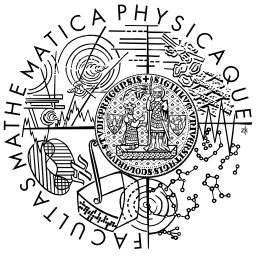

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Tomáš Domes

Streaming Algorithms for Estimating Quantiles with Novel Error Guarantees

Computer Science Institute of Charles University

Supervisor of the master thesis: Mgr. Pavel Veselý, Ph.D.
Study programme: Discrete Models and Algorithms

Prague 2024

I declare that I carried out this master thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

First of all, I would like to thank my supervisor Pavel Veselý for all his time and energy spent on the thesis, including countless meetings and extensive proofreading, and to my friend Jakub Tětek for coming up with the original idea behind the thesis.

In the past year, I neglected a large part of my duties as an organizer of M&M and our church's summer camp and as a member of SOM. I would like to thank my fellow organizers for not being grumpy with me for it.

I would like to thank Martin Mareš for creating and maintaining this nice LaTeX template.

And finally, I would like to thank my family for supporting me for those long 7 years of my studies even though my visits home were rare and short.

Title: Streaming Algorithms for Estimating Quantiles with Novel Error Guarantees

Author: Tomáš Domes

Institute: Computer Science Institute of Charles University

Supervisor: Mgr. Pavel Veselý, Ph.D., Computer Science Institute of Charles University

Abstract: This work deals with streaming algorithms for estimation of ranks and quantiles that perform a single pass through the input data stream using a small space. After reading a stream of N elements of a totally ordered universe, a streaming algorithm for rank (or quantile) estimation answers rank (or quantile) queries with *additive error* if the error is at most $\pm\varepsilon N$ and with *relative error* if for item y with rank $\mathrm{R}(y)$, the error is at most $\pm\varepsilon\,\mathrm{R}(y)$. The first problem is optimally solved by the KLL algorithm in space $\mathcal{O}(\varepsilon^{-1})$, and the best-known algorithm for the relative error is ReqSketch, which takes space $\mathcal{O}(\varepsilon^{-1}\log^{1.5} N)$.

Our algorithm called Jagged Sketch consists of two significant improvements to the ReqSketch algorithm. The first of the improvements reduces the error for high ranks by a factor of $\sqrt{\log(N)}$, the second one improves the error by a factor up to $\log(N)$ for important ranks chosen by the user and for ranks close to them, all while maintaining the same space complexity. We support our theoretical analysis by experiments that demonstrate that Jagged Sketch can indeed reduce the error for selected ranks while maintaining the same space and similar error for other ranks compared to ReqSketch.

For $\varepsilon \in \mathcal{O}(\log^{-1.5} N)$ Jagged Sketch achieves additive error in the same space as KLL while simultaneously retaining near-relative error guarantee. In practice, the error for large ranks is about four times larger, while for the lowest ranks, it is up to about a hundred times smaller.

Keywords: ranks estimation, quantile estimation, streaming algorithms, relative error

Contents

Introduction

Modern computers generate more data than we can store, however, we would still like to analyze them. That is where *streaming algorithms* come into play. The input of a streaming algorithm is a long *stream* of data; the algorithm processes the data sequentially (without storing them) and produces a small (sublinear) summary of the data, so-called *sketch*, which is designed to represent some aspect of the data accurately. The desired accuracy is specified in advance and the size of the sketch depends upon it.

In this thesis, we focus on streaming algorithms for representing the distribution of comparable data. That is, to approximate the median, percentiles, and more generally *quantiles* or *ranks*.

In this chapter, we formally define the problem and its variants and specify our area of interest, then we summarize the related work, explain our contribution, and finally we outline the structure of our thesis.

Problem definition

In this section, we formally define the problem of estimating ranks and quantiles in a streaming setting and its variants. For simplicity, we assume that all the items on the input are different. However, all the definitions can be extended to support equal items and all the algorithms are correct even without this assumption.

Ranks and quantiles estimation

Informally, the *rank* of an item present in a sequence is simply its position in the sequence after it is sorted. The rank of an item that is not present in the sequence is defined as the rank of the closest smaller item present in the sequence if such an item exists and 0 otherwise. *Quantiles* are the inverse of ranks, usually expressed as a number from $[0, 1]$. The item of rank r in a sequence of length n corresponds to a quantile r/n. The formal definition follows:

Definition 1 (Ranks and Quantiles)**.** *Let $\mathcal{S}$ be a sequence $x_1, \ldots, x_n$ drawn from universe $\mathcal{U}$ with a total order.*

For any item $y \in \mathcal{U}$ we define the rank of y in the sequence $\mathcal{S}$ as $\mathrm{R}(y, \mathcal{S}) \stackrel{\text{def}}{=} |\{i \in \{1, \ldots, n\} \mid x_i \leq y\}|$. If the sequence is clear from context, we write just $\mathrm{R}(y)$ instead of $\mathrm{R}(y, \mathcal{S})$.

For any real number $\alpha \in (0, 1]$, the α-quantile of the sequence $\mathcal{S}$ is the only item $z \in \mathcal{S}$ such that $\mathrm{R}(z, \mathcal{S}) = \lceil \alpha n \rceil$ and the 0-quantile of $\mathcal{S}$ is the minimum item from $\mathcal{S}$.

A *streaming algorithm* for estimating ranks and quantiles reads sequentially an input stream $\mathcal{S}$ of length N and builds a small summary called *sketch*. The algorithm itself and the sketch are commonly used interchangeably in the literature and we follow this convention. During (and after) processing of the stream, the sketch can answer rank and quantile queries for the already processed part of the stream.

A *rank query* consists of an item y comparable with the items of the input stream. The answer for a rank query is an estimate on $\mathrm{R}(y) = \mathrm{R}(y, \mathcal{S})$, and is denoted by $\hat{\mathrm{R}}(y)$. We say that the sketch answers the query with error $\mathrm{Err}(y) = \left|\mathrm{R}(y) - \hat{\mathrm{R}}(y)\right|$.

A *quantile query* consists of a real number $\alpha \in [0, 1]$. The answer for a quantile query is an item $\hat{y_\alpha} \in \mathcal{U}$ which is an estimate on the actual α-quantile y_α of $\mathcal{S}$. We say that the sketch answers the query with error $\mathrm{Err}(\alpha) = |\mathrm{R}(y_\alpha) - \mathrm{R}(\hat{y_\alpha})|$.

The most important parameter of the sketch is its size, which can depend on the stream length, accuracy of the sketch, size of the universe, and other parameters that we discuss below.

Variants of the problem

Additive and relative error guarantees

Let $\varepsilon \in (0, 1)$. The error for a rank or quantile query is *additive* if $\mathrm{Err} \leq \varepsilon N$. The error is *relative* if for a rank query y we have $\mathrm{Err}(y) \leq \varepsilon\, \mathrm{R}(y)$ or if in case of a quantile query α we have $\mathrm{Err}(\alpha) \leq \varepsilon \alpha N$.

A streaming algorithm has *additive (relative) error guarantee* of ε if it can answer all queries with additive (relative) error for a given ε.

Observe that a guarantee for rank queries implies the same type of guarantee for quantile queries with the same ε and vice versa. For this reason, we are mostly interested in the rank queries for the rest of the thesis.

Also note that the relative guarantee is strictly stronger than the additive and if we are interested in low-ranked items (which is often the case), the additive guarantee is insufficient, as for large N, the error εN can be larger than the rank $\mathrm{R}(y)$ we are asking for, rendering the additive guarantee useless.

In practice, we most often need the highest accuracy for high ranks. This need can be easily fulfilled by using a relative error sketch with a negation of the original comparison function (so that the highest-ranked items become lowest-ranked and vice versa).

Randomization

If the algorithm is *randomized* (uses internal randomness independent of the input), the guarantee must hold with some probability given in advance. The probability bound then also affects the size of the sketch. Randomized algorithms can have asymptotically smaller error than deterministic ones (such an example is ReqSketch which we discuss later).

In the case of randomized algorithms, we need to distinguish if the probability bound holds for any rank query, or for all rank queries simultaneously. The latter can be usually derived from the former with a small space increase by using a standard "epsilon net" argument together with the union bound.

Comparison model

In the definitions above, we only require the items to be comparable (so-called *comparison-based* algorithms). Thus, we only access the items through comparisons. However, some algorithms assume that the input stream consists of integers or

floating-point numbers. These *non-comparison-based* algorithms can use the items in other ways, e.g. to build a binary tree over the universe $\mathcal{U}$.

Foreknowledge of the stream length

Some algorithms do need to know the stream length N in advance. Sometimes, at least a polynomial upper bound is needed. In this work, we refer to these situations as the *static setting* (with foreknowledge of N) and the *dynamic setting* (without the foreknowledge of N).

Mergeability

An important property of a streaming algorithm is *mergeability*. Operation *merge* takes two sketches representing streams s_1 and s_2 and outputs one sketch representing the concatenation of s_1 and s_2. We say that a sketch is *fully mergeable*, if the error of the sketch built by any sequence of merge operations on items from a set S is the same as the error in the streaming model on a stream S (note that the streaming model can be seen as a repeated merge with a one-item sketch). There also exist some weaker versions of mergeability imposing additional conditions.

It is easy to see the practical advantages of mergeability, as it trivially enables seamless parallelization of the algorithm or applications in distributed settings.

Our setting

In this thesis, we focus exclusively on randomized algorithms for estimating ranks in the comparison model with the relative error guarantee. We design a sketch both for the static and dynamic setting (with and without the foreknowledge of N). Due to time constraints, our algorithm does not support the *merge* operation yet, but we believe it to be possible and we plan to make the sketch fully mergeable later.

Notation

Let us state here a few notes on the notation. For the exhaustive list of notation, please see the List of Notation.

- By *item*, we always mean an arbitrary item from a universe $\mathcal{U}$ with a total order.
- An input stream $\mathcal{S}$ consists of items from $\mathcal{U}$. Let $|\mathcal{S}| = N$ and $\mathcal{S}^t$ be the input stream at time t, thus the first t items of $\mathcal{S}$.
- For any sequence s let $\mathrm{R}(y, s)$ be the rank of item y in s and let $\mathrm{R}(y) = \mathrm{R}(y, \mathcal{S})$ be the rank of y in the input stream.
- $\hat{\mathrm{R}}(y)$ is the answer returned by the sketch for a rank query y.
- The error of a rank query y is $\mathrm{Err}(y) = \left|\mathrm{R}(y) - \hat{\mathrm{R}}(y)\right|$.
- By $\log x$ we always mean $\log_2 x$ and by $\ln x$ the natural logarithm $\log_e x$.
- We denote $\max(1, \log x)$ by $\overline{\log}(x)$ and $\max(1, x)$ by $\overline{x}$.

Prior work

In this section, we summarize the most relevant previous results. However, there are many different variants of the problem and many different properties of the sketches so our coverage of the area is limited, as this work is not primarily a survey. Particularly, we do not mention the results supporting deletion of elements (in addition to inserting), the possibility of weighted items, or the works focusing on the so-called relative-value error.

All the space bounds in this section are asymptotic and they are measured in memory words. For simplicity, we also assume the failure probability δ in the probabilistic sketches to be a constant and we omit it in the space bounds. All the bounds for randomized sketches hold for the single quantile approximation (the probability of failure is thus for one arbitrary query).

One of the most widely used sketches is t-digest [Dun21], which is fully mergeable and usually efficient in practice, but it gives no theoretical guarantees. Cormode et al. proved that for inputs drawn from certain distributions, its error can be almost arbitrarily large [Cor+21].

Additive error

Probably the most practically used of the theoretically robust sketches is the probabilistic comparison-based KLL sketch by Karnin et al. [KLL16]. It is simple, fully mergeable, and achieves the optimal space bound ε^{-1}.

The deterministic comparison-based sketches can not fit into this space due to the lower bound of $\varepsilon^{-1}\log(\varepsilon N)$ by Cormode and Veselý [CV20]. There is an optimal deterministic comparison-based GK sketch by Greenwald and Khanna [GK01] achieving the space $\varepsilon^{-1}\log(\varepsilon N)$, but the sketch is not fully mergeable. An older algorithm by Manku et al. [MRL99] achieves full mergeability in space $\varepsilon^{-1}\log^2(\varepsilon N)$.

From the non-comparison-based sketches, let us mention the deterministic q-digest by Shrivastava et al. [Shr+04] which uses space $\varepsilon^{-1}\log(|\mathcal{U}|)$. The sketch is fully mergeable, but the universe $\mathcal{U}$ must be known in advance. Gupta et al. recently announced that they can achieve the optimal space ε^{-1} deterministically [GSW24]. They build upon q-digest, but they did not prove that their sketch is fully mergeable.

Relative error

There are much fewer results regarding the relative error guarantee. For deterministic comparison-based sketches, we have a lower bound of $\varepsilon^{-1}\log^2(\varepsilon N)$ by Cormode and Veselý [CV20]. The state-of-the-art sketch in this setting is the work of Zhang and Wang [ZW07] achieving space $\varepsilon^{-1}\log^3(\varepsilon N)$. The sketch is however not fully mergeable.

There is a modified version of the aforementioned q-digest by Cormode et al. achieving relative error in space $\varepsilon^{-1}\log(\varepsilon N)\log(\mathcal{U})$ [Cor+06]. As the original q-digest [Shr+04], it is deterministic, non-comparison-based, fully mergeable, and requires the prior knowledge of the universe $\mathcal{U}$.

In the randomized comparison-based setting, the state-of-the-art algorithm is ReqSketch by Cormode et al. [Cor+23] upon which we build in this thesis.

ReqSketch is fully mergeable and it requires $\varepsilon^{-1}\log^{1.5}(\varepsilon N)$ space, which is just a factor of $\sqrt{\log(\varepsilon N)}$ from the lower bound the authors prove in the same paper.

This lower bound of $\varepsilon^{-1}\log(\varepsilon N)$ applies even to non-comparison-based randomized algorithms, however, as far as we know, there is no such algorithm with a relative error guarantee.

Our contribution

The purpose of this section is to present our main theorem (Theorem 1) and explain its meaning. The theorem is an analogy of Theorem 1 in [Cor+23] with a few differences. On one hand, our theorem is stated only for the static setting (with the foreknowledge of N) and without mergeability. On the other hand, for some settings of parameters, it yields better error than ReqSketch.

We believe that the limitations can be removed and the sketch can be made fully mergeable, but we leave it to future work. Before this is done, we offer a provisional analysis in the dynamic setting, without mergeability and with an additional factor of $\log^{1/4}(\varepsilon N)\log^{J}\mathrm{R}(y)$ in the error (where $J \geq 0$ is a parameter of the sketch). We state this result in Section 2.1 and we sketch its proof in Section 2.4.

Before stating the main result, let us explain the meaning of the parameters R and J of the sketch. The parameter R denotes the set of *important ranks*. These are the ranks for which the user needs the smallest error. We expect the most common case in practice to be $R = \{1\}$, as the motivation for considering relative error is the interest in small ranks. The parameter J expresses the priority of the important ranks relative to others – the larger the priority, the larger J. If all ranks are equally important to us, we set $J = 0$ and the set R is ignored.

Theorem 1 (The main theorem)**.** *Given any $N \in \mathbb{N}$, set R s.t. $|R| \in \mathcal{O}(1)$ and $R \subset \{1, \ldots, N\}$, and real parameters $0 < \delta \leq 0.5$, $0 < \varepsilon \leq 1$ and $0 \leq J \leq 1.4$, there is a comparison-based randomized algorithm that processes an input stream of length at most N and at any time answers any rank query y with probability $1 - \delta$ with error*

$$\mathrm{Err}(y) \leq \varepsilon\,\mathrm{R}(y) \cdot \sqrt{\frac{\overline{\log}\,\frac{N}{\mathrm{R}(y)}}{\log(\varepsilon N)}} \cdot \frac{\min_{r\in R}\left(\left|\overline{\log}\,\frac{\mathrm{R}(y)}{r}\right|^{J}, \log(\varepsilon N)\right)}{\log^{\min(1,J)}(\varepsilon N)}$$

in space

$$\texttt{SPACE} \in \mathcal{O}_J\left(\varepsilon^{-1}\cdot\log^{1.5}(\varepsilon N)\cdot\sqrt{\log\frac{1}{\delta}}\right)$$

in case of $J \neq 1$ and in space

$$\texttt{SPACE} \in \mathcal{O}_J\left(\varepsilon^{-1}\cdot\log^{1.5}(\varepsilon N)\cdot\sqrt{\log\frac{1}{\delta}}\cdot\log\log(\varepsilon N)\right)$$

in case of $J = 1$.

Before explaining the meaning of the theorem, let us stress that the hidden constant in the $\mathcal{O}$ notation depends on the parameter J. Specifically, the constant

grows with J close to 1 and for $J = 1$ it yields an additional $\log\log(\varepsilon N)$ factor in the space bound. Also note that the condition $J \leq 1.4$ is there only to simplify the analysis as the theorem holds for any value of $J \geq 0$.

To understand the theorem better, let us consider the three factors of the error bound:

$$\mathrm{Err}(y) \leq \varepsilon\,\mathrm{R}(y) \cdot \sqrt{\frac{\overline{\log \frac{N}{\mathrm{R}(y)}}}{\log(\varepsilon N)}} \cdot \frac{\min_{r \in R}\left(\overline{\left|\log \frac{\mathrm{R}(y)}{r}\right|}^{J}, \log(\varepsilon N)\right)}{\log^{\min(1,J)}(\varepsilon N)}$$

The first factor $\varepsilon\,\mathrm{R}(y)$ alone is by definition the relative error. ReqSketch [Cor+23], which we build upon, achieves error $\varepsilon\,\mathrm{R}(y)$ with the same space bound (except for the case $J = 1$ when our space bound is by $\log\log N$ factor worse). The second and third factors improving the error originate from our two major modifications of ReqSketch.

