Quantile Sketch Algorithms

Levente Birszki

Supervisors:

Gábor rétvári
Balázs Vass

Eötvös Loránd University

2025 January 09

Motivation

- **Financial Analysis**: Quickly estimating value at risk (VaR) and other financial metrics from large volumes of transaction data.
- **Network Monitoring**: Analyzing latency, bandwidth usage, and other network metrics in real-time.
- **Database Systems**: Enhancing query performance by maintaining approximate summaries of large tables.

Basic concepts

Definition (sketch)

A *sketch* $S(X)$ of some data set X with respect to some function f is a compression of X that allows us to compute, or approximately compute $f(X)$ given access only to $S(X)$.

Definition (rank)

Given an x element from the input stream. $r(x)$, the *rank* of x is the number of elements smaller or equal than x in the sorted input.

Definition (quantile)

The q-quantile for $q \in [0, 1]$ is the element x_q, whose rank is $\lceil qn \rceil$.

Why sketches?

- **Scalability**: Traditional methods for computing quantiles can be impractical for large datasets due to high computational and storage costs.
- **Stream Processing**: In many real-time applications, data arrives in streams, and it's crucial to compute quantiles without storing the entire dataset.

Definition (rank error)

An element $\tilde{x}_q$ is an ε-approximate q-quantile if $|r(x_q) - r(\tilde{x}_q)| \leq \varepsilon n$. This also known as rank error.

Former results

Definition (single quantile approximation problem)

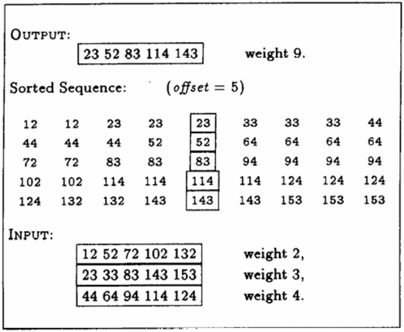
In the *single quantile approximation problem*, given an $x_1, \ldots, x_n$ input stream, q, ε and δ. Construct a streaming algorithm, which computes an ε-approximate q-quantile with probability at least $1 - \delta$.

Publication	Algorithm	Space Complexity	Mergeability	quantile type
2001	GK-sketch	$O\left(\frac{1}{\varepsilon}\log(\varepsilon n)\right)$	no	all
2004	q-digest	$O(\frac{1}{\varepsilon}\log u)$	yes	all
2016	KLL	$O(\frac{1}{\varepsilon}\log^2\log\frac{1}{\delta})$	yes	singe
2016	KLL	$O(\frac{1}{\varepsilon}\log^2\log\frac{1}{\delta\varepsilon}))$	yes	all
2017	FO	$O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$	no	all
2019	SweepKLL	$O(\frac{1}{\varepsilon}\log\log\frac{1}{\delta}))$	no	single
2019	SweepKLL	$O(\frac{1}{\varepsilon}\log\log\frac{1}{\delta\varepsilon}))$	no	all

MRL-sketch framework

b buffers, each can store k elements. each buffer X has a $w(X)$ weight. Three operations:

- *New*(X): Fills an empty buffer from input, $w(X) := 1$.
- *Collapse*$(X_1, X_2, \ldots, X_c)$:



- *Quantile*(q): After collapse, returns $X[q \cdot k]$

Merging policies

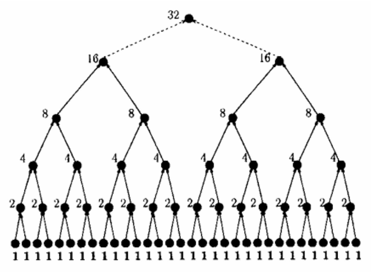


Figure: MP-sketch for $b = 6$ buffers.

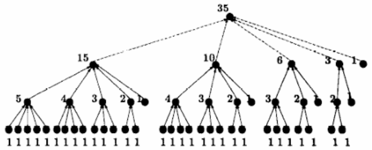


Figure: MRL-sketch for $b = 5$ buffers.

