegeboard.org/sat/scores/verifying-scores

This paper is structured as follows: Section 2 discusses background information of quantile sketches, and gives an overview of previous algorithms. Section 3 introduces the KLL$^{\pm}$ quantile sketch for datasets with an incoming input stream in the bounded deletion model, and along with the proposed algorithm for compaction referred to as *Conditional Compaction* algorithm. Section 4 analyzes the space complexity of our sketch, and demonstrates a potential use case of KLL$^{\pm}$ to maintain fixed-size sliding window quantile approximation over datasets. Section 5 presents the experiment results of an evaluation conducted using synthetic and real world datasets and compares KLL$^{\pm}$ with the state of the art KLL sketch (insertion-only) [20] and DCS sketch (turnstile) [28]. Finally, Section 6 summarizes our contributions and concludes this work.

2 BACKGROUND

Given a multiset of n elements $S = \{s_1, ..., s_n\}$, the rank of an element $s_i \in S$ is the number of elements in S that are less than or equal to s_i, and $R(s_i)$ is the function specific to a set S that takes as input an element s_i and returns its rank. The quantile of an element $s_i \in S$ is defined as $R(s_i)/n$. Equivalently, the ϕ-quantile of set S, $Q(\phi)$ returns the element s_i such that $Q(R(s_i)/n) = s_i$. Typically, quantile is represented as a fraction $\phi \in (0, 1]$. The relationship between the rank and quantile of an element can be represented as: $R(s_i) = \phi \cdot n$, where ϕ is a non-zero fraction less than or equal to one and n is the number of elements in the dataset. Equivalently, ϕ elements are less than or equal to s_i and $(1 - \phi) \times n$ elements are greater than s_i. The most familiar quantile value is $\phi = 0.5$ which is referred to as the *median* of the dataset.

2.1 Deterministic and Randomized algorithms

Computing the true median value for large data sets, is memory intensive. Munro and Paterson [24] proved that to find the true median of a set of size n with p sequential passes of the input requires at least $\Omega(n^{1/p})$ memory. Thus, to determine the true median using a one-pass algorithm requires memory linear to the size of the set. With limited memory and for large data sets, calculating the exact quantiles is infeasible. An alternate and more practical approach to the problem is to approximate the quantiles, represented as ϵ approximation ϕ quantile, where ϵ is the precision value.

The *deterministic* ϵ approximation ϕ quantile algorithms take as input a quantile query ϕ and a precision value ϵ and output an answer x such that x's quantile is in the range $[\phi - \epsilon, \phi + \epsilon]$. An alternative approach proposes a family of *randomized* algorithms where the output answer x is within the $[\phi - \epsilon, \phi + \epsilon]$ range with a high probability [18, 20, 21]. These algorithms provide guarantees by bounding the failure probability to at most δ such that the user has $1 - \delta$ confidence that the sketch's output is ϵ approximation.

2.2 Quantile Sketch: Insertion-only Model

Greenwald and Khanna [13] developed the GK sketch that tracks a sorted subset of elements in the input data stream such that these elements provide lower and upper bounds for each quantile individually instead of maintaining a single bound over all quantiles. The GK sketch is deterministic and requires $O(1/\epsilon \log(\epsilon N))$ space in the worst case. It is conjectured [2], however, that the GK sketch is not fully *mergeable* – the property of merging two sketches on

Sketch	Space	Update Time	Randomization	Model	Framework
GK Sketch [13]	$O(\frac{1}{\epsilon}log(\epsilon n))$	$O(log\frac{1}{\epsilon} + loglog(\epsilon n))$	Deterministic	Insertion-Only	Data-Driven
Q-digest [27]	$O(\frac{1}{\epsilon}logU)$	$O(log\frac{1}{\epsilon} + loglogU)$	Deterministic	Insertion-Only	Universe-Driven
MRL99 [22]	$O(\frac{1}{\epsilon}log^2(\frac{1}{\epsilon}))$	$O(log(\frac{1}{\epsilon}))$	Randomized	Insertion-Only	Data-Driven
Mergeable KLL [20]	$O(\frac{1}{\epsilon}log^2log(\frac{1}{\epsilon}))$	$O(\frac{1}{\epsilon})$ $O(log(\frac{1}{\epsilon}))$[18]	Randomized	Insertion-Only	Data-Driven
RSS [12]	$O(\frac{1}{\epsilon^2}log^2Ulog\frac{logU}{\epsilon})$	$O(\frac{1}{\epsilon^2}log^2Ulog\frac{logU}{\epsilon})$	Randomized	Turnstile	Universe-Driven
DCM [9]	$O(\frac{1}{\epsilon}log^2Ulog\frac{logU}{\epsilon})$	$O(logUlog\frac{logU}{\epsilon})$	Randomized	Turnstile	Universe-Driven
DCS [28]	$O(\frac{1}{\epsilon}log^{1.5}Ulog^{1.5}\frac{logU}{\epsilon})$	$O(logUlog(\frac{logU}{\epsilon}))$	Randomized	Turnstile	Universe-Driven
Mergeable KLL$^{\pm}$	$O(\frac{\alpha^{1.5}}{\epsilon}log^2log\frac{1}{\epsilon})$ s.t. $D \leq (1-1/\alpha)I$	$O(\frac{\alpha^{1.5}}{\epsilon})$ $O(log(\frac{\alpha^{1.5}}{\epsilon}))$[18]	Randomized	Bounded Deletions	Data-Driven

Table 1: Comparison between different quantile sketches

two different datasets is to create one combined or merged sketch that can then be used for quantile computation over the union of the two datasets. Mergeability allows users to compute sketches over multiple data partitions independently and to combine them in parallel to compute the summaries. The merged sketch should be as accurate as a single sketch over the entire data set. This is a crucial requirement in distributed settings, where data is often stored distributedly across different machines. Creating sketches independently and then merging them avoids the communication costs and large latency of transferring large amounts of data to a central repository. Furthermore, this also makes the quantile computation highly scalable in the context of very large datasets.

Karnin et al. [20] extended the GK sketch and presented the KLL sketch, an asymptotically optimal but non-mergeable sketch with $O(1/\epsilon \ \log\log(1/\delta\epsilon))$ space. Karnin et al. also presented a mergeable KLL sketch with $O(1/\epsilon \ \log^2\log(1/\delta\epsilon))$ space bounds. The core building block for the KLL quantile sketch is called the compactor, first introduced in [21]. KLL can be seen as an array of H compactors as depicted in Figure 1. One of the main contributions by Karnin et al. [20] is to obtain the optimal sketch size by having different capacity compactors at different heights and exponentially decreasing the capacity of compactors at lower heights.

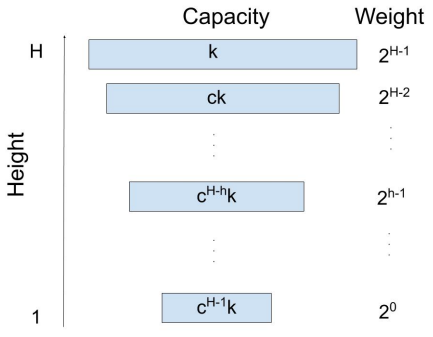


Figure 1: An illustration of a set of compactors. As the height decreases from H to 1, the capacity of the compactor decreases from k to $c^{H-1}k$. k is a constant given by the user, and c is a constant between 0.5 and 1.

When a $compactor[h]$'s size reaches its capacity, i.e., $k_h = c_h$, the compactor performs a compaction process, in which it pushes $k_h/2$ elements from $compactor[h]$ to $compactor[h+1]$. Hence, these $k_h/2$ elements update their weight to $2 \cdot w_h$, i.e., 2^h. The compaction process introduces error since each compaction pushes only half of the elements to the next level. For example, consider a compaction of two elements A, B with weight $w_h = 1$ in which rank(A)=1 and rank(B)=2. A compaction pushes either A or B, and the compacted element's weight becomes $w_{h+1} = 2$. If element A is chosen then the sketch loses the information about element B and believes element A appeared twice, hence rank(A)=2 and rank(B)=2 in which case we introduced +1 ($+w_H$) error for rank A. If element B is chosen then the sketch loses the information about element A and believes element B appeared twice, hence rank(A)=0 and rank(B)=2 in which case we introduced -1 ($-w_H$) error for rank A. In both cases, we introduce no error for rank B. Therefore the rank estimation before and after a compaction process differs by at most w_h as shown in Figure 2. Agarwal et al. [2] suggested that by removing the odd or even indexed elements with equal probability, the expected error becomes zero. This ensures that the total error is bounded by δ for $O(1/\epsilon)$ quantiles.