Let us first look at the second factor, which represents our error improvement for high ranks (we explain the intuition behind the improvement in Section 1.3). The maximum value of the second factor is achieved for $\mathrm{R}(y) = 1$ and for all sensible choices of ε, let us say $\varepsilon > N^{-1/2}$, the maximum of this factor is no larger than $\sqrt{2}$ (and in practice, it is close to 1). The expression is decreasing as a function of $\mathrm{R}(y)$ and for $\mathrm{R}(y) \approx N$ it becomes $\log^{-1/2}(\varepsilon N)$. In Figure 1 we show the impact of the second factor – we compare the full version of Jagged Sketch to a version without the improvement for high ranks. The x-axis is logarithmic and the y-axis shows the relative error (the error for a given rank divided by the rank)[1].

The third factor represents the jaggedness of the sketch which we explain in Sections 1.4 and 1.5. It is arguably more complicated, as it depends on the parameters R and J. Before we explain its meaning, let us argue that it cannot be too large. If $J \geq 1$, the denominator becomes $\log(\varepsilon N)$, and the numerator is clearly at most $\log(\varepsilon N)$, hence the whole expression is at most 1. If $J < 1$, we notice that $\mathrm{R}(y)/r$ is at most N and so the whole expression equals at most $\log(N)/\log(\varepsilon N)$. Again for $\varepsilon > N^{-1/2}$ this is at most 2 and in practice, it is close to 1.

As we concluded that the third factor does not worsen the error substantially, let us explain the dependence on the parameters R and J. If $\mathrm{R}(y)$ is close enough to some interesting rank r, i.e., their ratio is at most 2^c for a constant c, the numerator becomes c^J, thus the error is improved by a factor of $\Theta(\log^{\min(1,J)}(\varepsilon N))$.

For $J = 0$ the expression becomes 1 and disappears. For $J \in (0, 1)$ the whole expression equals

$$\left(\frac{\min_{r \in R}\overline{\left|\log \frac{\mathrm{R}(y)}{r}\right|}}{\log(\varepsilon N)}\right)^{J}.$$

This looks like the larger J, the better. However, as J gets closer to 1, the hidden constant in the $\mathcal{O}$ notation of the space bound grows and it becomes $\log\log(\varepsilon N)$ for $J = 1$. The experiments suggest that the most practical choice is $J \approx 0.5$, but it naturally depends on the priority we give to the important ranks. For $J \geq 1$

[1]Note that the experiments were made for a dynamic version of Jagged Sketch for which we do not have the tight analysis yet. For details, see Chapter 3.

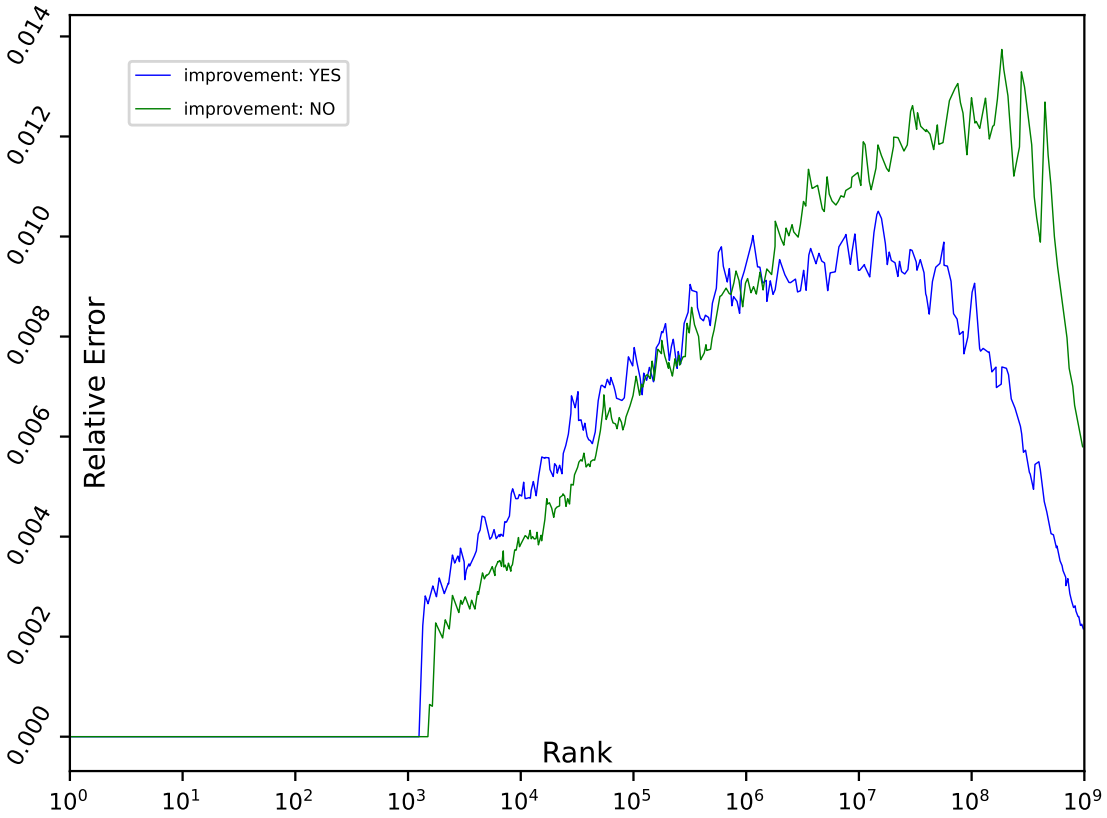

Figure 1 Comparison of Jagged Sketch with and without the improvement for high ranks for random input; $R = \{1\}$, $J = 0.5$ and $N = 10^9$

the denominator is fixed to $\log(\varepsilon N)$, hence at least in theory it does not make sense to choose J much larger than 1.

To wrap it up, the third factor can decrease the error by a factor of up to $\log(\varepsilon N)$ for ranks that are close to important ranks while not increasing it significantly for any other ranks. In Figure 2, we show Jagged Sketch with $R = \{1\}$ and $J = 0.4$ compared to ReqSketch. Indeed, there is a significant improvement for the low (important) ranks, whereas the worsening of error for higher ranks is negligible.

The contribution of our work is hence twofold. First, we give better error than the state-of-the-art sketch both in theory and in practice and second, we allow users of the sketch to influence the error for various ranks by expressing their priorities via the parameters R and J.

An unexpected side effect of our result is that for $\varepsilon < \log^{-1.5}(N)$ we get additive error in space $\Theta(\varepsilon^{-1}\sqrt{\log \delta^{-1}})$, which is only by a factor $\sqrt{\log \delta^{-1}}$ from the optimum, while maintaining a near-relative error. This is achieved by setting the parameters of the sketch $J > 1$, $R = \{N\}$ and $\varepsilon' = \varepsilon \log^{1.5}(N)$. The error becomes

$$\varepsilon \,\mathrm{R}(y) \cdot \sqrt{\log \frac{N}{\mathrm{R}(y)}} \cdot \min\left(\log^{J}\left(\frac{N}{\mathrm{R}(y)}\right), \log(\varepsilon N)\right) \leq \varepsilon \,\mathrm{R}(y) \cdot \log^{J+0.5}\left(\frac{N}{\mathrm{R}(y)}\right).$$

The error is always at most εN (additive error) and it is at most by $\log^{1.5}(N)$ factor larger than the relative error.

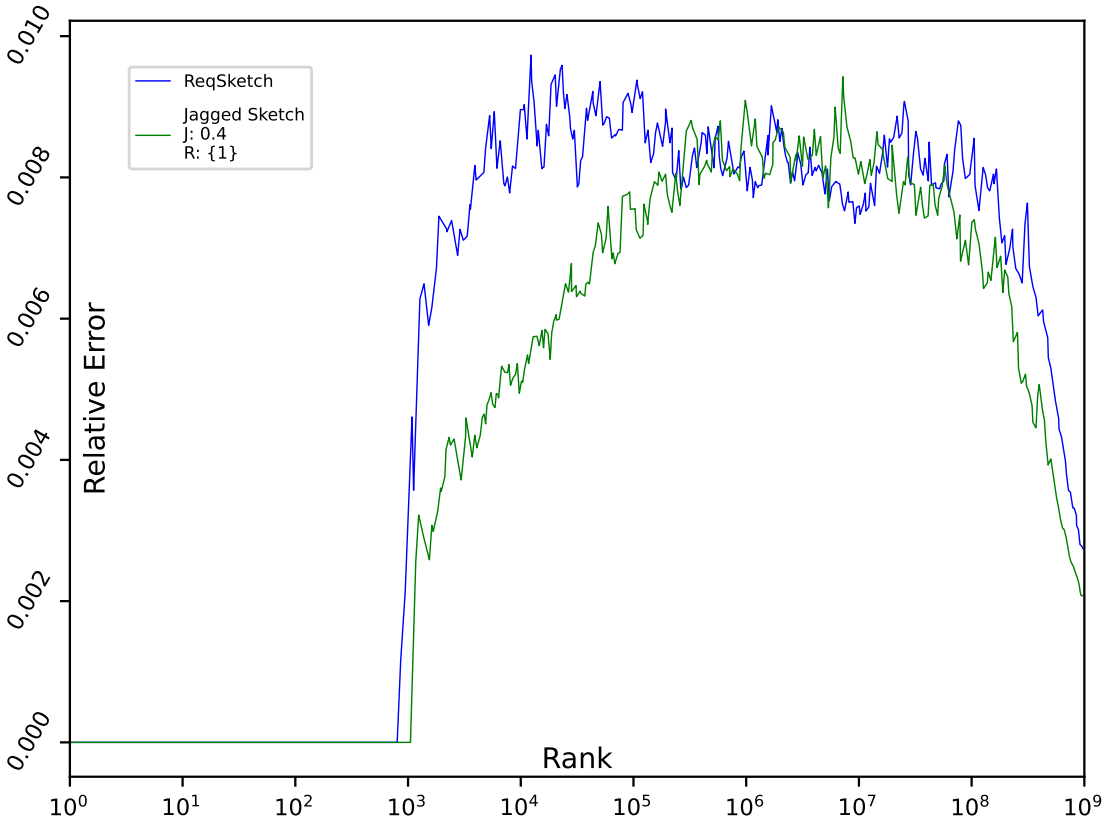

Figure 2 Comparison of ReqSketch and Jagged Sketch on random input with $R = \{1\}$, $J = 0.4$ and $N = 10^9$

Structure of the thesis

In this chapter, we have already defined the problem, summarized the related work, and presented our results. In Chapter 1, we informally explain the sketches which we build upon and the intuition behind Jagged Sketch. Chapter 2 contains a formal description of Jagged Sketch along with the proof of the main theorem (Theorem 1). We also discuss there the possibility of removing the assumption of the foreknowledge of N and sketch the not-yet-tight analysis in this setting. Finally, in Chapter 3 we introduce our proof-of-concept implementation of Jagged Sketch, explain our experimental setup, and present the results of the experiments for different settings of parameters R and J and compare them to ReqSketch [Cor+23] and KLL [KLL16].

1 Context and Intuition

In this chapter, we give an informal description of the streaming algorithm. We start in Section 1.1 by briefly explaining a basic version of the KLL sketch by Karnin et al. [KLL16] achieving additive error. In Section 1.2 we continue by showing the relative error sketch by Cormode et al. [Cor+23] (let us call it ReqSketch), which builds upon KLL. Then in Sections 1.3 to 1.5 we informally introduce our modifications of ReqSketch one by one, including the intuition behind them, and in Section 1.6 we describe the whole Jagged Sketch in one place.

All the error bounds stated in this chapter hold with some probability $1 - \delta$. For now, we treat δ as a constant and omit it in the space bounds. We assume for simplicity that $\log N \in \mathcal{O}(\log(\varepsilon N))$ (this is almost always true in practice) and we also omit rounding issues and some other technical details.

For the whole chapter, we assume the length N of the stream to be known in advance. We discuss lifting this assumption in Chapter 2 in Sections 2.1 and 2.4.

1.1 KLL

The KLL sketch by Karnin, Lang, and Liberty [KLL16] is the optimal randomized sketch for additive error in the comparison model which takes only $\mathcal{O}(\varepsilon^{-1})$ space. Note that such a small space is possible only because the additive error depends on N.

Description of the sketch

The KLL sketch consists of a sequence of buffers with given capacities (which are even). We call the buffers *compactors* and we imagine them as arranged into *levels* – the first compactor is at level 0 and the last at level $H - 1$. In the update operation, we simply add the new item to the compactor at level 0. If the compactor reaches its capacity, we perform a *compaction*, which operates on a single compactor as follows (see Figure 1.1):

We sort all the contained items and with probability 1/2 we discard all odd-indexed items, otherwise we discard all even-indexed items. Then, we move all the non-discarded items to the compactor one level higher (thus the compactor becomes empty after performing the compaction). The compaction at level 0 can of course trigger a compaction of the compactor at level 1 (if its capacity is reached), et cetera. Every time we perform a compaction on the highest level, we must increase the number of levels and create a new compactor on the new highest level.

The rank query

Intuitively, every item y in the level-1 compactor represents 2 original items: itself and its discarded neighbour from the level-0 compactor. Note that as we sort the items before discarding, the represented neighbour is similar to the item y (at least among the items currently present in the compactor), so item y is intuitively

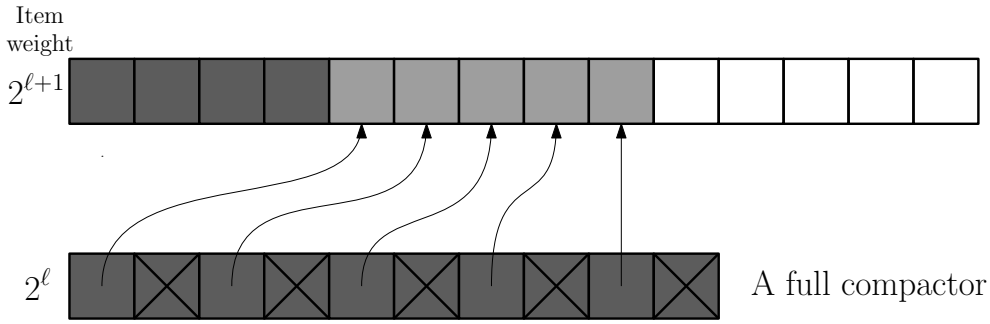

Figure 1.1 The compaction operation

a good choice for representing it[1]. By the same logic, an item at level 2 represents 4 original items, and generally an item at level l represents 2^l original items.

This intuition gives us the algorithm for answering a rank query. For a given item y, we just take the weighted sum of its rank in all the compactors, where the weight for level l is 2^l. For the formal definition, recall that rank of item x in sequence s is denoted by $\mathrm{R}(x, s)$, $\mathrm{R}(x)$ is the rank of x in the input stream, and $\hat{\mathrm{R}}(x)$ its estimate by the sketch. Then, for a sketch consisting of compactors $B_0 \dots B_h$, we have:

$$\hat{\mathrm{R}}(y) = \sum_{l=0}^{h} 2^l \, \mathrm{R}(y, B_l).$$

The space bound

It remains to set the compactor capacities. For a given parameter ε we set the capacity of the highest compactor to $C \stackrel{\text{def}}{=} \varepsilon^{-1}$. The capacities are then exponentially decreasing as we go down the levels, with some constant factor $0.5 < F < 1$. Thus the second-highest compactor has capacity FC, the one before it has capacity F^2C, and so on.

We need to note a few things now. First, for the compaction, we need the capacity C to be at least 2 and for a long input stream (or large ε), the low levels would be smaller. So we need a rule that any capacity is always at least 2. Second, we must tweak the capacities a bit so that they are even, which however does not change the asymptotic size. Last, if we compact the highest compactor and create a new one, the capacity of the already present compactors must decrease. This is however not a problem, as in the moment, the compactors are all empty.

Formally, let C_l be the capacity of the compactor at level l. For a given parameter ε and some constant $0.5 < F < 1$ we have:

$$C_l = \max\left(2, F^{H-l-1}\varepsilon^{-1}\right)$$

As the capacities are decreasing exponentially, the sum of the sizes of all the compactors of capacity larger than 2 is asymptotically bounded by the capacity of the highest compactor. It is not hard to see that all the compactors of capacity 2 can be replaced by a simple sampler – the sequence of x compactors of capacity 2 simply chooses one out of every 2^x items uniformly at random. This sampling

[1]Also note that we need the assumption that capacity is even and thus for every promoted item there is exactly one deleted neighbour.

can be performed in $\mathcal{O}(1)$ space[2], so after this modification, the overall size of the sketch is just $\mathcal{O}(\varepsilon^{-1})$. The whole situation is shown in Figure 1.2.