GK-sketch

GK-sketch uses only one array and stores tuples:

- v_i: one of the elements seen so far.
- $g_i = r_{\min}(v_i) - r_{\min}(v_{i-1})$.
- $\Delta_i = r_{\max}(v_i) - r_{\min}(v_i)$.

we can use these values to obtain lower and upper bounds for a rank of an element.
Sometimes we merge adjacent elements.

KLL-sketch

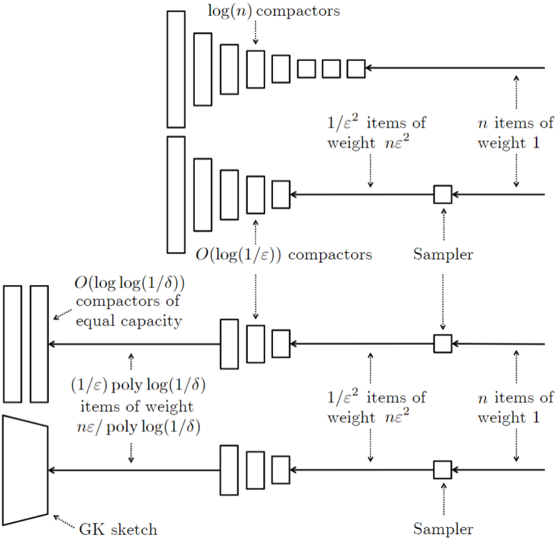


Figure: KLL step-by-step

Runtime of MP-sketc

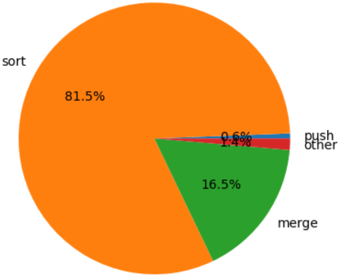


Figure: Operation proportions to the runtime of MP-sketch. $n = 10^5, \varepsilon = 0.001, b = 6, k = 3125$

Our contribution

- Improve performance using its own predictions. If we have a quantile sketching algorithm, we can use it, to approximate the CDF.
- The slowest part is to sort the buffers on the first level, so try to improve this.
- In every insertion, ask the top level bucket what is the rank of that element. Then try to insert it into the buffers corresponding position.
- To this last part, we used a clever sorting algorithm (Bai and Coester, 2024) that utilizes predictors.

Sorting with prediction

Algorithm 1 Sorting with prediction

Input: $A = a_1, \ldots, a_n$, predictor $\hat{p}$

BucketSort($A, \hat{p}$)

$T \leftarrow$ an empty scapegoat tree with finger.

$N \leftarrow n$

for $i = 1, \ldots, n$ **do**

Insert a_i into T.

end for

return nodes in T in sorted order (via inorder traversal)

Theorem

Algorithm 1 sorts an array within $O\left(\sum_{i=1}^{n} \log(\eta_i^{\Delta} + 2)\right)$ *running time and comparisons, where* $\eta_i^{\Delta} := |\hat{p}(i) - p(i)|$.

Measurements

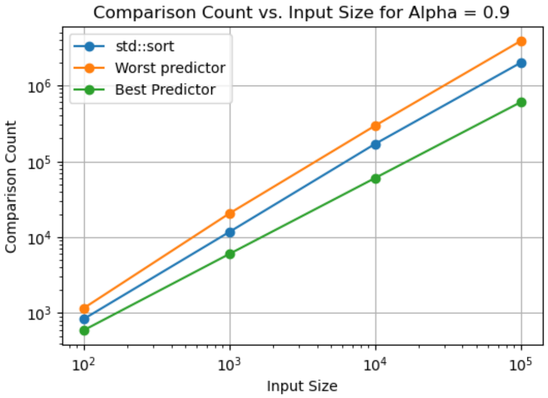


Figure: The number of comparisons needed to sort a buffer.

Using x-fast tries

- If we have data that can have an integer-like representation, we can use x-fast tries instead of predictors.
- The buffers on the lowest and highest levels should be implemented with an x-fast trie instead of an array.
- That way, we can fill a buffer in $O(k \log \log M)$ time instead of $O(k)$. From these, we can construct an array in $O(k)$ time, and merge them as usual.
- On the top level, we can construct an x-fast trie from a sorted array in $O(k)$ time, and use it to serve rank-queries in $O(\log \log M)$ time.

Thank you for your attention!