In summary, Karnin et al. [20] (i) exponentially decrease the compactor capacity; (ii) replace compactors of capacity 2 with a sampler, which randomly selects one element from $2^{w_H - \log_{1/c} k}$ elements; (iii) keep the size of the top $O(\log\log 1/\delta)$ compactors fixed (similar to the MRL sketch [21]); (iv) replace the top $O(\log\log 1/\delta)$ compactors with a GK sketch [13]. Recently, Ivkin et al. [18] extended the theoretical development of Karnin et al. [20] to achieve practical improvements for implementing KLL sketches. In summary the contributions are: (i) Implementing the algorithm with memory limit parameter, denoted by k, as opposed to δ (failure probability) and ϵ (precision) parameters; (ii) extending the functionality to handle weighted data streams; and (iii) various optimization strategies to reduce memory footprint and update time. These optimization strategies are orthogonal to our work, and in the technical report [1] we show how to integrate our work with the Sweep Compactor [18] to improve the worst case update time.

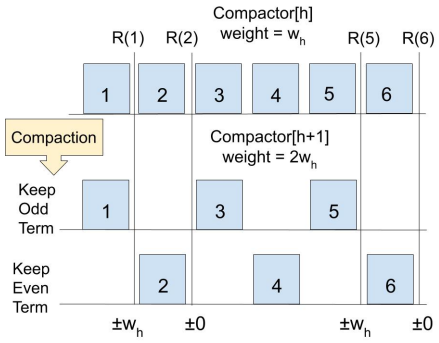


Figure 2: *Compactor*$[h]$ **reaches its capacity of 6 and performs a compaction. If the odd indexed terms are chosen then the compaction contributes** $+w_h$ **error to R(1), R(3), and R(5). If the even indexed terms are chosen then the compaction contributes** $-w_h$ **error to R(1), R(3), and R(5). In both cases the compaction contributes no error to R(2), R(4), and R(6).**

2.3 Quantile Sketch: Turnstile Model

In the turnstile model, the input data stream consists of both insertions and deletions and a deletion cannot delete an element that does not exist. This is also known as the strict definition for the turnstile model. Gilbert et al. [12] were the first to propose quantile sketches for the turnstile model and introduced quantile sketches to support both insertions and deletions in database management systems (DBMSs). The authors [12] presented the novel Random Subset Sums (RSS) sketch, which breaks down the universe U into dyadic intervals and maintains the frequency estimations of elements for each layer, using total space of $O(\frac{1}{\epsilon^2}\log^2 U \log\frac{\log U}{\epsilon})$, with update time $O(\frac{1}{\epsilon^2} log^2 U log(\frac{logU}{\epsilon}))$. The dyadic structure decomposes the universe into $\log U$ layers such that in each i layer the universe is partitioned into $U/2^i$ intervals of size 2^i. The top most layer represents the interval: $[1, U]$, the second top most layer represents two intervals: $[1, U/2]$, $[U/2+1, U]$, the third top most layer represents four intervals: $[1, U/4]$, $[U/4{+}1, U/2]$, $[U/2{+}1, 3U/4]$, $[3U/4{+}1, U]$ and so on until the bottom layer representing all the elements in U. Thus, to find the rank estimate of an element $x \in U$, we can decompose the interval $[1, x]$ into $\log x$ number of disjoint dyadic intervals, and query each of these intervals to the frequency estimation sketch of their corresponding layers and then sum all the estimations to obtain the rank estimation.

Later Cormode et al. [9] proposed the DCM (Dyadic Count Min) sketch with overall space $O(\frac{1}{\epsilon}\log^2 U \log(\frac{\log U}{\epsilon}))$ and update time $O(\log U \log(\frac{\log U}{\epsilon}))$. DCM sketch replaced the frequency estimation sketch in each layer with a Count Min Sketch, in which Count Min sketch uses small space to output the frequency estimation of each element with an additive factor of ϵ in high probability. Recently, Wang et al. [28] proposed DCS (Dyadic Count Sketch) sketch which replaced the Count Min sketch with Count sketch [6]. Count sketch is similar to Count Min sketch as it has small space overhead to output frequency estimation for each element with an additive factor of ϵ in high probability. While Count Min sketch's estimation is biased toward overestimation, Count sketch gives an unbiased frequency estimation by reporting the median. Wang et al. [28] point out that the property of unbiased frequency estimation from the Count sketch is appealing to the quantile problem. Estimations, by sketches for each dyadic layers, may give positive errors or negative errors, and these errors may cancel out each other. Thus, DCS sketches further improve the space bound to $O(\frac{1}{\epsilon}\log^{1.5} U \log^{1.5}(\frac{\log U}{\epsilon})))$ with the same update time complexity as DCM sketches.

These three sketches (RSS, DCM, DCS) guarantee that deleting a previously inserted element has no impact on the space or accuracy of the sketch. In fact, Gilbert et al. [12] compared the RSS sketch with 2-parallel GK sketches (one for insertions and one for deletions), and found that when deletions are relatively small, the simple 2-parallel GK sketches exceed the accuracy of RSS by two orders of magnitude. When the number of deletions are significantly large, RSS is more accurate than the 2-parallel GK sketches.

During the past three decades of research, various quantile summaries have been developed through a variety of models and frameworks. In Table 1, we compare the difference and similarity among several different sketches. Quantile sketches algorithms can be categorized by data-driven or universe-driven framework [8]. Algorithms in the data-driven framework keep a subset of items appeared in the stream and maintain their statistics. On the other hand, algorithms in the universe-driven framework maintain attributes over the universe, and have update time and space bounds depend on the universe size U. For instance, Shrivastava et al. [27] proposed a novel quantile sketch for the insertion-only model and universe-driven framework with $O(\frac{1}{\epsilon}\log U)$ space. Our algorithm uses the bounded deletion model and data-driven framework, more precisely the comparison framework[4]. The benefit for sketches in the data-driven framework is that these sketches make no assumptions on the universe size, hence can handle dataset with attributes involving variable-length strings while their space and update time are independent from the universe size; Recent surveys by Wang et al. [28], Greenwald et al. [14], Chen et al. [7], and Cormode et al. [10] provide comprehensive background on quantile sketches.

3 KLL$^{\pm}$ QUANTILE SKETCH

The challenges for supporting arbitrary number of delete operations in the turnstile model motivated researchers [9, 12, 28] to explore universe-driven algorithms which necessitate the sketch size and update time to be dependent on the size of the universe, which can be quite large. Our main goal is to extend data-driven algorithms to the bounded deletion model where the input consists of I insertions and at most $(1-1/\alpha)\times I$ deletions. We propose the KLL$^{\pm}$ Quantile Sketch, a generalization of the KLL sketch to maintain quantile information in the bounded deletion model using as small of a memory footprint as possible.

3.1 Basic Structure

Similar to MRL [21] and KLL [20], KLL$^{\pm}$ can be seen as an array of H compactors where H denote the total height. Each compactor is identified by its height, denoted as $compactor[h]$. The topmost compactor and the bottom compactor have height H and 1, respectively. Each $compactor[h]$ has a limited capacity $c_h = c^{H-h}k$ where

[4] Only comparisons are applied on elements.

k is the capacity of $compactor[H]$. Each $compactor[h]$ contains k_h elements such that $k_h < c_h$ and each element in $compactor[h]$ has weight $w_h = 2^{h-1}$. Once a compactor is full, $k_h = c_h$, the compactor will go thorough a compaction to free spaces for new elements.

3.2 Differentiating Inserts & Deletes

In the insertion-only model, the incoming data stream consists only insertions. To extend this model to contain both insert and delete operations, we first differentiate between *insert* and *delete* operations by using one additional bit (representing the sign) with the data value. We assume that deletions are negative numbers with negative weights and insertions are positive numbers with positive weights.Let function $sign(item)$ return 1 for insertions and return -1 for deletions. The weight of an item is defined based on the sign function and the height of the compactor. For any element in $compactor[h]$: $weight(item) = sign(item) * 2^{h-1}$. Hence, the deleted items are recorded with negative weights and inserted items with positive weights.