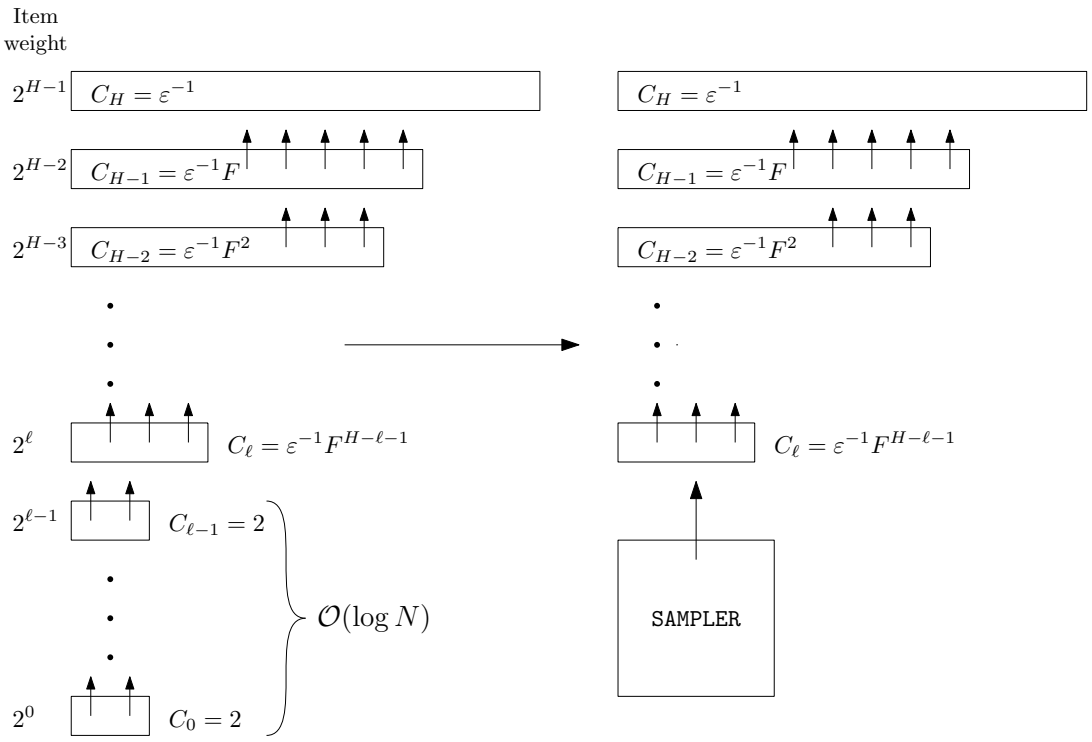

Figure 1.2 The KLL sketch

The error bound

It can be proven that this sketch can answer all rank queries within $\pm\varepsilon N$ error with constant probability (the additive error). The intuition behind the proof is that if the rank of an item y among the compacted items on level l is even, the compaction introduces no error for y, as for every removed item smaller than y with weight 2^l, there is exactly one item smaller than y that we "promoted" and whose weight doubles from 2^l to 2^{l+1}. If the rank of y among the compacted items is odd, the compaction introduced error 2^l or -2^l with equal probability (by the same reasoning). This means that the error for y can be bounded as

$$\mathrm{Err} = \sum_{l=0}^{H-1} \sum_{i=1}^{\substack{\#\text{ compactions} \\ \text{on level } l}} 2^l X_{i,l}$$

where $X_{i,l}$ are ± 1 zero-mean independent random variables. Hence, the error is a zero-mean random variable and we are interested in its variance. The number of compactions performed on level l is at most $(2/F)^{H-l-1}$ (it is trivially true for the highest level and extends to the lower levels by induction), so the variance is

$$\sum_{l=0}^{H-1} \left(\frac{2}{F}\right)^{H-l-1} 2^{2l} = \left(\frac{2}{F}\right)^{H-1} \sum_{l=0}^{H-1} (2F)^l.$$

[2]For details see Reservoir Sampling by Vitter [Vit85].

As we have $F > 1/2$, the variance is dominated by the next-to-last level. Since we have $H \leq \log \frac{N}{\text{size of the last level}}$, the variance is

$$\mathcal{O}\left(2^{2H}\right) \subseteq \mathcal{O}\left(2^{2\log(\varepsilon N)}\right) = \mathcal{O}\left(\varepsilon^2 N^2\right).$$

This is important for us as we use the property that the variance is dominated by the last level in the analysis of ReqSketch and Jagged Sketch. The proof can be finished by the use of Hoeffding's inequality with a little more refined argument.

For the error analysis, implementation of the merge operation, the error improvements, and other information, we refer to the original paper [KLL16].

1.2 ReqSketch

ReqSketch by Cormode et al. [Cor+23] is the state-of-the-art randomized sketch achieving relative error in the comparison model in space $\mathcal{O}(\varepsilon^{-1} \log^{1.5} N)$, which is a factor of $\sqrt{\log N}$ from the known lower bound.

In the analysis in the streaming setting, authors of the ReqSketch assume that the stream length N is known in advance. They get rid of the assumption in later chapters by introducing mergeability. We explain here only the simpler version with the foreknowledge of N.

Differences from KLL

The sketch works analogously to KLL with two differences. The first difference is that all the compactors have the same capacity. Particularly, for any level l we have $C_l = C = 2\varepsilon^{-1}\sqrt{\log N}$. This implies that unlike for KLL, there are no compactors of constant capacity and no need for a sampler.

The second (major) difference is the definition of compaction operation. After the compactor is sorted, we add a new step of the algorithm – choosing the number X of items to participate in the compaction. After the choice is made, we perform the compaction on the largest X items in the compactor (we discard odd/even ranked items and promote the rest). Thus in every compaction, there are $C - X$ *protected* items staying in the compactor. Particularly, we always choose $X \leq C/2$, so the smaller half of items in the compactor is always protected. The process of compaction is shown in Figure 1.3.

Protected part of the compactor

The reason for the protected left half of each compactor is that items with small ranks cannot move too high in the sketch. Particularly, the smallest $C/2$ items stay on level 0, the (representatives of the) next C items stay on level 1, and so on[3]. For an item y with a small rank, only a small number of compactors on the low levels ever perform a compaction containing items smaller than y and only their compactions can contribute to the error for y. As in the KLL sketch, the error is a zero-mean variable and its variance is dominated by the highest level

[3]It is also true that by the definition of relative error, the sketch must answer the queries for the smallest $1/\varepsilon$ items exactly with no error, so these items must be protected. An analogous argument holds for the next $2/\varepsilon$ items and level 1 and so on.

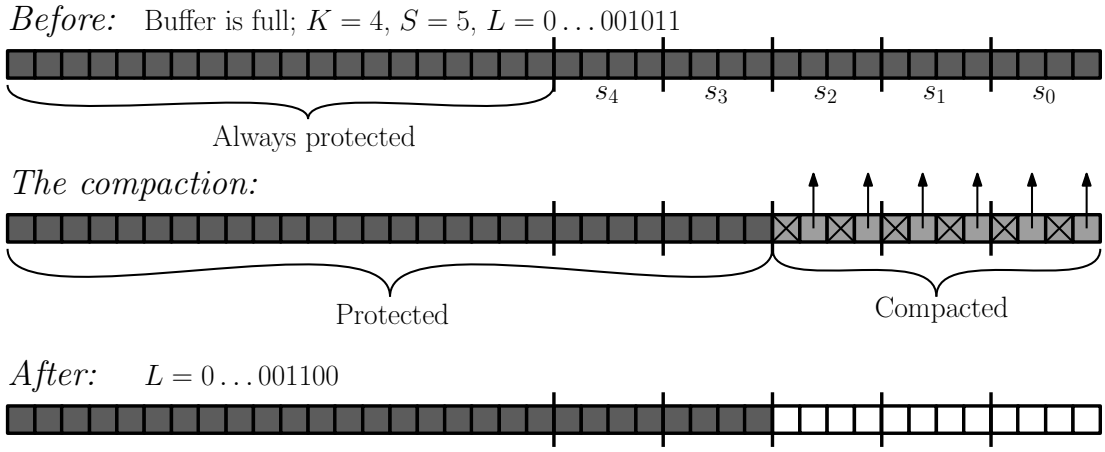

Figure 1.3 Compaction in ReqSketch

which performs compaction affecting the error, so we get much a smaller error for low-ranked items (which is the point of having a relative error).

Compaction schedule

Now, let us explain the choice of number X. For this choice, we use a parameter $K = \varepsilon^{-1}/\sqrt{\log N}$ and so called *compaction schedule*. The compaction schedule is determined by a binary representation of the number P of compactions already performed on the given compactor. Let T be the number of trailing ones of P before the compaction. We define $X \stackrel{\text{def}}{=} (T+1)K$.

Note that while P is smaller than $N/2$, T is smaller than $\log(N/2) = \log(N) - 1$ and so X is at most $K \log N = \varepsilon^{-1}\sqrt{\log N} = C/2$. It follows that the smallest $C/2$ items in the buffer are indeed always protected.

To see the motivation behind the compaction schedule, let us divide the right half (the one with larger items) of the compactor into S sections of size K and let us number the sections from right to left starting from zero. If the schedule has T trailing ones, we compact sections T to 0. Again we would like to bound the error for item y of a small rank. We show how to assign K items smaller than y to each compaction that affects the error of y in such a way that any item is assigned at most once. If this is true, it follows that for small y there cannot be too many compactions.

Let us consider a compaction affecting the error of y where the last compacted section is j. After the compaction, all the items in section $j+1$ are smaller than y (as the rank of y among non-protected items was not zero) and immediately after the compaction, we set the bit of the compaction schedule corresponding to section $j+1$ to one. This set bit indicates that section $j+1$ will eventually pay for this compaction. The next compaction which involves section j must also involve section $j+1$ and all the items in section $j+1$ are still smaller than y at the time of the compaction. We conclude that the compaction removes at least K items smaller than y from the compactor – all the items from section $j+1$. If such compaction never happens, some K items smaller than y stay in section $j+1$ forever. Either way, for every compaction affecting the error of y we have some K items smaller than y to pay for it.

Space and error bounds

Regarding the size of the sketch, note that we have $\mathcal{O}(\log N)$ levels, as the weight of items doubles with every step up, so the whole sketch takes space $\mathcal{O}(C \log N) = \mathcal{O}(\varepsilon^{-1} \log^{1.5} N)$.

It can be proven that the sketch answers a query for $\mathrm{R}(y)$ with error $\pm\varepsilon\,\mathrm{R}(y)$ with constant probability. The error analysis and other information including mergeability can be found in the paper [Cor+23].

1.3 Error improvement for high ranks

In this section, we introduce our first new result – an error improvement for high ranks, which is based on a practical adjustment of ReqSketch by Cormode et al. [Cor+21].

Explanation of the modification by Cormode et al.

As we mentioned earlier, the original analysis of ReqSketch in the streaming setting was done with the foreknowledge of a polynomial upper bound on N, which enabled the authors to analyze the sketch in a static case (fixed number of levels and their capacities). In the dynamic case, the sketch is built for some initial constant stream length, and all the parameters are recomputed whenever the logarithm of the stream length doubles (this makes sense as the parameters of compactors depend on the logarithm of N). In a paper where they compare ReqSketch to t-digest, Cormode et al. [Cor+21] suggest to do this update of parameters for each compactor separately whenever the logarithm of the number of compactions on the given compactor doubles. They do not explain the idea further, but we expect this modification can be analyzed similarly as the original version. Before we move on, let us explain the behaviour of a compactor in this model.

As in KLL, we start with just one compactor, and any time the highest compactor does its first compaction, we create a new compactor at the top. The new compactor is always created with the same constant capacity $C = \varepsilon^{-1}$ and constant number of sections – this means section size K is a constant fraction of C (let us say $K = C/6$). With this setting, we quickly get to the situation where (by the compaction schedule) we compact $C/2$ items. In this case, we multiply C by $\sqrt{2}$, divide K by $\sqrt{2}$, and reset the compaction schedule to zero. This means that the compaction schedule is no longer equivalent to the number P of already performed compactions. Let us denote the schedule L.

Note that after every reset, the number of sections S (which equals $\frac{C}{2K}$) doubles. Also note that a reset happens when the number of compactions performed from the last reset is $\approx 2^S$. This means that the overall number of compactions on the given compactor is $2^{S/2} < P < 2^{S+1}$, thus we have $S \approx \log P$. The number of resets performed is $E \approx \log(S) \approx \log\log P$, the compactor size is $C = \varepsilon^{-1}\sqrt{2}^E \approx \varepsilon^{-1}\sqrt{\log P}$, and the section size is $K \approx C/S \approx \varepsilon^{-1}/\sqrt{\log P}$. Recall that in the static version we had $C \approx \varepsilon^{-1}\sqrt{\log N}$ and $K \approx C/S \approx \varepsilon^{-1}/\sqrt{\log N}$.

Note that we did not break the analysis of the compaction schedule. The only compactions without assigned sections (represented by a bit in the schedule)

are the compactions immediately before the schedule reset. We can assign these compactions to some K-tuples of positions in the left half of the compactor at the time of the compaction and as the compactor size increases by factor $\sqrt{2}$ between any two resets, there are always free unassigned positions in the left half.

Our modification

In the version of ReqSketch explained above, the compactors on high levels have smaller sizes. It is natural to ask whether the asymptotic size of the whole sketch decreased. Unfortunately, it is not the case. All the compactors in the bottom (larger) half of the sketch have an input stream of length at least $\sqrt{N}$ and so each of them performs $\Omega(\sqrt{N}/K)) \subseteq \Omega(\varepsilon\sqrt{N}/\sqrt{\log N})$ compactions. This means that each of them has a size

$$\Omega\left(\varepsilon^{-1}\sqrt{\log P}\right) \subseteq \Omega\left(\varepsilon^{-1}\sqrt{\log \frac{\varepsilon\sqrt{N}}{\sqrt{\log N}}}\right) = \Omega\left(\varepsilon^{-1}\sqrt{\log N}\right).$$

So the size of the bottom half of the sketch is still $\Theta(\varepsilon^{-1}\log^{1.5} N)$.

Let us move back to the static version and exploit the smaller compactors in another way – we exchange the size improvement (which was not significant) for an error improvement. We set the size of all the compactors to $C = \varepsilon^{-1}\sqrt{\log N}$ (as it was originally), but preserve the ratio between K and C (the number of sections S) which equals approximately $\log P$. This implies that we have

$$K \approx \varepsilon^{-1}\frac{\sqrt{\log N}}{\log P}.$$

The analysis of the compaction schedule stays the same as in the modified version above, but the section size K is up to $\log N$ times larger for the highest levels compared to ReqSketch.

Recall that the variance of the error is dominated by the highest level which contributes to the error. For this reason, the increase of K translates to an error improvement for high-ranked items. Particularly, the error becomes

$$\Theta(1)\cdot\varepsilon\,\mathrm{R}(y)\sqrt{\frac{\log\frac{N}{\mathrm{R}(y)}}{\log N}}$$

instead of $\varepsilon\,\mathrm{R}(y)$, which is improvement by a factor of $\sqrt{\log N}$ for ranks in $\Omega(N)$, as the numerator becomes a constant.

1.4 Jaggedness

Here, we introduce our main novel idea which makes the sketch more versatile.

Once again we are going to utilize the fact that for a given rank the variance of the error is dominated by the highest reached level. If we know in advance which ranks we are interested in, we can set the capacities of the compactors so that they are larger on the levels that are critical for bounding the error for items of interested ranks. So we go back to the concept from KLL where the highest

compactor has the largest capacity and the capacities of other compactors are decreasing exponentially. However, we introduce a few changes.

First, we let the user choose a constant number of important ranks determining the large levels[4]. So it can be the last level as in KLL, it can be the first, it can be some level in-between, and there can even be more of them. This is where the name of the sketch comes from – the sketch has one or more jags represented by the large levels (see Figure 1.4).

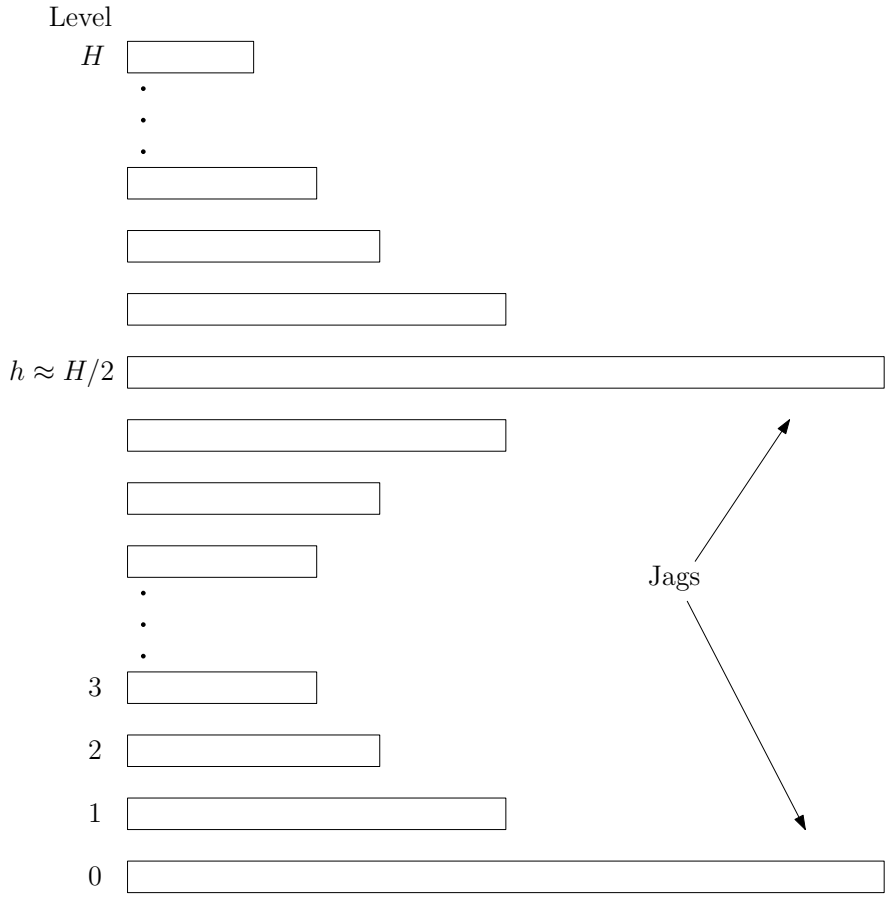

Figure 1.4 Jagged Sketch with important ranks 1 and $\sqrt{N}$.

Second, the level sizes do not decrease exponentially, but just polynomially. Particularly, if the largest compactors have capacity C, the capacity of the compactor with distance d from the closest largest compactor is C/d^J for some constant $J > 1$ (in KLL it would be $C \cdot F^d$ for a constant $0.5 < F < 1$). We stress that in this section $J > 1$, we discuss $J \leq 1$ in Section 1.5.

In this setting, the error for the items with the given important rank is the same as in ReqSketch, but the size of the sketch is $\mathcal{O}(C)$ instead of $\mathcal{O}(C \log N)$, as $\sum_{i=1} C/i^J \in \mathcal{O}(C)$ for $J > 1$.

For example, if we set 1 as the only important rank (which is the most natural setting, as the motivation for relative error is high accuracy for small items), and $C = \varepsilon^{-1} \log^{1.5} N$ (to achieve the same space as ReqSketch) we get error

$$\mathrm{Err}(y) \leq \varepsilon \, \mathrm{R}(y) \cdot \frac{\log^{0.5}\left(\frac{N}{\mathrm{R}(y)}\right) \log^{J} \mathrm{R}(y)}{\log^{1.5}(N)},$$

[4]The rank $\mathrm{R}(y)$ corresponds roughly to level $\log \mathrm{R}(y)$.

compared to $\varepsilon \,\mathrm{R}(y)$ of the original ReqSketch.

As the error grows with J, we want to choose J close to 1. On the other hand, J affects the hidden constant that grows as J gets closer to 1. Thus, the actual value used in practice should be determined by experiments.

Note that for ranks in $\Theta(N^c)$ for $0 < c < 1$, this error guarantee is actually worse than before, as $\log^J \mathrm{R}(y)$ becomes larger than $\log N$. This could be simply compensated (without asymptotic space increase) by running two sketches (with and without jags) of the same size side by side and later ask the sketch which gives a better guarantee for the given rank. However, there is even a simpler solution – for every compactor we let the capacity to be the maximum of the two possible sketches. This sketch gives us always the better error bound with the same asymptotic size. Thus, the final compactor capacity becomes approximately $C/\min(d^J, \log N)$ where C is the capacity of the largest compactors and d is the distance to the closest important level. For space $\varepsilon^{-1}\log^{1.5} N$ and with the only important rank 1 the error becomes

$$\mathrm{Err}(y) \le \varepsilon \,\mathrm{R}(y) \cdot \frac{\log^{0.5}\left(\frac{N}{\mathrm{R}(y)}\right) \min\left(\log^J \mathrm{R}(y), \log N\right)}{\log^{1.5}(N)}.$$

Note that for small ranks the error is nearly $\log N$ times better than for ReqSketch, for large ranks it is $\sqrt{\log N}$ times better, and for the ranks in between it is the same.

To compare Jagged Sketch to KLL, let us set the only important rank to N. The error becomes

$$\begin{aligned}\mathrm{Err}(y) &\le \varepsilon \,\mathrm{R}(y) \cdot \frac{\log^{0.5}\left(\frac{N}{\mathrm{R}(y)}\right) \min\left(\log^J \frac{N}{\mathrm{R}(y)}, \log N\right)}{\log^{1.5}(N)} \\ &\le \varepsilon \,\mathrm{R}(y) \cdot \frac{\log^{0.5+J}\left(\frac{N}{\mathrm{R}(y)}\right)}{\log^{1.5}(N)}\end{aligned}$$

which is $\varepsilon \,\mathrm{R}(y)/\log^{1.5} N$ for ranks in $\Omega(N)$. That means that if $\varepsilon \le \log^{-1.5} N$ we can use $\log^{1.5} N$ times larger ε to achieve the same space as KLL with the same error for high ranks. Then for ranks N^c for some constant $0 < c < 1$, the error becomes $\mathcal{O}(\varepsilon N^c \log^{J+0.5} N)$ and for ranks $\log^c N$ it becomes $\mathcal{O}(\varepsilon \log^{c+J+0.5} N)$. KLL has error εN for all ranks, thus Jagged Sketch has (asymptotically) the same error for ranks in $\Omega(N)$ and a much better error for the smaller ranks (under the assumption $\varepsilon \le \log^{-1.5} N$).

1.5 Weighted jaggedness

In the previous section, we introduce the jaggedness with parameter $J > 1$. Observe that by setting $J = 0$, we get ReqSketch with no jags, as the capacities are all the same. The natural question is: What happens for J between 0 and 1? Intuitively, constant J tells us the size (or significance) of the jags – the capacity of the largest compactors relative to others. From another point of view, it tells us the size of the sketch. Let C be the size of the important (largest) level. With $J = 0$, the sketch size is $\approx C \log N$, with $0 < J < 1$, it is $\approx C \log^{1-J} N$, for $J = 1$ it is $\approx C \log\log N$ (harmonic series) and for $J > 1$ it is $\approx C$.

For the appropriate setting of parameter C and again with the only important rank 1 the general error for any $J \geq 0$ becomes

$$\mathrm{Err}(y) \leq \varepsilon\,\mathrm{R}(y) \cdot \frac{\log^{0.5}\left(\frac{N}{\mathrm{R}(y)}\right) \min\left(\log^{J}\mathrm{R}(y), \log N\right)}{\log^{0.5+\min(1,J)}(N)}$$

where the size of the sketch $\mathcal{O}(\varepsilon^{-1}\log^{1.5} N)$ is preserved when $J \neq 1$ and for $J = 1$ it is $\mathcal{O}(\varepsilon^{-1} \cdot \log^{1.5} N \cdot \log\log N)$.

It is hard to say which setting of the parameter J is the best. All the settings with $J \neq 1$ are (asymptotically) never worse than the original ReqSketch and the intuition above generally holds – the larger J, the better the guarantee for important ranks. The error decreases as the parameter J goes to 1, but the hidden constant depending on J increases with J close to 1 and it becomes $\log\log N$ for $J = 1$ as we can see in the change of the space bound. The actual value of J should be determined by the priority we give to the important ranks relative to other ranks and by experiments. Our data suggest that the universally most practical choice of J is $J \approx 0.5$ (see Chapter 3).

1.6 The whole Jagged Sketch

Let us now put all the pieces together and describe the whole sketch. Recall that for simplicity, we ignore rounding issues and other technical details. For the complete formal description, see Chapter 2.

The sketch consists of a sequence of *compactors* arranged into *levels* from 0 to $H - 1 \approx \log(\varepsilon N)$. It is initialized by the error bound ε, jaggedness J, and a constant number of *important ranks*. The set of *important* levels is the set of logarithms of important ranks.

Each compactor has its own *compaction schedule* L (initially zero), the number of already performed compactions P, the *scaling factor* $F \approx \min(H, d^J)$ where d is the distance to the closest important level, the *capacity* $C \approx \sqrt{H}/(\varepsilon F)$, and the *section size* $K \approx C/\log P$.

Method *compact* works on a single compactor. We start by incrementing P, sorting the compactor, and calculating the number of protected items. If T is the number of trailing ones of L, the number of protected items is $C - (T+1)K$. Then we delete the non-protected items on even or odd indexes with equal probability and send the non-deleted half of them to the output of the compaction operation. Finally, if the size of the protected part was only $C/2$, we set $L = 0$ and $K = C/\log P$ (schedule reset), otherwise we just increment L. Note that the capacity does not change, as it does not depend on P.

The *update* operation simply adds a new item to the first compactor and if the first compactor is full (the number of items is greater or equal to its capacity), it performs a compaction. The output of the compaction is added to the compactor one level higher, which also performs a compaction if it is full, and so on.

The *rank query* for y returns a weighted sum of ranks of y in all the levels where the weight for level ℓ is 2^ℓ.

2 Analysis

As we explained in the previous chapter, Jagged Sketch is an improvement of ReqSketch by Cormode et al. [Cor+23], therefore the analysis is inspired by the analysis of ReqSketch. However, as the sketch became more complex, so did the proofs.

In Section 2.1 we state the result in a static setting and discuss the guarantees in the dynamic setting, in Section 2.2 we describe our algorithm, in Section 2.3 we do the full analysis in the static setting and in Section 2.4 we sketch the changes to be made in the analysis for the dynamization of the sketch.

2.1 Our main result

The following is a generalization of Theorem 1 in [Cor+23] in the static setting and without the mergeability. We stress that the hidden constant in the $\mathcal{O}$ notation depends on J and also that the condition $J \leq 1.4$ is there only to simplify the proofs, the analysis holds for an arbitrarily large J with different constants.

Theorem 1 (The main theorem)**.** *Given any $N \in \mathbb{N}$, set R s.t. $|R| \in \mathcal{O}(1)$ and $R \subset \{1, \ldots, N\}$, and real parameters $0 < \delta \leq 0.5$, $0 < \varepsilon \leq 1$ and $0 \leq J \leq 1.4$, there is a comparison-based randomized algorithm that processes an input stream of length at most N and at any time answers any rank query y with probability $1 - \delta$ with error*

$$\mathrm{Err}(y) \leq \varepsilon \,\mathrm{R}(y) \cdot \sqrt{\frac{\overline{\log}\,\frac{N}{\mathrm{R}(y)}}{\log(\varepsilon N)}} \cdot \frac{\min_{r \in R}\left(\left|\log\frac{\mathrm{R}(y)}{r}\right|^{J}, \log(\varepsilon N)\right)}{\log^{\min(1,J)}(\varepsilon N)}$$

in space

$$\texttt{SPACE} \in \mathcal{O}_J\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon N) \cdot \sqrt{\log\frac{1}{\delta}}\right)$$

in case of $J \neq 1$ and in space

$$\texttt{SPACE} \in \mathcal{O}_J\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon N) \cdot \sqrt{\log\frac{1}{\delta}} \cdot \log\log(\varepsilon N)\right)$$

in case of $J = 1$.

As in the original paper, we observe that at the cost of a small space increase, we can make the error bound to hold with probability $1 - \delta$ for all the queries simultaneously. We restate it without proof, which is analogous and can be found in Appendix B of [Cor+23].

Corollary 1 (All quantile approximation)**.** *The error bound from Theorem 1 holds for all rank queries simultaneously with probability $1 - \delta$ when the size of the sketch is*

$$\mathcal{O}\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon N) \cdot \sqrt{\log\left(\frac{\log(\varepsilon N)}{\varepsilon\delta}\right)}\right)$$

in the case $J \neq 1$ *and*

$$\mathcal{O}\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon N) \cdot \sqrt{\log\left(\frac{\log(\varepsilon N)}{\varepsilon\delta}\right)} \log\log(\varepsilon N)\right)$$

in case of $J = 1$.

Our sketch has not proven to be mergeable yet, but we believe it can be made so. We did not do it for time reasons and we plan to resolve this problem later. The mergeability will also help us remove the assumption of the foreknowledge of N. For now, we have a provisional modification of the analysis which removes the assumption on N at the cost of worsening the error by the factor of $\log^{1/4}(\varepsilon N) \log^{J} \mathrm{R}(y)$. We do not state this claim as a theorem, as we do not give a full proof. However, in Section 2.4 we roughly explain the changes to be made in the analysis.

To state the guarantees for the dynamic version, we must change the representation of the set of important ranks. It is given in the form of input for the quantile function: $|Q| \in \mathcal{O}(1), Q \subset [0, 1]$. The ranks R can be any time computed from the current value of N as $R = \left\{\overline{\lceil Nq \rceil} \mid q \in Q\right\}$.

Without the foreknowledge of N, with an additional assumption $\varepsilon \geq N^{-1/2}$ and with the same space bounds we have

$$|\mathrm{Err}(y)| \leq \varepsilon\, \mathrm{R}(y) \frac{\sqrt{\overline{\log \frac{N}{\mathrm{R}(y)}}}}{\log^{0.25+\min(1,J)}(\varepsilon N)} \max\left(\min_{q \in Q} \left|\overline{\log \frac{\mathrm{R}(y)}{\overline{\lceil qN \rceil}}}\right|^{J}, \log^{J} \mathrm{R}(y)\right).$$

In the special case where $Q \neq \emptyset$ and $Q \subseteq \{0, 1\}$, without the foreknowledge of N, with the same space bounds and with the assumption $\varepsilon \geq N^{-1/2}$ we have a better result:

$$|\mathrm{Err}(y)| \leq \varepsilon\, \mathrm{R}(y) \frac{\sqrt{\overline{\log \frac{N}{\mathrm{R}(y)}}}}{\log^{0.25+\min(1,J)}(\varepsilon N)} \min_{q \in Q} \left|\overline{\log \frac{\mathrm{R}(y)}{\overline{\lceil qN \rceil}}}\right|^{J}$$

And finally in the special case where $Q = \{1\}$ and $J \geq 0.5$ we have the same error as in Theorem 1 with the same space bounds, without the foreknowledge of N and without any additional assumption.

2.2 Description of the sketch

In this section, we give a full description of Jagged Sketch. For better understanding of the sketch we also recommend reading the simple version of our proof-of-concept implementation `jaggedSketchSimple.py`[1]. The only major difference from the description is that the implementation follows the dynamic version of Jagged Sketch which is explained in Section 2.4.1. For the reader's convenience we reprint there the schema of a compaction operation and the schema of Jagged Sketch from Chapter 1 – see Figures 2.1 and 2.2.

The description of the sketch requires a lot of notation. For later reference, we include a List of Notation at the end of the thesis.

[1] `https://github.com/domestomas/JaggedSketch/blob/main/jaggedSketchSimple.py`

The sketch

The sketch consists of a sequence of H *compactors* indexed from 0 to $H-1$. We imagine the compactors as arranged in *levels* – the compactor at level 0 is at the bottom, the compactor at level $H-1$ at the top. A compactor at level h has some internal state, a *capacity* C_h, an input stream I_h, an output stream O_h, and contains a set B_h of items (the buffer). We omit the index h if it is clear from context. Any set of items received from the input stream is simply added to B. Whenever the size of B exceeds the capacity C, the compactor performs a *compaction operation*, which removes some items from B, part of them is deleted and part of them is sent to the output stream. The input of the compactor on level 0 is the input of the sketch and the input of any compactor on level $l > 0$ is the output of the compactor on level $l-1$. We note that the items are sent in "batches" – if a compaction operation outputs a set of items, all the items are atomically added to the next-level compactor before any other compaction happens.

Apart from the sequence of compactors, the sketch consists of some additional information. An upper bound N on the stream length known in advance, the error bound $0 < \varepsilon < 1$, the height of the sketch $H \stackrel{\text{def}}{=} \lceil \log(\varepsilon N) + 1 \rceil$ (the number of compactors), the jaggedness $0 \leq J \leq 1.4$, the failure probability $0 < \delta \leq \frac{1}{2}$ and the set R of important ranks such that $|R| \in \mathcal{O}(1)$ and $R \subset \{1, \ldots, N\}$.

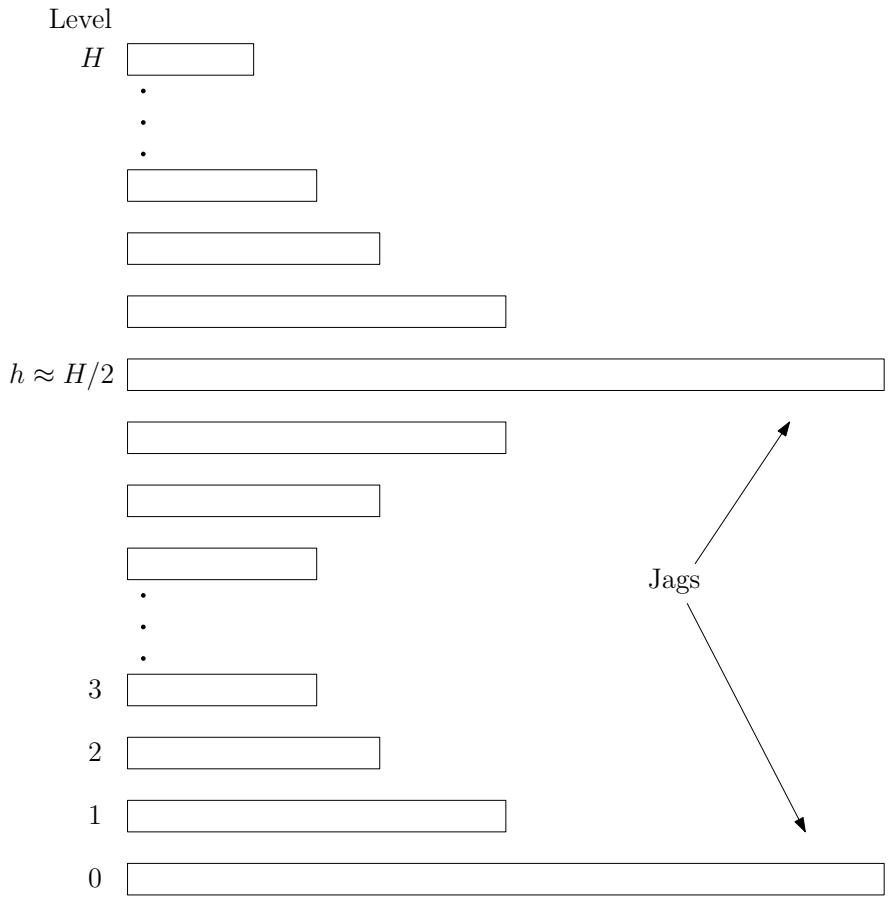

Figure 2.1 Jagged Sketch with important ranks 1 and $\sqrt{N}$.