3.3 Conditional Compaction

Once a compactor is full, i.e, $k_h = c_h$, the compactor needs to be compacted to free space for new incoming elements. The presence of inserted and deleted elements requires a new compaction algorithm. This section presents a novel compaction algorithm, Conditional Compaction, described in Algorithm 1. In the conditional compaction algorithm we assume that the capacity of the input compactor is even. If the capacity of the compactor is odd then the first or the last element is randomly kept to ensure that an even sized number of elements are compacted.

Before compacting each compactor[h], the elements inside the compactor[h] are sorted (the sorting order does not matter), line 1 of Algorithm 1, After sorting, Conditional Compaction Algorithm follows the following compaction process:

Discard$(-\mathbf{e_i}, \mathbf{e_i})$: For every element $-e_i$, if there is a matching e_i, then $Discard(-e_i, e_i)$. This is because an insert e_i and a delete e_i operation cancel each other and any such matching pairs of inserts and deletes are discarded in the compaction process. Since the compactor is sorted[5], a classical two-pointer technique[6], can be used to find all matching pairs of $\langle delete(e_i), insert(e_i)\rangle$. If such pairs exist then $Discard(-e_i, e_i)$ results in the removal of matched pairs which in turn creates free space in the compactor and hence the algorithm skips the rest of the compaction process. A crucial property of compaction is to push elements to the compactor at the next level. Since in this case a compactor will not push any elements to the compactor at the next level, this is not a full compaction.

Push$(\mathbf{e_i}, \mathbf{e_{i+1}})$: If no $\langle delete(e_i), insert(e_i)\rangle$ pairs are found and discarded, then the algorithm performs the push operation. For every pair of adjacent elements, e_i and e_{i+1} with the same sign (i.e., they are both inserts or deletes) at level h, the algorithm decides on a *random offset* (line 6) whether the first or the second element of the pair is pushed to the next level, $C[h+1]$.

Keep$(\mathbf{e_i}, \mathbf{e_{i+1}})$: If the pair of adjacent elements, e_i and e_{i+1}, have different signs (i.e., e_i is deleted and e_{i+1} is inserted), then both elements are retained in the compactor at level h. Note that there will be at most one such mismatched pair with different signs.

The rationale for keeping adjacent elements with different signs is based on the following observation. Assume in compactor[h] there is a $\langle delete(B), insert(A)\rangle$ pair. This information implies that the input contains w_h insertions and at least one of these insertions is element A, and also w_h deletions and at least one of these deletions is element B. If we randomly push either insert A or delete B into compactor[h+1] which contains elements with weight of $2w_h$, then the sketch not only lost the information about w_h deletions or w_h insertions respectively, but also introduce false information on the total number of insertions and deletions.

Algorithm 1: Conditional Compaction in KLL$^{\pm}$

```
1  sort(C[h]);
2  Discarded = discard((−x, x) pairs ∈ C[h]);
3  if Discarded then
4  |   // No compaction needed
5  |   return;
6  offset = random(0, 1);
7  for i = 0; i < C[h].length; i = i + 2 do
8  |   if sign(C[h][i]) == sign(C[h][i + 1]) then
9  |   |   push(C[h][i + offset]) to C[h+1];
10 |   else
11 |   |   keep(C[h][i], C[h][i + 1]) in TEMP;
12 end
13 C[h].clear();
14 C[h] ← TEMP;
15 return;
```

Two main distinctions that arise in the compaction process of KLL$^{\pm}$ are: (i) discarding matching pairs to free space before the push operations, and (ii) changing the minimum capacity of a compactor to at least three (from two in KLL). The first modification reduces the number of push operations by removing matched pairs which cancel each other. Later, we prove that discard operations do not introduce error and push operations can introduce at most w_h error.

The second modification changes the minimal capacity of a capacitor to three. This modification is necessary because if the minimum capacity of a compactor is two and these elements neither have the same operation nor are a matched pair, then the Conditional Compaction would retain both elements in the compactor; and hence the compactor remains full even after the compaction process. Increasing the minimum capacity of a compactor to three guarantees a reduction in size after a compaction, when the compactor becomes full. If the compactor contains matched pairs, then the discarded operation can be applied to free space. If there is no matched pair, then a full compactor with capacity three can have four possible cases in the compaction process, as shown in Table 2. Table 2 also describes the compaction strategy for each of these

[5]If compactor is not sorted, we need to keep a map while scanning the compactor to find all matching pairs.

[6]Put one pointer at beginning and one at the end, increments or decrements the pointers to find all matching pairs in a single scan.

Table 2: Compaction for a full compactor of size 3 consisting of sorted elements e_1, e_2, and e_3 and no matched pair.

3 Inserts + + +	50%: push(e_1 or e_2) and keep(e_3) 50%: keep(e_1) and push(e_2 or e_3)
2 Inserts and 1 Delete − + +	keep(e_1) and push(e_2 or e_3)
1 Insert and 2 Deletes − − +	push(e_1 or e_2) and keep(e_3)
3 Deletes − − −	50%: push(e_1 or e_2) and keep(e_3) 50%: keep(e_1) and push(e_2 or e_3)

cases, assuming elements are in the sorted order and no matching pairs. When the compactor is full, one element out of two elements with the same operations is randomly chosen and pushed to the next level, and the element not been pushed is removed.

3.4 Estimating Ranks and Quantiles

The Output Operation in Algorithm 2, summarizes the current snapshot of the sketch. It uses all the compactors inside the sketch to obtain the quantile information. First, the total weight of the sketch is calculated by summing the weights of all elements, where h is the height of the compactor and k_h is the number of elements in compactor[h] (line 9): $TotalWeight = \sum_{h=1}^{H} \sum_{i=1}^{k_h} weight(e_{h,i})$

Second, a map *ItemWeightSortedMap* of $< item, weights >$ pairs is created to contain the final aggregate weights of each element in the sketch. For each element, the aggregated weights is calculated by incrementing or decrementing the corresponding weights, based on the height of the compactor in which the element was encountered, and whether the corresponding operation was an insert or a delete.

Third, since $ItemWeightSortedMap$ is sorted in ascending order by item, map *Result* calculates the estimated rank of each item by summing through all weights of items whose values are less than or equal to itself. To transform the rank of an item into the quantile of an item, the rank information is divided by the total weight. The output operation returns a map that contains the quantile information for each item in the sketch. To estimate the rank of item, x, that is not in the output, we can use the rank of the largest element that is less than x as the estimation. For items with estimate rank less than zero or larger than the total weight, we treat their rank as zero or as the total weight since all deleted items are previously inserted and hence all ranks should be with in [0, TotalWeight].

3.5 An Illustrative Example

Assume a KLL$^{\pm}$ sketch with $k = 6$, $c = \frac{2}{3}$, the topmost compactor has capacity 6 i.e, k, and the second topmost compactor has capacity 4 i.e, ck. Items 1 through 8 are inserted and then items 7, 3, 2, and 1 are deleted. After the deletions, the multiset is left with {4,5,6,8} and their respective ranks are R(4)=1, R(5)=2, R(6)=3, R(8)=4. As illustrated in in Figure 3, the sketch needs to be compacted three times. Assume the first compaction has offset of 1; the second compaction has offset of 0; the third compaction again has offset of 1. The first conditional compaction occurs after item 6 is inserted, and pushes the 2, 4, and 6 from compactor[1] to compactor[2]. Compactor[2] maintains full capacity of 6, while the capacity of compactor[1]

Algorithm 2: Output Operation in KLL$^{\pm}$

```
1  Result = map();
2  // A map sorted in ascending Item Order
3  ItemWeightSortedMap = OrderedMap< item, weights >;
4  TotalWeight = 0;
5  for all compactor[h] in Sketch do
6      for item in compactor[h] do
7          weight = sign(item) 2^(h−1);
8          ItemWeightSortedMap[(abs(item)] += weight;
9          TotalWeight += weight;
10     end
11 end
12 PrevW = 0;
13 //traverse ItemWeightSortedMap in ascending order;
14 for < item, weights > in ItemWeightSortedMap do
15     Result[item] = (weights+PrevW)/TotalWeight;
16     PrevW += weights;
17 end
18 //result contains the quantile info for each item.
19 return Result;
```

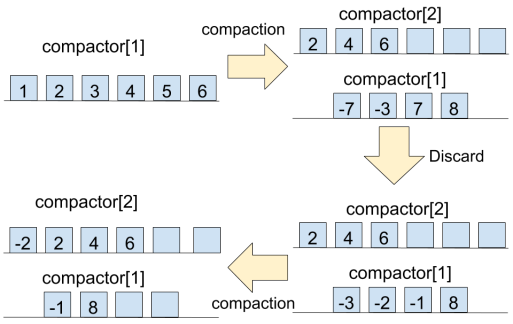


Figure 3: Items from 1 to 8 are inserted and then items 7,3,2,1 are deleted. The resulting dataset is {4,5,6,8}; KLL$^{\pm}$ estimate $\mathcal{R}(4) = 1$, $\mathcal{R}(5) = 1$, $\mathcal{R}(6) = 3$, $\mathcal{R}(8) = 4$.

is now reduced to 4. The second conditional compaction occurs after elements 7, 8, -7 and -3 are added. This compaction finds a matching pair of -7 and 7 and hence discards the pair, leaving compactor[1] with -3 and 8. The third conditional compaction occurs after elements -2 and -1 are added. This compaction pushes -2 to compactor[2], and -1 and 8 are both kept in compactor[1] since they have different signs. Based on the current sketch, the original data set are now represented as {-1, 4, 4, 6, 6, 8}. Finally, to estimate ranks, KLL$^{\pm}$ computes the weight of all $\langle abs(element), weight(element) \rangle$ pairs. Then sort based on the element's value: (1, -1), (2, 0), (4, 2), (6, 2), (8, 1), and compute their corresponding ranks: $\mathcal{R}(1) = \mathcal{R}(2) = 0$, $\mathcal{R}(4) = 1$, $\mathcal{R}(6) = 3$, and $\mathcal{R}(8) = 4$. To report $\mathcal{R}(5)$, we use the rank of largest element that is less than 5, i.e. $\mathcal{R}(5) = \mathcal{R}(4) = 1$, differs

from true $R(5)$ by 1. As a result, the output of the sketch in term of rank is: $\mathcal{R}(4) = \mathcal{R}(5) = 1, \mathcal{R}(6) = 3, \mathcal{R}(8) = 4$.

3.6 Error in Compaction

This section discusses the error introduced by Conditional Compaction. Essentially, the algorithm introduces errors when pushing one out of two elements and does not introduce any error when discarding matched elements or when retaining a pair of non-matching elements. Consider the Push(e_1,e_2) operation in $compactor[h]$ with weight w_h: at random, either e_1 or e_2 is pushed into $compactor[h+1]$ with weight w_{h+1}. Pushing elements to higher weight results in $+w_h$, 0, or -w_h error. By removing the odd or even items with equal probability, the expected error becomes zero [2].

Lemma 1. *Hoeffding's Inequality. Let $X_1, \cdots, X_n$ be independent random variables such that $-w_i \leq X_i \leq w_i$ and the expected value $E(X_i) = 0$ for $i = 1, 2, \ldots, n$. Then for any $t > 0$ we have:*

$$Pr[|\sum_{i=1}^{n} X_i| > t] \leq 2exp(-\frac{t^2}{2\sum_1^n w_i^2})$$

In Condition Compaction, Algorithm 1 discards matched pairs of inserted and deleted elements. We need to establish that this does not impact the overall error and make sure the expected error in one compaction is still zero in order to apply Hoeffding's inequality [16] to bound the total errors. Proof of Proposition 1 is presented in the extended technical report[1]

Proposition 1. *Discarding matched inserted and deleted elements within a compactor at level h during a compaction cycle does not introduce any error to the sketch's total error.*

The Conditional Compaction algorithm has three main components that can potentially introduce error: (i) Push(e_i or e_{i+1}), (ii) Keep(e_i and e_{i+1}), and (iii) Discard(-e_i and e_i). Pushing one of two elements from compactor[h] introduces $+w_h$, 0, -w_h error; keeping elements in their own compactor introduces no error; discarding the matched pairs of inserted and deleted elements introduces no error (proved in Proposition 1). Thus we can assert the expected error is still 0 when pushing even or odd index terms with equal probability, and we can apply Hoeffding's inequality to bound the probability of total error exceeding $\epsilon \cdot (I-D)$, where ϵ is the desired precision and $I - D$ is the size of the dataset after the deletions.

To show that the KLL$^{\pm}$ sketch ensures that the probability of the total error exceeding $\epsilon \cdot (I - D)$ is less than a small constant probability, δ. Let random variable $X_{i,h}$ denote the error introduced by the i^{th} compaction in $compactor[h]$ and let m_h be the number of compactions that occurred at compactor[h]. Note $X_{i,h}$ can be 1, 0, or -1. Therefore, the total error is computed as:

$$Err = \sum_{h=1}^{H} \sum_{i=1}^{m_h} w_h X_{i,h} \tag{1}$$

Since we know that $E(X_i, h) = 0$ by keeping even and odd terms with equal probability, Hoeffding's inequality can be applied:

$$P(|Err| > \epsilon(I-D)) \leq 2exp(-\frac{\epsilon^2(I-D)^2}{2\sum_{h=1}^{H}\sum_{i=1}^{m_h} w_h^2}) \leq \delta$$

Where δ is a small constant probability. δ denotes the maximum failure probability; hence, when δ is small, the algorithm has high confidence for success.

3.7 Space Bound

In this section, we analyze the space bound and approximation guarantees for KLL$^{\pm}$ with an array of H compactors. The minimum capacity of compactors is three to handle both insert and delete operations. To guarantee any compactor will have capacity greater than or equal to three, a compactor at height h has capacity of $max(\left\lfloor c^{H-h}k \right\rfloor, 3)$. Hence, as the total height H increases, there will be a stack of bottom compactors with capacity three. In particular, for compactors with small h such that $\left\lfloor c^{H-h}k \right\rfloor \leq 3$,the compactors will have capacity of three.

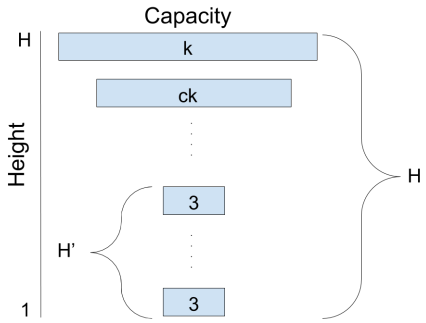


Figure 4: H' denotes the height of the bottom compactors with size 3, and H is the height at the highest level.

Assume the KLL$^{\pm}$ sketch has $k \geq 4$ and $c \in (0.5, 1)$ with $c_h = max(\left\lfloor c^{H-h}k \right\rfloor, 3)$, and the incoming data have I insertions and D deletions such that $D \leq (1 - 1/\alpha)I$. Let $r = \frac{D}{I}$ be the rate of deletions such that $r \leq (1 - \frac{1}{\alpha})$ ensuring that the incoming updates have bounded deletions compared to insertions. H is the height of the compactor at the highest level; H' is the height of the bottom compactors with capacity three (see Figure 4) such that $H' \leq H - 1$.

Because each compactor will undergo compaction when it becomes full, every compactor of capacity three contains at most two items, and hence the total weight of these bottom compactors is:

$$W_{bottom} \leq \sum_{h=1}^{H'} 2w_h = \sum_{h=1}^{H'} 2^h \leq 2^{H'+1}$$

The total weights of the top compactors is:

$$W_{top} \leq \sum_{h=H'+1}^{H} (k_h - 1)w_h \leq (k-1)2^H$$

Combining the total weights of the bottom H' and top $H - H'$ compactors results in an upper bound on the number of items in the dataset: $I - D \leq 2^{H'+1} + (k-1)2^H \leq k2^H$. We can then bound the total number of items:

$$n = I + D = (1+r)I \leq k2^H \frac{1+r}{1-r} \tag{2}$$

Since topmost $compactor[H]$ exists, it implies that $compactor[H-1]$ has been compacted. Thus , $n \geq c_{H-1}w_{H-1} \geq ck2^{H-2} = k2^H(c/4)$. By moving H to the left hand side and n to the right hand side:

$$H \leq log_2(4n/ck) = log_2(n/ck) + 2$$

The *keep* and *discard* operation introduce no error as shown in Proposition 1, and only the push operation will introduce at most $\pm w_h$ error. Also, because the conditional compaction Algorithm 1 will perform no compaction if there are any discard operations, the push operations are performed on compactor[h] that contains $k_h = c_h$ elements. Then let m_h denote the number of compactions occurred for compactor at height h, and the total error introduced in the sketch is upper bounded by the number of compactions: $Err = \sum_{h=1}^{H}\sum_{i=1}^{m_h} w_h X_{i,h}$ where $X_{i,h}$ is a random variable denote the error introduced by the i^{th} compaction in which $E(X_{i,h}) = 0$ and $|X_{i,h}| \leq 1$, shown in Equation 1. We can bound the number of compactions, m_h, at height h, where each compaction is performed on $k_h = c_h$ elements with weight w_h as $m_h \leq \frac{n}{c_h w_h} \leq \frac{2n}{k2^H}(2/c)^{H-h}$ and by substituting in Equation 2, $m_h \leq 2\frac{1+r}{1-r}(2/c)^{H-h}$. Note in this upper bound, we did not consider discard operations; discard operations only reduce the number of compactions. We now apply Hoeffding's inequality to bound the probability for compactors to introduce more than $\epsilon \cdot (I-D)$ error:

$$P(|Err| > \epsilon(I-D)) \leq 2exp(-\frac{\epsilon^2(I-D)^2}{2\sum_{h=1}^{H}\sum_{i=1}^{m_h} w_h^2}) \leq \delta$$