Definition 2 (Important levels)**.** *For any important rank $r \in R$ we define an important level $\mathcal{L}(r)$ as*

$$\mathcal{L}(r) \stackrel{\text{def}}{=} \max\left(\log \frac{\varepsilon r}{2^2\sqrt{H \ln(\delta^{-1})}}, 0\right)$$

The intuition behind the definition is that $\mathcal{L}(r) \approx \log(r/C_{\mathcal{L}(r)})$ (see the definition of C below). It corresponds to the *critical level* that we see later in Section 2.3.2. Note that the important levels are determined at the beginning of the algorithm and never change.

The compactor

The compactor consists of set B of items, *height* h (the index of the compactor in the sketch), *capacity* C, *section size* K, *scaling factor* F, the number P of already performed compactions and *compaction schedule* L.

Let $d \stackrel{\text{def}}{=} \min_{r \in R} |\mathcal{L}(r) - h|$ be the distance to the closest important level. The initial setting of the parameters follows:

$$\begin{aligned}
P &= 0 && \text{(Number of compactions)}\\
L &= 0 && \text{(Compaction schedule)}\\
F &\stackrel{\text{def}}{=} \overline{\min(H, d^J)} && \text{(Scaling factor)}\\
K &\stackrel{\text{def}}{=} 2^3 \frac{\sqrt{H \ln(\delta^{-1})}}{\varepsilon F \,\overline{\log}(P)} && \text{(Section size)}\\
C &\stackrel{\text{def}}{=} 2^5 \frac{\sqrt{H \ln(\delta^{-1})}}{\varepsilon F} && \text{(Capacity)}\\
S &\stackrel{\text{def}}{=} \frac{C}{2K} = 2\,\overline{\log}(P) && \text{(Number of sections)}
\end{aligned}$$

Recall that $\overline{x}$ denotes $\max(1, x)$ and $\overline{\log}\, x$ denotes $\max(1, \log x)$. Also note that the parameters F and C stay fixed forever, whereas the rest of the parameters are updated at some time during the algorithm (which is explained below).

The compaction

The compaction operation is triggered when the size of B reaches the capacity and it is defined as follows: First, the number of compactions P is incremented by 1 and the stored items are sorted. Then the number of *protected* items is determined. Let T be the number of trailing ones of the binary representation of number L. If $K < 2$, the number of protected items is defined as $\lceil C/2 \rceil$, if $K \geq 2$, it is defined[2] as $\lceil C - (T+1)K \rceil$. If the number of non-protected items is odd, the number of protected items is incremented by 1. With probability 1/2, the odd-indexed non-protected items are sent to the output of the compactor, and in the other case, the even-indexed non-protected items are sent to the output. All the non-protected items are deleted from B. Finally, if $C - (T+1)K \leq \lceil C/2 \rceil$,

[2]Note that if we imagine the buffer B as divided into sections of size K, the compaction essentially compacts the rightmost $T+1$ sections.

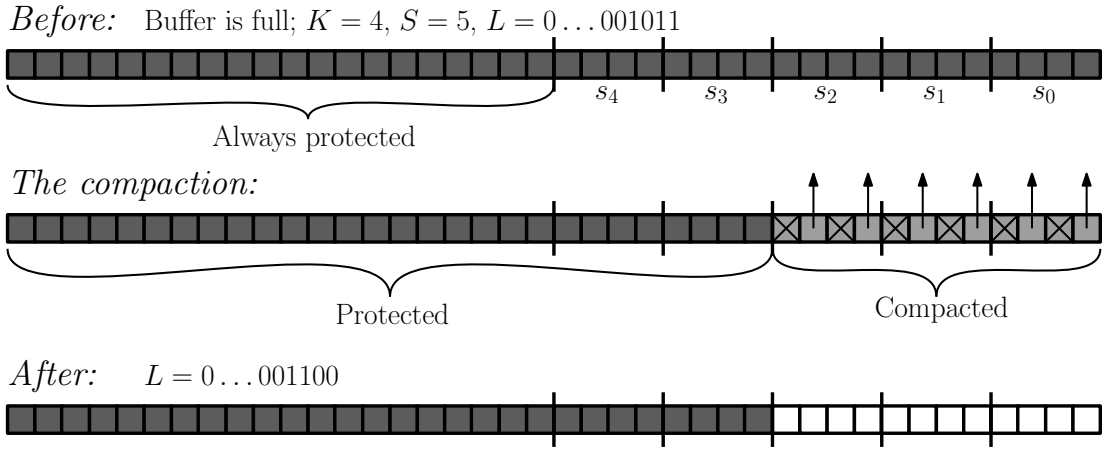

Figure 2.2 Compaction in Jagged Sketch

the compaction schedule L is reset to 0, and K is updated based on the current value of P. Otherwise, L is incremented by 1.

Note that we need $C \geq 4$ for the compaction operation to work properly, however, the current definition permits even $C < 1$. We resolve this problem by assuming $\varepsilon \leq \log^{-\min(1,J)} N$ (Assumption 1) and we get rid of the assumption in Section 2.3.5.

The update operation

The update operation takes the next item from the input stream of the algorithm and sends it to the input stream of the level-0 compactor. The whole stream is processed by a sequence of N update operations.

The rank query

Recall that for any item $y \in \mathcal{U}$, $\mathrm{R}(y)$ is its rank in the input stream. For the rank query y the sketch returns value $\hat{\mathrm{R}}(y)$, which is defined as follows:

$$\hat{\mathrm{R}}(y) \stackrel{\text{def}}{=} \sum_{h=0}^{H-1} 2^h \, \mathrm{R}(y, B_h)$$

Notation

If it is not clear from context, we denote the level of the compactor by lower index and the time by upper index. Thus, P_ℓ^t is the number of already performed compactions of a compactor on level ℓ at time t. However, all the lemmas in Section 2.3 hold at any fixed time t, thus we rarely use the upper index. We stress that $\mathrm{R}(y)$ is fixed at the same time as all the other parameters. For example if we say that $\mathrm{R}(y) = \mathrm{R}(y, I_0)$, it means that for any time t we have $\mathrm{R}(y, \mathcal{S}^t) = \mathrm{R}(y, I_0^t)$.

For any item y, the items *important* with respect to y (or y-important items) are all the items i such that $\mathrm{R}(i) \leq \mathrm{R}(y)$.

Simple observations

Observe that after the compaction is performed, the compactor cannot be overfilled, as the number of protected items is always smaller than C and all the non-protected

items are deleted.

Also observe that the last compactor on level $H-1$ is never overfilled, as every item on level h has weight 2^h (corresponds to 2^h-1 deleted items and itself). If there would be more than $C > \varepsilon^{-1}$ items in B_{H-1}, the input stream of the sketch would be longer than $\varepsilon^{-1} \cdot 2^{H-1} = N$, which is a contradiction.

For the compaction operation, let us note that the number of sections of the right half of the buffer indeed is $S = C/(2K)$ as defined above. Immediately after a reset at time t, we have $S = 2\,\overline{\log}(P^t)$ and by the compaction schedule, the number of compactions before the next reset is $2^{S-1} = (P^t)^2/2$. Thus, the logarithm of P approximately doubles between any two schedule resets. We use this fact in the proof of Lemma 2.

Let us point out some properties of the compactor as a separate observation:

Observation 1 (Properties of the compactor)**.** *At any time t during the algorithm, for any compactor we have*

$$\begin{aligned}
K^t &= \min_{u \le t} K^u \\
P &\le \frac{2\,|O|}{K} \le \frac{|I|}{K} \\
2^3\sqrt{H \ln(\delta^{-1})}/(\varepsilon F\,\overline{\log}(P)) \le K &\le 2^4\sqrt{H \ln(\delta^{-1})}/(\varepsilon F\,\overline{\log}(P)) \\
2\,\overline{\log}(P) \le \frac{C}{K} &\le 4\,\overline{\log}(P) \\
\frac{1}{3}F_{h+1} \le \frac{1}{2^J}F_{h+1} \le F_h &\le 2^J F_{h+1} \le 3F_{h+2} \\
\frac{1}{5}F_{h+2} \le \frac{1}{3^J}F_{h+2} \le F_h &\le 3^J F_{h+2} \le 5F_{h+2} \\
\frac{1}{3}C_{h+1} \le C_h &\le 3C_{h+1}
\end{aligned}$$

Recall that I and O are the input and output stream of the compactor respectively. All these (in)equalities follow directly from the definition. The first equality says that K is non-increasing in time (because P is non-decreasing). The second says that each compaction sends at least $\frac{1}{2}K$ items to the output (and uses the fact that K is non-increasing). The third line is from the definition of compaction (as we update K during each reset), the fourth follows from the third and the rest just says that F can not differ too much on neighbouring levels (and consequently neither does C).

2.3 Analysis in the static setting

We start by analysing the behaviour of a single compactor (Section 2.3.1). Then, for any item y we define a so-called *critical level* and prove that no y-important item gets above this level with large probability (Section 2.3.2). In Section 2.3.3 we use all the previous work to bound the variance of the $\mathrm{Err}(y)$ so that we can bound the error by the tail bound for sub-Gaussian variables in Section 2.3.4. Finally, in Section 2.3.5 we utilize the correspondence between error and space to get rid of the additional assumption on ε below and obtain the final result presented in Section 2.1.

We use the following assumption on ε only to simplify the analysis, and we remove it in Section 2.3.5.

Assumption 1. *We assume that*

$$\varepsilon \leq \log^{-\min(J,1)}(\varepsilon N).$$

Note that we really need some upper bound on ε, as with $\varepsilon = 1$ we could have $C < 4$ which does not allow us to perform the compaction operation. However, with the assumption we have $C \geq 2^5\sqrt{H \log \delta^{-1}} \geq 4$. Also note that the constants in the statements of the lemmas are far from tight. We show in Chapter 3 that the sketch performs well in practice.

2.3.1 Analysis of a single Compactor

Let y be any item $y \in \mathcal{U}$. We say that a compaction is *important* (with respect to y) if it affects $\hat{\mathrm{R}}(y)$. There is an observation about important compactions that follows directly from the definition of $\hat{\mathrm{R}}(y)$:

Observation 2 (Important compactions)**.** *Compaction is important with respect to item y if and only if the rank of y among the compacted (non-protected) items is odd. This particularly means that any important compaction includes at least one important item.*

The only lemma of this section bounds the number of important compactions for an item y:

Lemma 2 (Number of important compactions)**.** *At any time u, the number of important compactions with respect to y on level h is at most $2\,\mathrm{R}(y, I_h^u)/K_h^u$.*

Proof. For the whole proof let us limit ourselves to a single compactor on level h and a fixed item y. If $K^u \leq 2$, then the lemma directly follows from the second part of Observation 2. For the rest of the proof, we assume that $K^u \geq 2$.

It suffices to assign at least K^u important items in I^u to each important compaction in such a way that no item is assigned more than once.

At any previous time t during the run of the algorithm, let us divide the buffer into sections of size K^t indexed from right to left starting from index zero (so that the section 0 equals $B[C - K^t : C - 1]$). Let us index the bits of L from right to left starting from index 1, so that section i corresponds to bit i (recall that section 0 participates in every compaction). Imagine the buffer as always sorted (as we sort it before any compaction) and let *special compaction* be any compaction that happens immediately before a schedule reset. Note that $K^t \geq K^u$.

Non-special compactions Let us first choose an arbitrary non-special important compaction c and denote by j the highest-indexed section that participated in this compaction. After the compaction, all the items in section $j+1$ are important (by Observation 2). Before the compaction, the bit $j+1$ is set to zero and all the bits $\{j, \ldots, 1\}$ are set to one. After the compaction, the bit $j+1$ is set to one, and all the bits $\{j, \ldots, 1\}$ are set to zero. We assign the section $j+1$ (but not yet the items in it) to the compaction c. We claim that before there is some

compaction d that includes section $j+1$, section $j+1$ contains important items only. This is because the only reason for an item i to be moved out of section $j+1$ is that there comes a new item smaller than i and so i is shifted to the right in the buffer. Also note that in the time between compactions c and d there cannot be a compaction that includes section j, as the bit $j+1$ is set to 1 so the compaction would also include section $j+1$.

If there is no such compaction d that includes section $j+1$ after the compaction c, the bit $j+1$ stays set to the end of the algorithm and we assign the K^t items that are present after the end of the algorithm in the section $j+1$ to the compaction c.

If there is a compaction d that includes section $j+1$ after the compaction c, we assign to compaction c the $K^u = K^t$ items that are present in section $j+1$ immediately before compaction d. During compaction d, the items are removed from the buffer and bit $j+1$ is set to zero.

Note that we never assign the same section twice, as the assigned section is represented by a set bit in the compaction schedule and the bit is set to zero only after the items from the assigned section are assigned and immediately removed. Also note that immediately before the schedule reset, there is only one set bit in the schedule and it was set by the special compaction that happened immediately before the reset so it does not correspond to any assigned section and can be safely reset to zero.

Special compactions To any special compaction, we simply assign some K^u items remaining in the left half of the buffer at time u. If there was ever an y-important compaction, all the items in the left half of the buffer are important. It remains to prove that the number of items in the left half is at least K^u times the number of special compactions – in other words, that the number of special compactions is at most $C^u/(2K^u)$.

The number of special compactions is at most the number of resets as a special compaction happens immediately before one. Immediately after a reset, let us say at time v, the right half of the buffer has exactly $S^v = C^v/(2K^v) = 2\,\overline{\log}(P^v)$ sections, by the definition of C and K. By the definition of compaction schedule, the number of compactions to be done before the next reset is $2^{S^v-1} \geq (P^v)^2/2$. Hence, the number of resets before time u is at most $2\log\overline{\log}(P^u)$, which is smaller than $C^u/(2K^u) = 2\,\overline{\log}(P^u)$. This concludes the proof.

□

2.3.2 The critical level

First, we prove that low-ranked items cannot move too high in the sketch. Specifically, for any item y, we define a critical level $H(y)$ and prove that with high probability $\mathrm{R}(y, O_{H(y)}) = 0$.

Definition 3 (Critical level)**.** *For any item $y \in \mathcal{U}$ at any time we define $H(y)$ as the smallest $h \geq 0$ such that*

$$2 \cdot \frac{\mathrm{R}(y)}{2^h} \leq \frac{C_h}{2}.$$

The next observation follows from the minimality of $H(y)$:

Observation 3. *For any item $y \in \mathcal{U}$ and any $H(y) > 0$ at any time we have*

$$C_{H(y)-1} < 2^{3-H(y)} \operatorname{R}(y).$$

By putting together the definition of the critical level and Observation 3 we obtain tight bounds on $H(y)$ (note that the max function is there for the case of $H(y) = 0$ which is not covered by Observation 3). We use these bounds in Section 2.3.4 for bounding the error of the sketch.

Observation 4 (Bounds on the critical level)**.** *For any item $y \in \mathcal{U}$ at any time we have*

$$\max\left(\log \frac{2^3 \operatorname{R}(y)}{C_{H(y)}}, 0\right) - 1 \leq H(y) \leq \max\left(\log \frac{2^3 \operatorname{R}(y)}{C_{H(y)}}, 0\right) + 1.$$

Let us now state the main lemma of this section.

Lemma 3 (Ranks decrease exponentialy)**.** *For any item $y \in \mathcal{U}$ and any $0 \leq h \leq H(y)$ at any time, with probability at least $1 - \delta$ we have*

$$\operatorname{R}(y, I_h) \leq 2 \cdot \frac{\operatorname{R}(y)}{2^h}.$$

Lemma 3 together with the definition of critical level and the fact that smallest $\frac{1}{2}C$ items always stay in the buffer immediately gives us an important corollary:

Corollary 2 (Property of the critical level)**.** *For any item $y \in \mathcal{U}$ at any time, with probability at least $1 - \delta$ we have*

$$\operatorname{R}(y, O_{H(y)}) = 0.$$

Before we prove the lemma, we need a fact about so-called sub-Gaussian variables.

Definition 4 (Sub-Gaussian variables)**.** *Let X be a zero-mean random variable with variance $\operatorname{Var}[X]$. We say that X is* sub-Gaussian*, if for any $s \in \mathbb{R}$ we have*

$$\mathsf{E}[\exp(sX)] \leq \exp\left(-\frac{1}{2}s^2 \operatorname{Var}[X]\right)$$

Fact 1 (Properties of sub-Gaussian variables – Lemma 1.3 in [Rig15])**.** *A weighted sum of independent zero-men sub-Gaussian random variables is a zero-mean sub-Gaussian random variable. For every zero-mean sub-Gaussian random variable X and any $a > 0$ we have*

$$\mathsf{P}[X > a] \leq \exp\left(-\frac{a^2}{2\operatorname{Var}[X]}\right) \qquad \text{and} \qquad \mathsf{P}[X < a] \leq \exp\left(-\frac{a^2}{2\operatorname{Var}[X]}\right).$$

Proof of Lemma 3. We prove for all $h \leq H(y)$ that if for all $\ell < h$ we have

$$\operatorname{R}(y, I_\ell) \leq 2\frac{\operatorname{R}(y)}{2^\ell}, \tag{2.1}$$

then with probability $1 - 2^{h-H(y)-1}\delta$ we have

$$\mathrm{R}(y, I_h) \le 2\frac{\mathrm{R}(y)}{2^h}. \tag{2.2}$$

This is sufficient to prove the lemma as by the union bound, Equation (2.2) is simultaneously true for all h with probability $1 - \delta\sum_{h=0}^{H(y)} 2^{h-H(y)-1} > 1 - \delta$. For $h \le 1$, Equation (2.2) is trivially true with probability 1. For the rest of the proof, let us assume $1 < h \le H(y)$.