The denominator can be expanded to $\sum_{h=1}^{H}\sum_{i=1}^{m_h} w_h^2 = \sum_{h=1}^{H} m_h w_h^2 \leq \frac{1+r}{1-r}\frac{16n^2}{c(2c-1)k^2}$, where $r = \frac{D}{I} \leq (1-1/\alpha)$. Let $C = \frac{c(2c-1)}{32}$,

$$P(|Err| > \epsilon(\frac{1-r}{1+r}n)) \leq 2exp(-(\frac{1-r}{1+r})^3 C\epsilon^2 k^2) \tag{3}$$

Since $0 \leq r \leq (1-\frac{1}{\alpha})$, $1+r \leq 2-\frac{1}{\alpha}$, $1-r \geq \frac{1}{\alpha}$, and hence $\frac{1-r}{1+r} \geq \frac{1}{2\alpha-1}$. Setting $k = (2\alpha-1)^{1.5}/(\epsilon\sqrt{C})\sqrt{\ln(2/\delta)}$ suffices to bound the failure probability by δ, and the total space used is:

$$\sum_{h=1}^{H} c_h \leq 3H + k\sum_{h=1}^{H} c^{H-h} = O(k + log(n/k)) \tag{4}$$

Theorem 1. *In the bounded deletion model $D \leq (1-1/\alpha)I$, there exists a data sketch that computes an ϵ approximation for the rank of a single item with probability $1-\delta$ whose space complexity is $O(\frac{\alpha^{1.5}}{\epsilon}\sqrt{log(1/\delta)} + log(\frac{\epsilon n}{\alpha^{1.5}}))$.*

In order to have ϵ approximations for all items, the failure probability need to be decreased into $\epsilon\delta$, bounding the failure probability for approximating a set of $O(1/\epsilon)$ items. Thus, to solve all quantile approximation, the space complexity becomes:

$$O(\frac{\alpha^{1.5}}{\epsilon}\sqrt{log(1/\epsilon\delta)} + log(\frac{\epsilon n}{\alpha^{1.5}})$$

Note that the space complexity includes $log(\epsilon n)$ term and this is due to the stack of compactors with size 3. We can further reduce the space complexity by replacing the bottom compactors with samplers as discussed in the next section.

4 SAMPLER AND MERGEABILITY

In this section, we describe how to incorporate the sampler from the original KLL [20] to further reduce the space bound while still maintaining full mergeability. We also discuss the merge operation between two KLL$^{\pm}$ sketches to construct a combined sketch over the union of the underlying datasets, and, in the end, propose a new algorithm that leverages KLL$^{\pm}$ sketches to maintain quantile approximations in fixed-size sliding window.

4.1 Sampler-based Bottom Compactors

From the Conditional Compaction Algorithm, when a bottom compactor is full, an inserted item or a deleted item is randomly chosen out of two insertions or two deletions. This is equivalent to replacing the bottom H' compactors with two *samplers* (one for insertions and one for deletions) to simulate the bottom compactors and consume O(1) space. Karnin et al. [20] introduced the mergeable sampler and prove the correctness of the sampling schema.The sampler has an associated height h, and h increases over more and more inputs. The sampler outputs an item with weight 2^h as input to the compactor of level $h+1$. Therefore, the sketch only contains compactors with heights greater than H' (from $H'+1$ to H).

More precisely [20], the sampler has an associated height h and stores one item with weight at most 2^{h-1}. An update is processed as follows: Let $w_{internal}$ be the weight of the internal item stored in the sampler and w_{new} be the weight of the newly arriving item. This w_{new} is 1 for items from the input; However, when merging two sketches (as in Section 4.2), w_{new} may represent the item weight from another sampler. If $w_{internal}+w_{new} < 2^h$, the sampler stores the new item with probability $w_{new}/(w_{internal}+w_{new})$. If $w_{internal}+w_{new} = 2^h$, the sampler pushes the stored item into $compactor[h+1]$ and resets $w_{internal}$ to zero. If $w_{internal}+w_{new} > 2^h$, the sampler pushes the item that has larger weight with probability $max(w_{internal}, w_{new})/2^h$; the sampler also updates the stored item to the item that has smaller weight and resets $w_{internal}$ to $min(w_{internal}, w_{new})$. The last case is necessary to support merge operations. For a sampler at height h, this approach takes an input of W items where $2^{h-1} < W \leq 2^h$. With probability $W/2^h$, it pushes one of the observed items chosen at random and otherwise pushes nothing. Lemma 2 in [20] establishes that the sampler's push operation introduces an unbiased error of $2^h Y_{h,i}$ where $Y_{h,i}$ is a random variable with $E(Y_{h,i}) = 0$ and $|Y_{h,i}| \leq 1$.

In KLL$^{\pm}$, the bottom compactors of capacity 3 are replaced by two samplers, one for insertions and one for deletions. The insert and delete samplers share the same height thus ensuring that when an item is output to the compactor, the item will have the same weight independent of whether it is an insertion or a deletion. Since there are total I insertions and D deletions, as the shared samplers' height increase the insert sampler at height h at most pushes $\frac{I}{2^h}$ elements of weight 2^h to the first compactor and similarly the delete sampler of height h at most pushes $\frac{D}{2^h}$ elements of weight 2^h to the first compactor. Hence there are at most $\frac{n}{2^h}$ elements pushed for sampler height h. The total errors introduced are the sum of all the errors produced by the push operations from both of these two samplers. Given the probability of the samplers total error exceeding $\epsilon(I-D)$ should be less than or equal to δ, setting

$k = (2\alpha - 1)^{1.5}/\epsilon\sqrt{log(1/\epsilon\delta)}$ is sufficient. The proof is presented in the extended technical report [1].

Theorem 2. *In the bounded deletion model, there exists a data sketch that computes an ϵ approximation for the rank of a single item with probability $1-\delta$ whose space complexity is $O(\alpha^{1.5}/\epsilon\sqrt{log(1/\epsilon\delta)})$.*

The total sketch size becomes $\sum_{h=H'}^{H} c_h = \sum_{h=H'}^{H} c^{H-h}k \leq \frac{k}{1-c} = O(k)$, $c \in (0.5, 1)$, and $O(k) = O(\alpha^{1.5}/\epsilon\sqrt{log(1/\delta)})$. Moreover, fixing the capacity of the top $O(loglog(1/\delta))$ compactors to k as in KLL[20] can be applied to reduce the KLL$^{\pm}$ sketch size to $O(\frac{\alpha^{1.5}}{\epsilon}log^2 log\frac{1}{\epsilon})$ and still remain mergeable. In KLL$^{\pm}$ with input size of $I + D$, the probability of total error in rank estimation exceeding $\epsilon(I - D)$ should be less than or equal to δ. The proof is presented in the extended technical report [1]. Lastly, Karnin et al. [20] suggest to replace the top compactors by the GK sketch to obtain an asymptotically optimal space bound while sacrificing mergeability. However, since the GK sketch does not support deletes, this optimization is not applicable in KLL$^{\pm}$.

4.2 Mergeability

KLL$^{\pm}$ sketches with the same deletion upper bound α are fully mergeable, and the merge operation is described in Algorithm 3. The mergeability of KLL$^{\pm}$ sketches is desirable for distributed settings. The samplers in Section 4.1 are designed to support merge operations by supporting weighted updates. To merge two sketches $< S_A, S_B >$, first combine the samplers from both sketches. Assume S_A's samplers have height h_A and S_B's samplers have height h_B such that $h_A \geq h_B$. S_B feeds its stored insert and delete items from its insert and delete samplers, respectively, with their internal weights into S_A's samplers, and then all compactors with height less than or equal to h_A in S_B feed their items into the appropriate sampler in S_A with proper weights, calculated using their height ($w_h = 2^{h-1}$). Compactors with height larger than h_A are appended to the same height compactors in S_A. Lastly, each compactor that contains elements more than its capacity is compacted and the new maximal H is computed after all compactions are completed i.e., using the combined length of these two sketches.

4.3 Fixed-Size Sliding Window

We consider a potential use-case for KLL$^{\pm}$ sketch to maintain ϵ-approximate quantiles in a fixed-size sliding window over append only datasets, e.g., transaction logs. In the fixed-size sliding window, the window boundary synchronously shifts over the data, and the sketch reports the quantile information for the most recent w items. In the fixed-size sliding window, the input items are insertions, and the expired elements (elements except the recent w items) need to be deleted. Since the KLL$^{\pm}$ sketch only tolerates bounded deletions, it cannot directly support such a sliding window setting as the ratio between deletions and insertions will eventually approach to one. However, we observe that by partitioning the input data into *three* overlapping blocks, we can ensure the deletions are bounded for each block. The three blocks or data partitions are termed as *active*, *under-construction*, and *backup* sketches as shown in Figure 5. Each block can be represented by one KLL$^{\pm}$ sketch.

Initially all sketches contain no elements. When the active sketch has inserted $\frac{w}{2}$ elements, new incoming elements are inserted into both the active sketch and backup sketch. Once the active sketch contains w elements and the backup sketch contains $\frac{w}{2}$ elements, incoming elements are inserted into the backup and under-construction sketches and the outdated (oldest) elements are deleted from the active sketch. When the backup sketch has w elements, the active sketch, with $\frac{w}{2}$ deletions, becomes expired. The backup sketch now becomes the new active sketch and the under-construction sketch becomes the new backup sketch. Deleting the expired (previously active) sketch saves space, the freed space is used to construct the new under-construction sketch.

At any point in time, the quantile information is reported of the most recent w elements by merging the active and under-construction sketches, i.e, the yellow and blue blocks shown in Figure 5. Also note that, since deletions only arrive when the sketch has already inserted w items, all sketches will have a deletions to insertions ratio of $r = \frac{d}{w}$, and because a sketch expires after at most $\frac{w}{2}$ deletions, i.e, $d \leq \frac{w}{2}$, $r \leq \frac{0.5w}{w} \leq \frac{1}{2}$ and hence setting $\alpha = 2$ is

Algorithm 3: Merge Two KLL$^{\pm}$ Sketches

```
if S_A.SamplerHeight < S_B.SamplerHeight then
    swap(S_A, S_B)
S_A,InsertSampler.update(S_B,InsertSampler);
S_A,DeleteSampler.update(S_B,DeleteSampler);
for all compactor[h] in S_B do
    if h ≤ S_A.SamplerHeight then
        for item in compactor[h] do
            if insert(item) then
                S_A,InsertSampler.update(item, 2^(h-1));
            if delete(item) then
                S_A,DeleteSampler.update(item, 2^(h-1));
        end
    else
        S_A.compactor[h].concatenate(compactor[h]);
end
S_A.compaction();
return S_A;
```

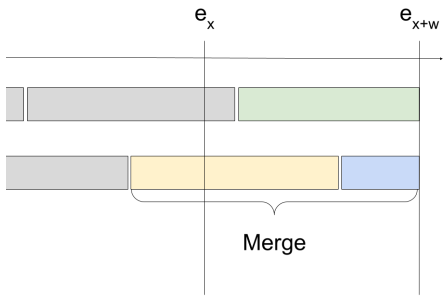


Figure 5: Grey blocks are expired sketches; yellow block is the current *active* sketch; blue block is the current *under-construction* sketch; green block is the current *backup* sketch.

sufficient to ensure that $r \leq \frac{1}{2}$. In fact, the α can also be changed to other values; however, the number of backup sketches will also change accordingly. Thus, we have demonstrated a randomized algorithm to maintain quantile summary in fixed-size sliding window with three $O(\frac{\alpha^{1.5}}{\epsilon} log^2 log\frac{1}{\epsilon})$ sketches and $\alpha = 2$.

5 EXPERIMENTS

This section experimentally evaluates KLL$^{\pm}$ for datasets that experience an input data stream that consists of both insertions and bounded deletions of elements. KLL$^{\pm}$ is the first quantile sketch algorithm to operate in the bounded deletion model and the experiments aim to identify the overhead incurred in accounting for bounded deletes compared to other sketches:

- **KLL**: Since KLL is insertion-only, the input stream only inserts those elements that are left after all the deletions.
- **Two-Parallel KLL**: Two independent KLL sketches: one for insertions and one for deletions; then aggregate their estimations to approximate quantiles.
- **DCS**[7]: A universe-driven sketch that assumes a bounded universe to tolerate an arbitrary number of deletions.

5.1 Experimental Setup

We set $c = \frac{2}{3}$ for all experiments, and implemented KLL$^{\pm}$ by enhancing the Python 3.7.6 code-base of the KLL sketch algorithm presented in [18, 20]. The changes incorporated in our implementation are: (i) the minimum capacity of a compactor is increased to 3; (ii) bottom compactors of capacity 3 are replaced by 2 samplers, one for insertions and one for deletions; and (iii) the compaction algorithm is modified to implement our new Conditional Compaction algorithm. The metric for accuracy measurement is the Kolmogorov-Smirnov divergence [5], the *maximum* deviation among all quantile queries, a measurement widely used to perform comparisons between CDFs with different distributions [18]. For each experiment, the maximum errors are averaged over 5 independent runs.

5.2 Data Sets

The experimental evaluation is conducted using both synthetic and real world data sets consisting of items that are inserted and deleted. For the synthetic data, we consider three different distributions:

- **Uniform Distribution**: The insertions are randomly generated from a discrete uniform distribution, and the deletions are uniformly chosen from the insertions.
- **Zipf Distribution**: The elements drawn from bounded universe and the frequencies of elements follow the Zipf Law [29], in which the frequency of element with rank R: $f(R, s) = \frac{constant}{R^s}$ where s indicate skewness. Deletions are uniformly chosen from the insertions.
- **Binomial Distribution**: The elements are generated according to the binomial distribution with parameters n and p where p is the probability of success in n independent Bernoulli trials.

In addition to the synthetic data sets, we used the following real world Wiki dataset[8]:

- **Page View Statistics for Wikimedia (Wiki)** [23]: This is an extensive data set consisting of page count files from 2007 to 2016. The experiments use the 2016 page count files, which include 8 months of <*projectname, pagename, #requests, #bytes*> tuples. The data are aggregated by day and within each day, data are sorted on *projectname* and *pagename*. In the experiments, deleted items are random uniformly chosen from the inserted items and each update is a concatenation of *projectionname* and *pagename* and the comparison model is lexicographic

We also conducted experiments by exploring two additional properties of the data sets:

- **Sorted Dataset**: Input is sorted in descending order such that insertions arrive before deletions; The deletions are uniformly chosen from the insertions.
- **Shuffled Dataset**: The insertions are randomly shuffled and the deletions are also randomly shuffled and uniformly chosen from the insertions.

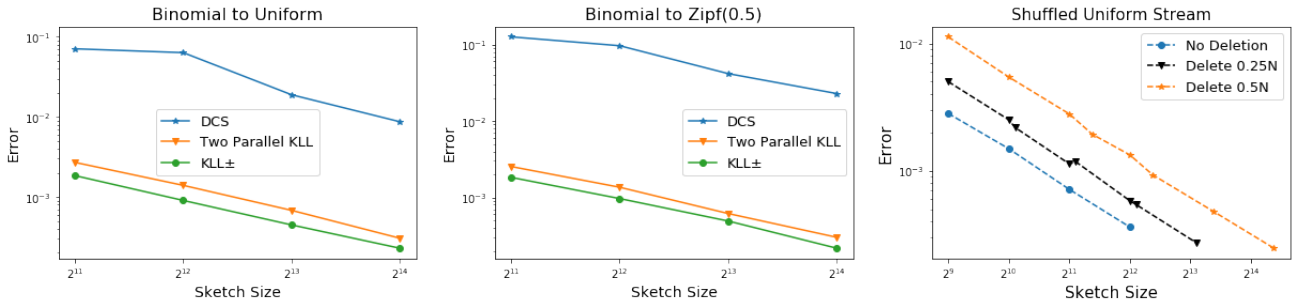
5.3 Evaluation

The y-axis depicts the average of maximum quantile error over 5 independent runs: lower y-axis values indicate better accuracy. Most of the following experiments evaluate the error value in approximating quantiles while increasing the sketch size in which the x-axis denotes the sketch size. Except for Section 5.3.6, we assume all insertions arrive before any deletions into the sketch. This input pattern is in fact an adverse pattern as the discard operation will find less matched pairs.