Let us investigate the dependence of $\mathrm{R}(y, I_\ell)$ on $\mathrm{R}(y, I_{\ell-1})$. Recall that $\mathrm{R}(y, I_\ell) = \mathrm{R}(y, O_{\ell-1})$ so we are comparing the input and the output of the compactor on level $\ell - 1$. Some of the items from $I_{\ell-1}$ can stay in the buffer $B_{\ell-1}$, the rest are subject to a compaction. Recall that an important compaction is a compaction that affects $\mathrm{Err}(y)$ and by Observation 2 these are exactly those compactions where the rank of y among the compacted items is odd. This means that any not-important compaction always sends half of the important items to the output, and any important compaction always sends one important item more or less than half with equal probability. Let us denote the number of important compactions on level ℓ by m_ℓ. Neglecting the items staying in the buffer, we can bound the rank of y in the output stream as

$$\mathrm{R}(y, I_\ell) = \mathrm{R}(y, O_{\ell-1}) \le \frac{1}{2}(\mathrm{R}(y, I_{\ell-1}) + \mathrm{Binomial}(m_{\ell-1})),$$

where $\mathrm{Binomial}(n)$ is a sum of n independent random variables taking values from $\{-1, 1\}$ with equal probability.

Let us use this bound recursively for all $\mathrm{R}(y, I_\ell)$ where $0 \le l \le h$. Let $Y_0 \stackrel{\text{def}}{=} \mathrm{R}(y)$ and for $0 < \ell \le h$ let

$$Y_\ell \stackrel{\text{def}}{=} \frac{1}{2}(Y_{\ell-1} + \mathrm{Binomial}(m_{\ell-1})).$$

It follows that $\mathrm{R}(y, I_h) \le Y_h$. Thus, it suffices to prove that with probability $1 - 2^{h-H(y)-1}\delta$ we have

$$Y_h \le 2^{-h+1}\,\mathrm{R}(y).$$

By unrolling the definition of Y_h we obtain

$$Y_h = 2^{-h}\,\mathrm{R}(y) + \sum_{\ell=0}^{h-1} 2^{-h+\ell}\,\mathrm{Binomial}(m_\ell).$$

Note that the first summand has a fixed value and the second summand is a zero-mean sub-Gaussian variable by Fact 1. Let us denote the second summand by $Z_h \stackrel{\text{def}}{=} \sum_{\ell=0}^{h-1} 2^{-h+\ell}\,\mathrm{Binomial}(m_\ell)$. It suffices to prove that

$$\mathsf{P}[Z_h > 2^{-h}\,\mathrm{R}(y)] \le 2^{h-H(y)-1}\delta.$$

To invoke the second part of Fact 1, we must bound the variance of Z_h. By Lemma 2 we have $m_\ell \le 2\,\mathrm{R}(y, I_\ell)/K_\ell$ and by (2.1) we have $\mathrm{R}(y, I_\ell) \le 2^{-\ell+1}\,\mathrm{R}(y)$ for each $\ell < h$. That gives us $m_\ell \le 2^{-\ell+2}\,\mathrm{R}(y)/K_\ell$. Using the fact that $\mathrm{Var}[\mathrm{Binomial}(n)] = n$, we can bound the variance of Z_h:

$$\mathrm{Var}[Z_h] \le \sum_{\ell=0}^{h-1} 2^{-2h+2\ell} m_\ell \le \sum_{\ell=0}^{h-1} 2^{-2h+2\ell}\frac{4\,\mathrm{R}(y)}{2^\ell K_\ell} = \sum_{\ell=0}^{h-1}\frac{2^{\ell+2}\,\mathrm{R}(y)}{2^{2h}K_\ell} \le \frac{2^{-h+2}\,\mathrm{R}(y)}{\min_{\ell<h} K_\ell}$$

Now we can finally use the tail bound for sub-Gaussian variables (Fact 1). For simplicity let us write $K_{\min}$ instead of $\min_{\ell<h} K_\ell$.

$$\begin{aligned}
\mathsf{P}[Z_h > 2^{-h}\,\mathrm{R}(y)] &< \exp\left(-\frac{2^{-2h}\,\mathrm{R}^2(y)}{2\,\mathrm{Var}[Z_h]}\right) \\
&\le \exp\left(-\frac{2^{-2h}\,\mathrm{R}^2(y)K_{\min}}{2^{-h+3}\,\mathrm{R}(y)}\right) \\
&= \exp\left(-2^{-h-3}\,\mathrm{R}(y)K_{\min}\right) \\
&= \exp\left(-2^{-h-6+H(y)}2^{3-H(y)}\,\mathrm{R}(y)K_{\min}\right)
\end{aligned}$$

We invoke Observation 3 to substitute $C_{H(y)-1}$ for $2^{3-H(y)}\,\mathrm{R}(y)$:

$$\mathsf{P}[Z_h > 2^{-h}\,\mathrm{R}(y)] < \exp\left(-2^{-h-5+H(y)}C_{H(y)-1}K_{\min}\right) \tag{2.3}$$

Let us bound the expression $C_{H(y)-1}K_{\min}$. By definition, we have

$$C_{H(y)-1}K_\ell \ge \frac{2^8 H\ln(\delta^{-1})}{\varepsilon^2 F_{H(y)}F_\ell\,\overline{\log}(P_\ell)}$$

for arbitrary level ℓ.

We use Assumption 1 which says that $\varepsilon \le \log^{-\min(J,1)}(\varepsilon N) \le H^{-\min(J,1)}$ together with the fact that from definition we have $F_i \le \min(H, d^J) \le \min(H, H^J) \le H^{\min(1,J)}$:

$$C_{H(y)-1}K_\ell \ge \frac{2^8 H\ln(\delta^{-1})}{\varepsilon^2 F_{H(y)}F_\ell\,\overline{\log}(P_\ell)} \ge \frac{2^8 H\ln(\delta^{-1})}{H^{2\min(J,1)}H^{-2\min(J,1)}\,\overline{\log}\,P_\ell} \ge \frac{2^8 H\ln(\delta^{-1})}{\overline{\log}\,P_\ell}$$

And we notice that

$$\overline{\log}\,P_\ell \le \overline{\log}\,\frac{N}{2^\ell K_\ell} \le \overline{\log}(\varepsilon N F_\ell\,\overline{\log}\,P_\ell) \le \overline{\log}(\varepsilon N) + \overline{\log}\,\overline{\log}(\varepsilon N) + \overline{\log}\,\overline{\log}\,P_\ell,$$

hence $\overline{\log}\,P \le 2\log(\varepsilon N) \le 2H$. This implies that

$$C_{H(y)-1}K_{\min} \ge 2^7\ln(\delta^{-1}).$$

Now we can get back to (2.3) to finish the proof:

$$\begin{aligned}
\mathsf{P}[Z_h > 2^{-h}\,\mathrm{R}(y)] &< \exp(-2^{-h-6+H(y)}2^7\ln(\delta^{-1})) \\
&\le \exp(-2^{-h+H(y)+1}\ln(\delta^{-1})) \\
&= \delta^{2^{H(y)-h+1}} \le \delta 2^{-H(y)+h-1}
\end{aligned}$$

In the last inequality, we used the fact that $\delta < 1/2$. This concludes the proof. □

2.3.3 Bounding the variance

We plan to bound the variance of the error of the whole sketch so we can apply the tail bound for sub-Gaussian variables in Section 2.3.4. However, before we start bounding the variance, we need to prove one more lemma. Intuitively, it just says that K decreases only polynomially as we go up the levels. This seems true at first sight, but the proof of the lemma is quite technical.

Lemma 4. *For any two levels $0 \leq h \leq k < H$ at any time we have*

$$K_k \leq 50(k-h+4)^{1+J} K_h.$$

Proof. Let there be some expression x such that

$$K_k \leq x K_h.$$

Our goal is to determine the value of x. By the definition of K it is enough to prove

$$F_h \, \overline{\log}(P_h) \leq x F_k \, \overline{\log}(P_k).$$

It suffices to find the values of y and z such that $x = yz$ and

$$F_h \leq y F_k; \; \overline{\log}(P_h) \leq z \, \overline{\log}(P_k).$$

Let us start with the variable y. Let d be the distance of level k from the closest important level. It follows that the distance of h from the closest important level is at most $d + k - h$ (the triangle inequality). Now we obtain simple bounds on F_k and F_h which follow directly from the definition of F:

$$F_k \geq \overline{\min(H, d^J)}$$
$$F_h \leq \overline{\min(H, (d+k-h)^J)} \leq \min(H, (d+k-h+1)^J)$$

It is enough to set y such that

$$y \geq \frac{\min(H, (d+k-h+1)^J)}{\overline{\min(H, d^J)}}.$$

We set the value of y to

$$y = (k-h+4)^J,$$

which satisfies the inequality.

Now let us proceed to variable z. It needs to satisfy $\overline{\log}(P_h) \leq z \, \overline{\log}(P_k)$. Let w be the highest level such that $|I_w| \neq 0$. We first find some z_0 such that if $k \leq w-2$ we have $\overline{\log}(P_h) \leq z_0 \, \overline{\log}(P_k)$ and some z_1 such that $\overline{\log}(P_{w-2}) \leq z_1 \, \overline{\log}(P_w)$. Then we can set $z = z_0 z_1$.

Let us start with z_0. As in the previous cases, we use suitable bounds on $\overline{\log}(P_i)$ and for them, we need a few auxiliary statements:

$$C_\ell < \frac{6}{7} |I_\ell| \qquad \forall 0 \leq \ell < w-1 \tag{2.4}$$

$$|I_\ell| > \frac{1}{14} |I_{\ell-1}| \qquad \forall 0 < \ell < w \tag{2.5}$$

For the first statement note that if $C_\ell \geq \frac{6}{7} |I_\ell|$, we would have $|I_{\ell+1}| = |O_\ell| \leq \frac{1}{3} C_\ell$ and as by the basic properties of a compactor (Observation 1) $C_{\ell+1} \geq \frac{1}{3} C_\ell$, level $\ell + 1$ would be the last level, which contradicts $\ell \leq w - 2$. The second statement follows from the first one as $|I_\ell| \geq \frac{1}{2}(|I_{\ell-1}| - C_{\ell-1})$.

We also need an observation about binary counters – when starting from zero, after i increments the number of times we flipped a bit from 1 to 0 is at most i. This is because each such bit has to be first set to 1 and we do one such setting with each increment.

Now let us bound $\overline{\log}(P_h)$:

$$\overline{\log}(P_h) \leq \overline{\log} \frac{|I_h|}{K_h} \leq \overline{\log} \frac{2^{4(k-h)}\,|I_k|}{K_h} \leq \overline{\log} \frac{|I_k|}{K_h} + 4(k-h)$$

In the second inequality we used (2.5). For bounding $\overline{\log}(P_k)$ let K_k^{max} be the maximum value of K_k achieved so far:

$$\begin{aligned}\overline{\log}(P_k) &\geq \overline{\log} \frac{|I_k| - C_k}{2K_k^{\max}} \geq \overline{\log} \frac{|I_k|}{14K_k^{\max}} \geq \overline{\log} \frac{|I_k|}{14K_k \overline{\log} P_k} \geq \overline{\log} \frac{|I_k|}{14xK_h \overline{\log} P_k} \\ &\geq \overline{\log} \frac{|I_k|}{K_h} - \log(14x \overline{\log}(P_k))\end{aligned}$$

In the first inequality, we used the observation about binary counters (that an average number of items participating in a compaction is at most $2K$), in the second we used (2.4), in the third we used the definition of K and in the last inequality we used the meaning of the variable x. From the two bounds, we conclude that

$$\overline{\log}(P_h) \leq 4(k-h) + \log(14x \overline{\log}(P_k)) + \overline{\log}(P_k).$$

Hence, it suffices to find a value of z_0 such that

$$4(k-h) + \log(14yz_0 \overline{\log}(P_k)) + \overline{\log}(P_k) \leq z_0 \overline{\log}(P_k),$$

and we claim that the right choice is $z_0 = 5(k-h+4)$. After substituting for z_0 and $y = (k-h+4)^J$ we get

$$4(k-h) + \log(70(k-h+4)^{J+1} \overline{\log}(P_k)) + \overline{\log}(P_k) \leq 5(k-h+4) \overline{\log}(P_k)$$

and after rearrangement we obtain

$$\begin{aligned}&4(k-h) + \log(70) + (J+1)\log(k-h+4) + \log \overline{\log}(P_k) + \overline{\log}(P_k) \leq \\ &(4(k-h) + \quad 7 \quad + \quad (k-h+13) \quad + \quad 1 \quad + \quad 1 \quad) \overline{\log}(P_k),\end{aligned}$$

which is true as $\overline{\log}(P_k) \geq 1$ and $(J+1)\log(k-h+4) \leq (k-h+13)$ (as $J \leq 1.4$ from definition).

It remains to find a value of z_1 such that $\overline{\log} P_{w-2} \leq z_1 \overline{\log} P_w$. As $\overline{\log}(P_w) = 1$ by definition, we simply need to bound the value $\overline{\log}(P_{w-2})$ from above. We have

$$\begin{aligned}P_{w-2} \leq \frac{2\,|I_{w-1}|}{K_{w-2}} \leq \frac{2(C_{w-1} + 2C_w)}{K_{w-2}} &\leq \frac{2(2^J C_{w-2} + 2 \cdot 3^J C_{w-2})}{K_{w-2}} \\ &\leq \frac{24C_{w-2}}{K_{w-2}} \leq 96 \overline{\log}(P_{w-2}).\end{aligned}$$

The second inequality follows from the fact that w is the highest level with nonempty input stream, the rest are the basic properties of a compactor (Observation 1). Hence, we have $\overline{\log}(P_{w-2}) < 10$, so it suffices to set $z_1 = 10$ and we have $z = z_0 z_1 = 50(k-h+4)$ and $x = yz = 50(k-h+4)^{J+1}$ which concludes the proof. □

Before we bound the variance of the error, let us define $\mathrm{Err}_h(y)$ as a sum of the ± 1 errors induced by y-important compactions on level h.

Definition 5 ($\mathrm{Err}_h(y)$)**.** *For any item $y \in \mathcal{U}$ and any level $0 \le h < H$ at any time let*

$$\mathrm{Err}_h(y) \stackrel{\mathrm{def}}{=} \mathrm{R}(y, I_h) - 2\,\mathrm{R}(y, O_h) - \mathrm{R}(y, B_h).$$

This gives us a more suitable definition of $\mathrm{Err}(y)$:

Observation 5 (Alternative definition of $\mathrm{Err}(y)$)**.** *For any item $y \in \mathcal{U}$, we have*

$$\mathrm{Err}(y) = \sum_{h=0}^{H-1} 2^h\,\mathrm{Err}_h(y).$$

Now we finally state the lemma about the variance of error.

Lemma 5 (Variance of the error)**.** *For any item y at any time, $\mathrm{Err}(y)$ is a zero-mean sub-Gaussian random variable and with probability $1-\delta$, we have*

$$\mathrm{Var}[\mathrm{Err}(y)] \le 2^{20}\frac{\mathrm{R}^2(y)}{K_{H(y)}C_{H(y)}}.$$

Proof. For the whole proof, let us fix an arbitrary item y and for level h let us denote the number of y-important compactions by m_h.

For the first part of the lemma, note that $\mathrm{Err}_h(y)$ is a sum of m_h independent random variables each taking on a value -1 or 1 with equal probability. Hence, by Fact 1, $\mathrm{Err}_h(y)$ is a zero-mean sub-Gaussian random variable with $\mathrm{Var}[\mathrm{Err}_h(y)] = m_h$. Thus, $\mathrm{Err}(y)$ itself is a weighted sum of independent zero-mean sub-Gaussian variables and so it is also a zero-mean sub-Gaussian variable.

For bounding the variance of $\mathrm{Err}(y)$ we use Corollary 2, which implies that with probability $1-\delta$ for any $h > H(y)$ we have $\mathrm{Err}_h(y) = 0$. Then we use the fact that $\mathrm{Var}[\mathrm{Err}_h(y)] = m_h$ together with Lemma 2 which says that $m_h \le 2\,\mathrm{R}(y, I_h)/K_h$, then Lemma 3 to get that $\mathrm{R}(y, I_h) \le 2^{-h+1}\,\mathrm{R}(y)$, the freshly proven Lemma 4 to bound $K_{H(y)}$ by $50(H(y)-h+4)^{J+1}K_h$ and finally Observation 3 together with the fact that $C_i < 3C_{i+1}$ to bound $C_{H(y)} < 2^{4-H(y)}\,\mathrm{R}(y)$.