5.3.1 KLL$^{\pm}$ vs. Two-Parallel KLL vs. DCS. This experiment compares the accuracy among KLL$^{\pm}$, two-parallel KLL method, and DCS under the same memory budget. The experiment measures the quantile estimation when the underlying dataset is entirely changed: We first insert one million elements from the binomial distribution with parameter of B(2^{16}, 0.5) in which the elements should be densely centered at value 2^{15}. Then another one million elements are inserted from a uniform distribution in Figure 6(a), or from a zipf distribution with skewness factor of 0.5 (moderate skew) in Figure 6(b). Both the uniform distribution and zipf distribution assume a bounded universe where $|U| = 2^{16}$, and all inputs are randomly shuffled. Finally, all elements from the binomial distribution are deleted. Since the total insertions are two millions and the total deletions are one million, the delete:insert ratio is 0.5.

Figure 6(a) and (b) show that both data driven sketch approaches, KLL$^{\pm}$ and Two Parallel KLL, perform significantly better than the universe driven DCS sketch. Although the maximum error of KLL$^{\pm}$ and Two Parallel Method decreases as the sketch size increases, KLL$^{\pm}$ has less maximum error across all sketch sizes. This finding is expected since KLL$^{\pm}$ makes a best effort to apply discard operations thus catching cancellations early on which reduces the number of compactions, whereas in the two-parallel KLL method each sketch has no knowledge about the other and accumulates all the errors. We observe that DCS performs worse on the zipf distribution compared to a uniform distribution. The skewness in the input data distribution affects the performance of DCS and decreases its accuracy as skewness increases. This observation for universe-driven

[7] See chapter 4 in [10] for implementation details.
[8] https://dumps.wikimedia.org/other/pagecounts-ez/



(a) (b) (c)

Figure 6: Comparison of KLL$^{\pm}$ with Two Parallel KLL and DCS when the underlying data distribution is entirely changed from Binomial to (a) Uniform and to (b) Zipf (.5). (c) KLL$^{\pm}$ accuracy with different delete:insert ratios.

DCS sketch is in consistent with the theoretical expectations [28]. On the other hand, the performance of both data-driven sketches, KLL$^{\pm}$ and two-parallel KLL, is not affected by the skew in the input.

5.3.2 Error correlations: deletion ratio & sketch size. We experimentally verified that the delete to insert ratio on the data set affects the accuracy of the sketch. By scaling the sketch size by a factor of $(2\alpha - 1)^{1.5}$ according to Section 4, KLL$^{\pm}$ can keep the errors to the same level compare with errors from KLL with no deletions, as shown in Figure 6 (c). This scaling factor depends on the delete:insert ratio r. For example, when $r = 0.5$, we set $\alpha = 2$ to increase the KLL$^{\pm}$ sketch size. Note $(2\alpha - 1)^{1.5}$ is actually an upper bound, as in the proof we make no assumptions on the number of discard operations which is affected by the input pattern. Hence. we expect KLL$^{\pm}$ with scaled sketch sizes to perform no worse than the original KLL with no deletions.

The input random shuffled stream contains a million insertions drawn from a uniform distribution. The deletions are uniformly chosen from the insertions. This experiment shows the interplay of space and accuracy when the delete:insert ratio r in the input stream increase, and verifies the theoretical claim made in Section 4.1 that changing the sketch size in accordance with r keeps the error roughly constant. Figure 6 (c) shows that the higher delete:insert ratio leads to larger errors, while increasing the sketch size with the increase in deletions ensure the accuracy is no worse than the original KLL with no deletions. For instance, when KLL uses $k = 512$, the corresponding average maximum error value is 0.0028. For $r = 0.25$, scaling KLL$^{\pm}$ space with $\alpha = \frac{4}{3}$, k increased to 1102 and the corresponding averaged maximum error value is 0.0022. Similarly for $r = 0.5$, scaling KLL$^{\pm}$ space with $\alpha = 2$, k increases to 2661 and the corresponding averaged maximum error value is 0.0019. This verifies the theoretical expectation that scaling the sketch size according to the delete:insert ratio keeps the error roughly constant.

5.3.3 Different Data Distributions. In this experiment, we further demonstrate the trade-off between space and accuracy on different data distributions. Figures 7(a) through 7(c) depict the maximum quantile errors with increasing sketch size for synthetic datasets *uniform* and *zipf* and for real world data *Wikimedia page view statistics*. The experiment also plots the behaviour of KLL$^{\pm}$ and KLL when the input steam is sorted vs. shuffled. While the universe-driven DCS's accuracy is independent from the input pattern and the number of deletions [28], the DCS plots in Figure 7(a) and (b) are used as a comparison reference for KLL$^{\pm}$ with large delete:insert ratio, namely 90%. In Figure 7 (a) and (b), note that even when a significant number of items are deleted and the delete:insert ratio becomes 90%, KLL$^{\pm}$ still performs well compare to DCS.

The behavior of all sketches is consistent across all data distributions: larger space leads to smaller errors. The experiment shows that KLL$^{\pm}$ behaves worse on shuffled streams compared to sorted streams, across all types of data distributions. This finding is consistent with [18] where it is observed that the randomness within the stream affects the accuracy of the sketch.

5.3.4 Update Time. In this section, we experimentally compare the update time among KLL, KLL$^{\pm}$, and DCS sketches. The items in the input stream are shuffled uniform distribution of $|U| = 2^{16}$. We also include the update time of DCS for $U = 2^{32}$. All sketches share the same space budget. In Figure 8(a), the y-axis is the update-time and the x-axis is the stream length, smaller y-value implies faster update time per item. The result is aligned with our expectation, namely that KLL$^{\pm}$ incurs slightly more time than KLL using the same memory budget, because KLL$^{\pm}$ makes a best effort to apply discard operations before compaction. In DCS, on the other hand, the universe size affects the update time as a larger universe size leads to slower update time per item.

5.3.5 Error Sensitivity to stream length. This experiment demonstrates that the quantile approximation error of KLL$^{\pm}$ is independent of the input stream length, as shown in Figure 8(b). In this figure, the x-axis denotes the input stream length in which *N =(I+D)*, where N is the total number of operations in the stream, and r determines the percentage of elements that are deleted. For this experiment,KLL and KLL$^{\pm}$ both uses parameter $k = 1024$. The insertions are randomly shuffled items from uniform distribution and the deletions are randomly chosen from the previously inserted items. This experiment highlights that, for a given delete:insert ratio, the error remains roughly constant with the increase in stream length and hence is independent of the input stream size.