With probability $1-\delta$ we have

$$\begin{aligned}
\mathrm{Var}[\mathrm{Err}(y)] &= \sum_{h=0}^{H(y)} 2^{2h}\,\mathrm{Var}[\mathrm{Err}_h(y)] \le \sum_{h=0}^{H(y)} 2^{2h}\frac{2\,\mathrm{R}(y, I_h)}{K_h} \le \sum_{h=0}^{H(y)} 2^{h+1}\frac{2\,\mathrm{R}(y)}{K_h} \\
&\le \sum_{h=0}^{H(y)} 2^{h+2}\frac{\mathrm{R}(y)50(H(y)-h+4)^{J+1}}{K_{H(y)}} \\
&= \frac{4\cdot 50\cdot \mathrm{R}(y)}{K_{H(y)}}\sum_{h=0}^{H(y)} 2^h(H(y)-h+4)^{J+1} \\
&\le \frac{4\cdot 50\cdot \mathrm{R}(y)}{K_{H(y)}}\cdot 161\cdot 2^5\cdot 2^{-5}\cdot 2^{H(y)+1} \\
&\le 2^{20}\cdot\frac{\mathrm{R}^2(y)}{K_{H(y)}C_{H(y)}}
\end{aligned}$$

Note that we used the fact that $\sum_{i=0}^{n} 2^i(n-i+4)^3 \le 161\cdot 2^{n+1}$.

□

2.3.4 The error bound

Now we are ready to bound the error of the sketch. Note that the statement of the lemma differs from the Theorem 1 by a factor of $\log^{-\min(1,J)}(\varepsilon N)$. This difference is resolved in Section 2.3.5.

Lemma 6. *For any $y \in \mathcal{U}$ at any time, if $\mathrm{R}(y) = 0$ we have $\mathrm{Err}(y) = 0$ and if $\mathrm{R}(y) > 0$ we have*

$$\mathrm{Err}(y) \leq 2^{15} \varepsilon \, \mathrm{R}(y) \min_{r \in R} \left(\left| \log \frac{\mathrm{R}(y)}{r} \right|^J, \log(\varepsilon N) \right) \sqrt{\frac{\overline{\log} \frac{N}{\mathrm{R}(y)}}{\log(\varepsilon N)}}$$

with probability $1 - \delta$.

Proof. Let us fix any item y. If $\mathrm{R}(y) = 0$ the sketch answers 0 by definition. For the rest of the proof let us assume $\mathrm{R}(y) > 0$.

Lemma 5 tells us that $\mathrm{Err}(y)$ is a zero-mean sub-Gaussian random variable and it bounds its variance, thus we use the tail bound for sub-Gaussian variables (Fact 1) with setting $a = \sqrt{2 \ln(2\delta^{-1}) \operatorname{Var}[\mathrm{Err}(y)]}$:

$$\begin{aligned} \mathsf{P}\left[|\mathrm{Err}(y)| \geq a\right] &= \mathsf{P}\left[|\mathrm{Err}(y)| \geq \sqrt{2 \ln(2\delta^{-1}) \operatorname{Var}[\mathrm{Err}(y)]}\right] \\ &< 2 \exp\left(-\frac{2 \ln(2\delta^{-1}) \operatorname{Var}[\mathrm{Err}(y)]}{2 \operatorname{Var}[\mathrm{Err}(y)]} \right) = 2 \exp\left(-\ln\left(2\delta^{-1}\right) \right) \\ &= \delta \end{aligned}$$

Now we continue bounding the error, conditioning on the event that $|\mathrm{Err}(y)| \leq a$. We start by plugging in the estimate on $\operatorname{Var}[\mathrm{Err}(y)]$ from Lemma 5 and by unrolling the definition of K, C and H.

$$\begin{aligned} |\mathrm{Err}(y)| &< \sqrt{2 \ln(2\delta^{-1}) \operatorname{Var}[\mathrm{Err}(y)]} \leq \sqrt{2 \ln(2\delta^{-1}) 2^{20} \frac{\mathrm{R}^2(y)}{K_{H(y)} C_{H(y)}}} \\ &= 2^{10} \, \mathrm{R}(y) \sqrt{2 \ln(2\delta^{-1}) \frac{\varepsilon^2 F_{H(y)}^2 \, \overline{\log}(P_{H(y)})}{H \ln(\delta^{-1})}} \\ &\leq 2^{10} \, \mathrm{R}(y) \sqrt{2(\ln(\delta^{-1}) + 1) \frac{\varepsilon^2 F_{H(y)}^2 \, \overline{\log}(P_{H(y)})}{(\log(\varepsilon N) + 1) \ln(\delta^{-1})}} \\ &\leq 2^{10} \varepsilon \, \mathrm{R}(y) F_{H(y)} \sqrt{2 \frac{\overline{\log}(P_{H(y)})}{\log(\varepsilon N)}} \end{aligned}$$

Now it suffices to prove that

$$F_{H(y)} \leq \min_{r \in R} \left(\left(\left| 2 \log \frac{\mathrm{R}(y)}{r} \right| + 2 \right)^J, H \right) \leq 2^4 \min_{r \in R} \left(\left| \log \frac{\mathrm{R}(y)}{r} \right|^J, \log(\varepsilon N) \right)$$

and that $\overline{\log}(P_{H(y)}) \leq 2 \log \frac{N}{\mathrm{R}(y)}$. The result then follows by a simple substitution. For both we need Observation 4 which says that

$$H(y) = \max \left(\log \frac{2^3 \, \mathrm{R}(y)}{C_{H(y)}}, 0 \right) \pm 1$$

Let us start by proving that $\overline{\log}(P_{H(y)}) \leq 2\,\overline{\log}\,\frac{N}{\mathrm{R}(y)}$. We invoke Observation 4 and use the basic properties of a compactor (Observation 1).

$$\overline{\log}(P_{H(y)}) \leq \overline{\log}\,\frac{\left|I_{H(y)}\right|}{K_{H(y)}} \leq \overline{\log}\,\frac{N}{2^{H(y)}K_{H(y)}} \leq \overline{\log}\,\frac{N}{\frac{2^2\,\mathrm{R}(y)}{C_{H(y)}}K_{H(y)}} \leq \overline{\log}\,\frac{N\,\overline{\log}(P_{H(y)})}{\mathrm{R}(y)}$$

We conclude that $\overline{\log}(P_{H(y)}) \leq 2\,\overline{\log}\,\frac{N}{\mathrm{R}(y)}$.

Now it only remains to prove that $F_{H(y)} \leq \min\left(\left(\left|2\log\frac{\mathrm{R}(y)}{r}\right|+2\right)^J, H\right)$ for any $r \in R$. If $|H(y)-\mathcal{L}(r)| \leq 2$, we have $F_{H(y)} \leq 2^J \leq \left(\left|2\log\frac{\mathrm{R}(y)}{r}\right|+2\right)^J$. Otherwise, we have $F_{H(y)} \leq |H(y)-\mathcal{L}(r)|^J$ and by the definition of $\mathcal{L}(r)$ and by Observation 4 we get

$$\begin{aligned}
F_{H(y)} &\leq \left|\max\left(\log\frac{\varepsilon\,\mathrm{R}(y)F_{H(y)}}{2^2\sqrt{H\log\delta^{-1}}},0\right)-\max\left(\log\frac{\varepsilon r}{2^2\sqrt{H\log\delta^{-1}}},0\right)\pm 2\right|^J \\
&\leq \left(\left|\log\frac{\varepsilon\,\mathrm{R}(y)F_{H(y)}}{2^2\sqrt{H\log\delta^{-1}}}-\log\frac{\varepsilon r}{2^2\sqrt{H\log\delta^{-1}}}\right|+2\right)^J \\
&= \left(\left|\log\frac{\mathrm{R}(y)F_{H(y)}}{r}\right|+2\right)^J.
\end{aligned}$$

Again we have $F_{H(y)} \leq \left(\left|2\log\frac{\mathrm{R}(y)}{r}\right|+2\right)^J$. From the definition of F, we have $F_{H(y)} \leq H$, which concludes the proof. □

2.3.5 The space bound

Asymptotic space estimate

Let us first calculate the amount of space the sketch occupies. First let us note that there are only constantly many of important ranks, so the space is asymptotically bounded by the size of the sketch with just one important rank and let C be the capacity of the compactor on its only important level. By the definition of F, the size of the sketch can be bounded by

$$\texttt{SPACE} \leq \Theta(1)\sum_{h=1}^{H}\frac{C}{\overline{\min(h^J,H)}} \in \mathcal{O}\left(C\sum_{h=1}^{H}\overline{\min(h^J,H)}^{-1}\right).$$

The behaviour of the sum depends on the value of J. For $J=1$, this is harmonic series and we have

$$\begin{aligned}
\texttt{SPACE} \in \mathcal{O}\left(C\log H\right) &= \mathcal{O}\left(C\log\log(\varepsilon N)\right) \\
&= \mathcal{O}\left(\varepsilon^{-1}\sqrt{\log(\varepsilon N)\log\delta^{-1}}\log\log(\varepsilon N)\right),
\end{aligned}$$

for $J>1$ the sum equals a constant and we have

$$\texttt{SPACE} \in \mathcal{O}(C) = \mathcal{O}(\varepsilon^{-1}\sqrt{\log(\varepsilon N)\log\delta^{-1}})$$

and for $0 \leq J < 1$ the sum is $\Theta(H^{1-J})$ which gives us

$$\texttt{SPACE} \in \mathcal{O}(C\log^{1-J}(\varepsilon N)) = \mathcal{O}(\varepsilon^{-1}\sqrt{\log(\varepsilon N)\log\delta^{-1}}\log^{1-J}(\varepsilon N)).$$

Elimination of the assumption on ε

Now we exploit the fact that both space and error depend on ε to get rid of Assumption 1 and at the same time obtain a version of Jagged Sketch that is better compared to ReqSketch and where also the space does not depend on J (except for $J = 1$).

We eliminate the assumption that $\varepsilon \leq \log^{-\min(1,J)}(\varepsilon N)$. The user gives us $\varepsilon \leq 1$ and we simply set

$$\varepsilon' = \frac{\varepsilon}{2^{15} \log^{\min(1,J)}(\varepsilon N)}$$

and build the sketch with parameter ε'. From the space estimates above and Lemma 6 we obtain Theorem 1 by a simple substitution:

Theorem 1 (The main theorem)**.** *Given any $N \in \mathbb{N}$, set R s.t. $|R| \in \mathcal{O}(1)$ and $R \subset \{1, \ldots, N\}$, and real parameters $0 < \delta \leq 0.5$, $0 < \varepsilon \leq 1$ and $0 \leq J \leq 1.4$, there is a comparison-based randomized algorithm that processes an input stream of length at most N and at any time answers any rank query y with probability $1 - \delta$ with error*

$$\mathrm{Err}(y) \leq \varepsilon \, \mathrm{R}(y) \cdot \sqrt{\frac{\overline{\log} \frac{N}{\mathrm{R}(y)}}{\log(\varepsilon N)}} \cdot \frac{\min_{r \in R} \left(\overline{\left| \log \frac{\mathrm{R}(y)}{r} \right|}^{J}, \log(\varepsilon N) \right)}{\log^{\min(1,J)}(\varepsilon N)}$$

in space

$$\texttt{SPACE} \in \mathcal{O}_J \left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon N) \cdot \sqrt{\log \frac{1}{\delta}} \right)$$

in case of $J \neq 1$ and in space

$$\texttt{SPACE} \in \mathcal{O}_J \left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon N) \cdot \sqrt{\log \frac{1}{\delta}} \cdot \log\log(\varepsilon N) \right)$$

in case of $J = 1$.

2.3.6 The time complexity

Amortized time of update

First, let us estimate the time spent on level 0 after processing a stream of length N. If we represent the compactor by an appropriate data structure, let us say a balanced binary search tree, removing or adding an item to the compactor both cost time $\mathcal{O}(\log C_0)$. Hence, the time spent on level 0 is $\mathcal{O}(N \log C_0)$, as each item is once added and once removed.

Now observe that time spent on level ℓ is at most $\frac{1}{2^\ell}\mathcal{O}(N \log(C_0 \ell^J))$, as $|I_\ell| \leq |I_0| / 2^\ell$ and $C_\ell \leq C_0 \ell^J$. We conclude that the overall time of processing stream of length N is

$$\sum_{h=0}^{H-1} \frac{1}{2^\ell} \mathcal{O}(N \log(C_0 \ell^J)) \in \mathcal{O}(N \log C_0).$$

Hence the amortized time of the update operation is

$$\mathcal{O}(\log C_0) = \mathcal{O}\left(\log(\varepsilon^{-1}) + \log\log(\varepsilon N) + \log\log(\delta^{-1}) \right).$$

The rank query

If the compactors are represented by binary search trees (as we assumed above), the rank query can be performed by a single query for each of the search trees. Hence, the time complexity of a rank query is

$$\sum_{h=0}^{H-1} \log(C_h) \in \mathcal{O}\left(\log(\varepsilon N)\left(\log(\varepsilon^{-1}) + \log\log(\varepsilon N) + \log\log(\delta^{-1})\right)\right).$$

However, if we plan to perform more rank queries, we can sort all the items saved in the sketch together with their weights and calculate prefix sums of the weights. This precalculation costs us $\mathcal{O}\left(\texttt{SPACE} \cdot \log(\texttt{SPACE})\right)$ and then we can answer a single rank query by a simple binary search over the items in time $\mathcal{O}\left(\log(\texttt{SPACE})\right)$ where `SPACE` is the space bound for the whole sketch from Theorem 1.

The quantile query

For the quantile query, we can use the same auxiliary prefix sums as for the rank query. With these, we can answer any quantile query in time $\mathcal{O}(\log(\texttt{SPACE}))$ by binary search over the prefix sums of the weights.

2.4 Extension to the dynamic setting

In this section, we explain how to make the sketch dynamic and we describe the changes to be made in the analysis to support the dynamic case. We invite the reader to open this section side by side with Sections 2.2 and 2.3 as we only highlight the differences. This section contains only an informal description without the proper quantification of constants and with some simplifying assumptions, however, we try to cover all important changes, thus the proper proofs could be derived from our explanation if necessary. We consider this section to be only provisional as we later plan to make the sketch mergeable, which solves also the dynamic case (as insertion can be interpreted as a merge with a sketch of size one).

For better understanding, we also recommend reading the code of the simple version of our proof-of-concept implementation `jaggedSketchSimple.py`[3], which mostly follows the theoretical description in the dynamic setting and is intended as a substitute to pseudocode.

2.4.1 Changes to the sketch

Description of the changes

The differences from the static version are following: The number N is initially 0 and is incremented with each update of the sketch. The number of compactors H is initially 1. The important ranks must be given in the form of input for the quantile function: $|Q| \in \mathcal{O}(1), Q \subset [0,1]$. The ranks R can be any time computed from the current value of N as $R = \left\{\overline{\lceil Nq \rceil} \mid q \in Q\right\}$. For simplicity, we write $\mathcal{L}(q)$ instead of $\mathcal{L}(\overline{\lceil Nq \rceil})$.

[3] `https://github.com/domestomas/JaggedSketch/blob/main/jaggedSketchSimple.py`

Any time the last compactor performs a compaction, a new compactor must be created on the top of the sketch to receive its output. Let us call this situation a *recalculation*. During recalculation, we first go through the levels bottom-up and for each level, we perform a full compaction – we compact the whole right part of the buffer which triggers a schedule reset. Then the height H of the sketch and the important ranks R are updated and for all the compactors, the scaling factor F, section size K, and capacity C are recalculated based on H and R. Between the recalculations, the sketch behaves as the static version.

Initial observations and notation

First let us assume that between two subsequent recalculations a and b, N changed only by a constant factor.

Conditioning on the assumption, we see from the definition of $\mathcal{L}$ that during the recalculation b, for each $q \in Q$ the important level $\mathcal{L}(q)$ changed only by a constant (also note that $\mathcal{L}^t(q)$ is a non-decreasing function of time). This implies that for each compactor its scaling factor F changed only by a constant factor during b, thus C also changed only by a constant factor.

Moreover, we can realize that before a recalculation, all the compactors are at least half-full. This means that after the recalculation b, all the compactors are at least constant a fraction full.

This altogether implies that between b and the next recalculation c, N changes only by a constant factor, as N equals the weighted sum of the elements of the sketch and the weighted sum changed only by a constant fraction. Thus, all these properties are always true which can be formally proven by induction.

On a particular level h, let us denote $K_h^{\min,t} = \min_{s \le t} K_h^s$ the minimum achieved value of K_h up to time t and analogously $C_h^{\min,t} = \min_{s \le t} C_h^s$. Note that it is no longer true that $K_h^{\min,t} = K_h^t$. It also does not generally hold that $P_h \le 2\,|O_h|\,/K_h$, but we have $P_h \le 2\,|O_h|\,/K_h^{\min}$. Let us summarize this all to a separate observation:

Observation 6. *On any level h at any time t we have*

$$
\begin{aligned}
P &\le \frac{2\,|O|}{K^{\min}} \\
\mathcal{L}^{t-1}(q) \le \mathcal{L}^t(q) &\le \mathcal{L}^{t+1}(q) + \mathcal{O}(1) && \forall q \in Q \\
F^t &\in \Theta(F^{t+1}) \\
C^t &\in \Theta(C^{t+1}) \\
|B| &\in \Omega(C) && \text{If } |O_h| > 0
\end{aligned}
$$

2.4.2 Changes in the analysis

Let us now go through the whole analysis and highlight the differences.