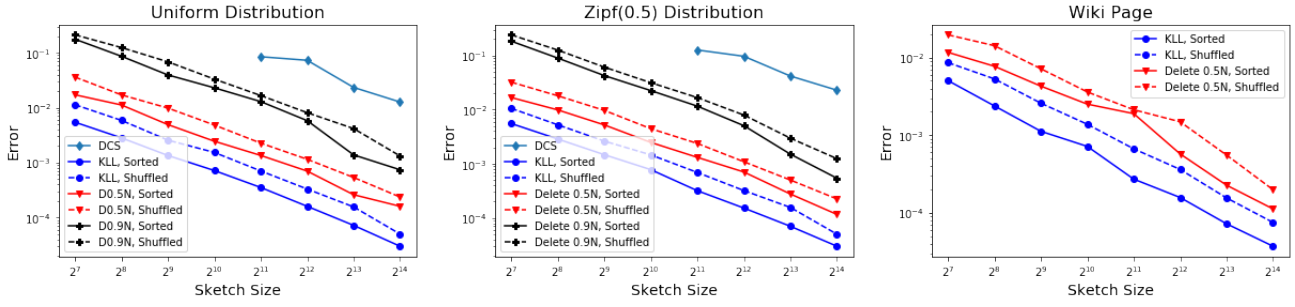


Figure 7: Trade-off between space and accuracy on different data distributions

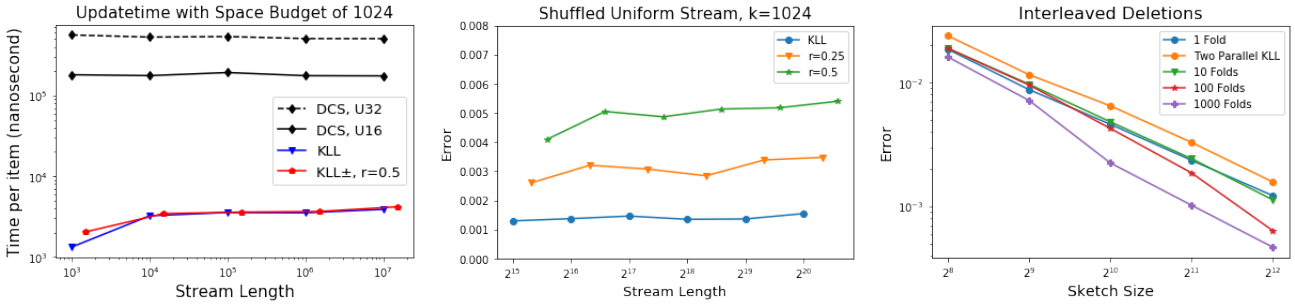


Figure 8: (a) Update time per item in KLL, KLL$^{\pm}$, and DCS with universe size of 2^{16} and 2^{32}; (b) Maximum error of shuffled uniform streams with varying stream length; (c) Interleaved deletion pattern.

5.3.6 Interleaved Deletions. All prior experiments were performed under the assumption that the input consists of insertions first, followed by deletions. In this experiment, we explore the performance of KLL$^{\pm}$ with inputs consisting of interleaved inserts and deletes. The input stream is a shuffled uniform distribution. The input is divided into a number of *folds*, such as 1, 10, 10^2, and 10^3, in which 1 fold means the whole input stream consist of a single pair of insertions and deletions i.e., <all inserts, all deletes>. Similarly, 10 folds means the stream consists of 10 pairs of $< \frac{I}{10}, \frac{D}{10} >$ substreams where I and D are the total number of inserts and deletes. For each pair of <inserts, deletes>, the deletes are uniformly chosen from its inserts. More folds imply deleted items are closer to their corresponding inserted items. In Proposition 1, we showed that discard operations introduce no error. When deleted items are closer to the inserted items, we expect more discard operations leading to fewer errors. Figure 8(c) shows that when number of folds are small, the averaged maximum error is higher, and when number of folds increases, KLL$^{\pm}$ improves its performance by applying more discard operations, reducing the overall averaged maximum error. On the other hand, the performance of Two Parallel KLL sketches will not improve in the interleaved deletion pattern, as the insertions and deletions are separately managed. When the insertions and deletions are mixed, the maximum error of the KLL$^{\pm}$ decreases which reflects more realistic real world scenarios.

6 CONCLUSIONS

Quantile approximations have an important role in both research as well as real world systems. Many algorithms have been proposed to approximate quantiles for the insertion-only and the turnstile models. In this work, we propose a data-driven algorithm KLL$^{\pm}$ to approximate quantiles in the bounded deletions model. To our knowledge, this is the first work to account for bounded deletions for approximating quantiles. The experimental evaluations of KLL$^{\pm}$ highlight that the accuracy provided by the quantile approximations of KLL$^{\pm}$ is significantly better than the state of the art DCS sketch even when a significant fraction (90%) of elements are deleted. We also demonstrate that the accuracy of KLL$^{\pm}$ is not affected by the underlying distribution of the data which is not the case with the universe-driven sketch such as DCS. Furthermore, the experiments highlight that KLL$^{\pm}$ has much faster update times compared to DCS. These characteristics of KLL$^{\pm}$ makes it a practical choice for real world applications. Finally, we also demonstrated that the deletion property of KLL$^{\pm}$ can be leveraged for maintaining quantile estimation in fixed-sized sliding windows over datasets.

ACKNOWLEDGMENTS

Thanks to anonymous reviewers for their valuable feedback. Sujaya Maiyya is partially supported by an IBM PhD Fellowship. This work is funded in part by NSF grants CNS-1703560 and CNS-1815733.

REFERENCES

[1] 2021. KLL$^{\pm}$ Technical Report. https://sites.cs.ucsb.edu/~sujaya_maiyya/assets/papers/KLL_Delete.pdf.

[2] Pankaj K Agarwal, Graham Cormode, Zengfeng Huang, Jeff Phillips, Zhewei Wei, and Ke Yi. 2012. Mergeable summaries. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*. 23–34.

[3] Arvind Arasu and Gurmeet Singh Manku. 2004. Approximate counts and quantiles over sliding windows. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 286–296.

[4] Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.

[5] Francesco Paolo Cantelli. 1933. Sulla determinazione empirica delle leggi di probabilita. *Giorn. Ist. Ital. Attuari* 4, 421-424 (1933).

[6] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*. Springer, 693–703.

[7] Zhiwei Chen and Aoqian Zhang. 2020. A Survey of Approximate Quantile Computation on Large-Scale Data. *IEEE Access* 8 (2020), 34585–34597.

[8] Graham Cormode, Theodore Johnson, Flip Korn, Shan Muthukrishnan, Oliver Spatscheck, and Divesh Srivastava. 2004. Holistic UDAFs at streaming speeds. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. 35–46.

[9] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[10] Graham Cormode and Ke Yi. 2020. *Small Summaries for Big Data.* Cambridge University Press.

[11] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm.

[12] Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin J Strauss. 2002. How to summarize the universe: Dynamic maintenance of quantiles. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 454–465.

[13] Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record* 30, 2 (2001), 58–66.

[14] Michael B Greenwald and Sanjeev Khanna. 2016. Quantiles and equi-depth histograms over streams. In *Data Stream Management*. Springer, 45–86.

[15] Xiaojia Guo, Kenneth C Lichtendahl, and Yael Grushka-Cockayne. 2018. *Quantile Forecasts of Product Life Cycles Using Exponential Smoothing*. Harvard Business School.

[16] Wassily Hoeffding. 1994. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*. Springer, 409–426.

[17] Piotr Indyk. 2004. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. 373–380.

[18] Nikita Ivkin, Edo Liberty, Kevin Lang, Zohar Karnin, and Vladimir Braverman. 2019. Streaming Quantiles Algorithms with Small Space and Update Time. *arXiv preprint arXiv:1907.00236* (2019).

[19] Rajesh Jayaram and David P Woodruff. 2018. Data streams with bounded deletions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 341–354.

[20] Zohar Karnin, Kevin Lang, and Edo Liberty. 2016. Optimal quantile approximation in streams. In *2016 ieee 57th annual symposium on foundations of computer science (focs)*. IEEE, 71–78.

[21] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. 1998. Approximate medians and other quantiles in one pass and with limited memory. *ACM SIGMOD Record* 27, 2 (1998), 426–435.

[22] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. 1999. Random sampling techniques for space efficient online computation of order statistics of large datasets. *ACM SIGMOD Record* 28, 2 (1999), 251–262.

[23] Domas Mituzas. 2013. Page view statistics for Wikimedia projects.

[24] J Ian Munro and Mike S Paterson. 1980. Selection and sorting with limited storage. *Theoretical computer science* 12, 3 (1980), 315–323.

[25] Viswanath Poosala, Peter J Haas, Yannis E Ioannidis, and Eugene J Shekita. 1996. Improved histograms for selectivity estimation of range predicates. *ACM Sigmod Record* 25, 2 (1996), 294–305.

[26] Viswanath Poosala, Yannis E Ioannidis, et al. 1996. Estimation of query-result distribution and its application in parallel-join load balancing. In *VLDB*, Vol. 96. 3–6.

[27] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. 2004. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*. 239–249.

[28] Lu Wang, Ge Luo, Ke Yi, and Graham Cormode. 2013. Quantiles over data streams: an experimental study. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 737–748.

[29] George Kingsley Zipf. 2016. *Human behavior and the principle of least effort: An introduction to human ecology.* Ravenio Books.