Analysis of a single compactor

We weaken the statement of Lemma 2 – we claim that at any time u the number of important compactions with respect to y on level h is $\mathcal{O}(\mathrm{R}(y, I_h^u)/K_h^{\min})$. Thus

we need to assign to each y-important compaction some $K^{\min}$ y-important items such that each item is assigned at most a constant number of times.

As each recalculation triggers a schedule reset for all the compactors, the analysis of the non-special compactions remains valid. Hence, we just need to resolve the special compactions (note that the original analysis does not work as it is no longer true that all the items in the left half at time u must be y-important and moreover, we also increased the number of schedule resets).

Let us denote the number of sections in one half of the buffer by $S \stackrel{\text{def}}{=} C/(2K)$. We have $S \in \Theta(\overline{\log} P)$. The original number of schedule resets was at most $2 \log \log P$ and we added at most $H - h$ new resets which is $\Theta(\overline{\log} P)$. Thus at any time, the number of sections of the left half of the buffer is up to constant factor the number of already performed special compactions. Thus during any special compaction, we can find some $K^{\min}$ items in the left half of the buffer which were assigned at most constantly many times and directly assign them to the special compaction.

The critical level

We need to modify the definition of the critical level (Definition 3) – we replace C_h by $C_h^{\min}$ in the definition. Thus we define $H(y)$ as the smallest h such as $2\,\mathrm{R}(y)/2^h \leq C_h^{\min}/2$. The reason for this change is that important items can get to level $H(y)$ at some time when the capacity is $C^{\min}$.

The adjusted versions of Observations 3 and 4 are naturally obtained just by substituting C_i by $C_i^{\min}$.

Lemma 3 stays the same, however, we use Lemma 2 in the proof which now holds for a different constant and with $K^{\min}$. The different constants force us to change the constants in the definition of K and C. Using $K^{\min}$ instead of K changes nothing in the proof.

Corollary 2 stays the same, but only because we changed the definition of $H(y)$.

Bounding the variance

Instead of Lemma 4 we newly bound $K_k^{\min} \leq c(k - h + b)^{1+J} K_h^{\min}$ for some constants b and c. The exchange of K_i for $K_i^{\min}$ is due to the upper bound on $\overline{\log} P$ (see Observation 6) and due to the fact that we need this version anyway.

The constants are different because in (2.4) and (2.5) we implicitly assumed that at least half of the buffer is always full. Now this is true at the time of a recalculation, otherwise, we only know that a constant fraction of the buffer is full. In this setting, we get worse constants in (2.4) and (2.5) and thus in the whole lemma. A second change in constants is during the setting of value for z_1. There is again the same issue and in addition, we use the fact that in the last two levels, the values of $K^{\min}$ and K are asymptotically the same.

As an analogy to Lemma 5 we have that $\mathrm{Var}[\mathrm{Err}(y)] \leq c\,\mathrm{R}^2(y)/(K_{H(y)}^{\min} C_{H(y)}^{\min})$ again for some constant c. The proof is completely analogous.

The error bound

By the same argument as in the proof of Lemma 6 we successfully prove that with probability $1-\delta$ at any time t we have

$$|\mathrm{Err}(y)| \leq \Theta(1) \cdot \varepsilon \,\mathrm{R}(y) \max_{u \leq t, v \leq t} \sqrt{\frac{F^u_{H(y)^t}}{\sqrt{H^u}} \cdot \frac{F^v_{H(y)^t}\, \overline{\log}(P^v_{H(y)^t})}{\sqrt{H^v}}}$$

We postpone bounding the expression under the square root to Section 2.4.3.

The space bound

The space bound naturally does not change. We stress we can perform the ε substitution even though we do not know the value of N in advance. We just multiply the definitions of C and K by the expression $H^{\min(1,J)}$. This brings nothing new as they already depended on H. Also note that by the same trick, we can move the $\Theta(1)$ constant from the error estimate to the space bound, as we did in the static version.

2.4.3 Bounding $C^{\min}$ and $K^{\min}$

It remains to bound the value of the expression

$$\max_{u \leq t, v \leq t} \frac{F^u_{H(y)^t}}{\sqrt{H^u}} \cdot \frac{F^v_{H(y)^t}\, \overline{\log}(P^v_{H(y)^t})}{\sqrt{H^v}}.$$

Let us for simplicity denote $H(y)^t$ by h. We have

$$\max_{u \leq t, v \leq t} \frac{F^u_h}{\sqrt{H^u}} \cdot \frac{F^v_h\, \overline{\log}(P^v_h)}{\sqrt{H^v}} \leq \max_{x \leq t}(F^x_h)^2 \cdot \max_{y \leq t} \frac{\overline{\log}(P^y_h)}{\sqrt{H^y}}$$

Bounding F

Let us first bound the value of F^x_h for any $x \leq t$. We have

$$F^x_h = \overline{\min\left(H, |\mathcal{L}(q)^x - h|^J\right)} \leq \overline{|\mathcal{L}(q)^x - h|}^J$$

for some $q \in Q$. The key observation is that we have

$$F^x_h \leq \max\left(F^t_h, h^J\right).$$

This is trivially true if $|\mathcal{L}(q)^x - h| \leq h$. In the other case, we have $\mathcal{L}(q)^x > 2h$ and $\mathcal{L}(q)^x$ is by definition the closest important level at time x. Hence there are no important levels smaller than $\mathcal{L}(q)^x$ at time x and as important levels move only up, we conclude that the closest important level at time $t \geq x$ can not be closer to h than $|\mathcal{L}(q)^x - h|$.

We have already proven in the proof of Lemma 6 that $F^t_h \leq \min_{r \in R} \overline{\left|\log \frac{\mathrm{R}(y)}{r}\right|}^J$ and by Observation 4 we have $h \leq \log \mathrm{R}(y)$. Thus we have

$$\max_{x \leq t}(F^x_h) \leq \max\left(\min_{q \in Q} \overline{\left|\log \frac{\mathrm{R}^t(y)}{\overline{\lceil qN^t \rceil}}\right|}^J, \log^J \mathrm{R}^t(y)\right). \tag{2.6}$$

Bounding $K^{\min}$

Now we bound the expression $\frac{\overline{\log}(P_h^w)}{\sqrt{H^w}}$ for any time $w \le t$. We first bound $\overline{\log} P$ the same way as in Lemma 6. Let $z \le w$ be such a time that $K_h^{\min,w} = K_h^z$. We have (up to constant factors)

$$\begin{aligned}\overline{\log} P_h^w &\le \overline{\log} \frac{|I_h^w|}{K_h^{\min,w}} \le \overline{\log} \frac{N^w}{2^h K_h^{\min,w}} \le \overline{\log} \frac{N^w C_h^{\min,t}}{\mathrm{R}^t(y) K_h^{\min,w}} \le \overline{\log} \frac{N^w C_h^z}{\mathrm{R}^t(y) K_h^z} \\ &\le \overline{\log} \frac{N^w \overline{\log} P_h^z}{\mathrm{R}^t(y)} \le \overline{\log} \frac{N^w \overline{\log} P_h^w}{\mathrm{R}^t(y)}.\end{aligned}$$

Thus we conclude that $\overline{\log} P_h^w \le \Theta(1) \cdot \overline{\log} \frac{N^w}{\mathrm{R}^t(y)}$. Now we just need to bound the expression

$$\frac{\overline{\log} \frac{N^w}{\mathrm{R}^t(y)}}{\sqrt{\log(\varepsilon N^w)}}.$$

This is non-decreasing as a function of N as long as $2\log(\varepsilon N) \ge \log(N/\mathrm{R}^t(y))$. This is true if we assume $\varepsilon \ge N^{-1/2}$ which is a very weak assumption as it is almost always true in practice. Thus we conclude (conditioning on the assumption on ε) that

$$\max_{w \le t} \frac{\overline{\log}(P_h^w)}{\sqrt{H^w}} \le \Theta(1) \cdot \frac{\overline{\log} \frac{N^t}{\mathrm{R}^t(y)}}{\sqrt{\log(\varepsilon N^t)}}. \tag{2.7}$$

The error

Putting together (2.6) and (2.7), we can finish the proof of the lemma to get

$$|\mathrm{Err}(y)| \le \Theta(1) \cdot \varepsilon \, \mathrm{R}(y) \max\left(\min_{q \in Q} \left| \log \frac{\mathrm{R}(y)}{\lceil qN \rceil} \right|^J, \log^J \mathrm{R}(y) \right) \sqrt{\frac{\overline{\log} \frac{N}{\mathrm{R}(y)}}{\sqrt{\log(\varepsilon N)}}}.$$

Hence, after the substitution for $\varepsilon' = \Theta(1)\varepsilon \cdot \log^{-\min(1,J)}(N)$ and conditioning on $\varepsilon \ge N^{-1/2}$, the final error is

$$|\mathrm{Err}(y)| \le \varepsilon \, \mathrm{R}(y) \frac{\sqrt{\overline{\log} \frac{N}{\mathrm{R}(y)}}}{\log^{0.25+\min(1,J)}(\varepsilon N)} \max\left(\min_{q \in Q} \left| \log \frac{\mathrm{R}(y)}{\lceil qN \rceil} \right|^J, \log^J \mathrm{R}(y) \right).$$

Special cases

We can get better error bounds for some special cases.

First, observe that if the closest important level to level h at time t is level 0, then for all times $s \le t$ the closest important level to level h was the level 0. The same holds for important level $H - 1$: if at time t the closest important level to h is $H^t - 1$ then for all times $s \le t$ the closest important level to level h was the level $H^s - 1$. This is true because the function $\mathcal{L}^s(q)$ is non-decreasing in s, thus important levels can move only up.

This observation implies that if the closest important level at time t is 0 or $H^t - 1$, we have

$$\max_{s \le t} F^s = F^t.$$

This particularly means that if $Q \subseteq \{0,1\}$ and $Q \neq \emptyset$ (which is a natural setting), we have

$$|\mathrm{Err}(y)| \leq \varepsilon\,\mathrm{R}(y)\frac{\sqrt{\log \frac{N}{\mathrm{R}(y)}}}{\log^{0.25+\min(1,J)}(\varepsilon N)} \min_{q\in Q} \left|\log \frac{\mathrm{R}(y)}{\lceil qN \rceil}\right|^{J},$$

which differs from the static version only by a factor of $\log^{0.25}(\varepsilon N)$.

Moreover, note that if the closest important level at time t is $H^t - 1$ and at the same time $J \geq 0.5$, we have

$$C^{\min,t} = C^t.$$

This means that if $Q = \{1\}$ and $J \geq 0.5$, we have the same error bound as in the static case.

3 Experiments

In this chapter, we present our proof-of-concept implementation of Jagged Sketch. The implementation serves two purposes. First, it complements the formal description of the algorithm from Section 2.2. We intentionally chose Python as a readable programming language, we included lots of comments inside the code, and we provide a simple version `jaggedSketchSimple.py` which mostly follows the theoretical description. We recommend reading the code as a good check of understanding the algorithm fully.

The main purpose of the implementation is to show that the error/space tradeoff of the sketch is reasonable in practice (as the constants in the proofs are unrealistically large). We measure the space in the number of contained items, which is accurate enough, as apart from them we store only a few variables for each compactor and for the whole sketch. We intentionally ignored the matter of time efficiency, as it is highly dependent on the used programming language and optimizations that are not the focus of this work. The purpose of the tests is not to be exhaustive (they are quite far from that), but only to show that the sketch can be useful in practice.

We start in Section 3.1 by explaining our testing framework in detail, in Section 3.2 we introduce the implementation, and in Section 3.3 we discuss the results of the experiments and compare them to ReqSketch and KLL.

3.1 Experimental setup

In this section, we explain how the experiments were performed and how do we visualise and compare the results.

Conditions

All the tests are performed in the dynamic setting – the algorithm does not know the stream length in advance. The reason for this decision is that we expect this to be often the case in practice and also because we wanted to do a fair comparison with ReqSketch which has this property. On the other hand, we do not test the mergeability at all, as we have no theoretical analysis and it is not obvious how exactly the merge algorithm from ReqSketch should be used on Jagged Sketch.

As our algorithm is comparison-based, its output does not depend on the actual sizes of items, only on their permutation. Therefore, we used a permutation on integers $\{1, \ldots, N\}$ as the input stream. This simplifies the testing phase, as the rank of an item equals its value. The construction of adversarial streams is a nontrivial problem, thus we decided to simply test the sketches on a random permutation, which we expect to resemble real-world data accurately enough for our purposes.

The tests

For fixed parameters ε, J, Q and δ of the sketch and stream length N, for each item y in the input stream our objective is to determine an error bound $\mathrm{b}(y)$ such that $|\mathrm{Err}(y)| \leq \mathrm{b}(y)$ with probability $1 - \delta$. Thus, we ran the sketch m times

on the same input stream (for a sufficiently large m), measure $|\mathrm{Err}(y)|$ for each instance, and take the $1-\delta$ quantile of the m measurements. Note that all the m sketches have exactly the same size as the size is independent of the random bits used in the algorithm. We set $N = 10^9$, $m = 10^3$ and $\delta = 0.01$ for all the tests.

It would be really impractical to do all of this for all items of the input stream. First, most of the answers would be the same (as the sketch can produce only `SPACE` different values) and second we would need to store a large amount of data. Hence, we choose a sufficient set of *sample points*, for which we determine the error bound. It is tempting to simply choose the items present in the sketch as the sample points. However, the sample points must be the same for all the m measurements and the items present in the m sketches are different (as they depend on the random bits of the algorithm). Hence, we sort all the items from all the sketches and take every m-th of them as a sample point. By this, we ensure that the number of sample points equals the number of items in the sketch (which is the number of possible different answers the sketch can give) and moreover, the distribution of the sample points is similar to the distribution of the possible answers of the sketch.

Thus, with given ε, J, Q, $N = 10^9$, $\delta = 0.01$ and $m = 10^3$ we fix a random permutation on $\{1, \ldots, N\}$ as an input stream, run the sketch m times on the input stream with parameters J, Q and δ, determine the `SPACE` sample points and for each sample point s we get the bound $\mathrm{b}(s)$ as the 99-th percentile of the m values of $|\mathrm{Err}(s)|$.

Visualisation

For visual comparison, we plot the results of the experiments. We are mostly interested in the relative error, thus for each sample point s we put $s = \mathrm{R}(s)$ on the x-axis and $\mathrm{b}(s)/s$ on the y-axis. For additive error (which we use only for comparison to KLL) we simply put $\mathrm{b}(x)/N$ on the y-axis.

If we plot the data as they are, the graphs are quite unreadable as there are too many data points. We solve this problem by taking the maximum of each 50 consecutive points and plotting only these maxima.

All the graphs we present are logarithmic in the x-axis. This is because the motivation for using relative error is an interest in small ranks, which would not be visible on a linear scale.

Comparison of distinct tests

The asymptotic size of the sketch depends on ε, δ and N. However, the size in practice also depends on the other parameters and the performance of the sketch naturally depends on the size. Hence, to compare two different sketches, we need them to be of the same size. Thus, we choose one fixed value V for all the experiments, and before each experiment we binary search for an epsilon such that the sum of capacities of all the compactors is approximately V. Note that we cannot require the sum of capacities to be exactly V, as not all values of the sum are possible.

We must note here that we used a different random stream for each of the tests. To make sure this does not affect the results, we tried the same experiment

repeatedly with different random streams and we verified that the results are similar enough.

ReqSketch and KLL

For comparison with ReqSketch, we used the original proof-of-concept implementation[1] by Cormode et al. [Cor+23]. The implementation has an even integer parameter k instead of ε, hence the ability to set a particular size of the sketch is even more restricted than for Jagged Sketch. Hence, we run ReqSketch with the parameter $k = 50$ set as recommended by the authors, and then we set the value V as the sum of the capacities of all the compactors. Thus, we have $V = 15180$ for all the experiments.

We also did a small change in the implementation. The authors made the implementation lazy (we explain the laziness in Section 3.2), but they later restricted the laziness to decrease update time [Cor+21]. As we did not care about the time at all, we restored the full laziness in ReqSketch to make the comparison more fair.

For comparison with KLL, we used the original proof-of-concept implementation[2] by Karnin et al. [KLL16]. We stress that the comparison is only illustrative as the authors explicitly state they implemented just the most simple version with no optimizations.

3.2 Implementation

In the file `jaggedSketchSimple.py`[3] we attach a straightforward implementation of the dynamic version of Jagged Sketch, which is very close to the description in Sections 2.2 and 2.4.1 and is most suitable for reading. However, for the experiments, we used an improved version `jaggedSketchImproved.py`[4]. In this section, we discuss its differences from the original description in Sections 2.2 and 2.4.1. We tested all the improvements first by adding them to the simple implementation separately and checking the change of the error. The Figure 3.1 shows the difference between our two implementations for $Q = \{0\}$ and $J = 0.5$ (this is the setting we suggest as a default for practice).

Important levels

The important level $\mathcal{L}(r)$ is intentionally defined such that if $\mathrm{R}(y) = r$, we have $H(y) = \mathcal{L}(r) \pm 1$, and the critical level $H(y)$ is in turn defined such that it is the first level h where $\mathrm{R}(y, O_h) = 0$ with high probability. In the dynamic case, this holds only with some unknown additive constant. Thus, instead of relying on the asymptotics, with each recalculation we can set the important levels explicitly by their meaning as follows:

For each important rank $r \in R$ we simply ask the sketch for an item y of rank approximately r and then we binary search for the highest level ℓ such that the

[1] `https://github.com/edoliberty/streaming-quantiles/blob/master/relativeErrorSketch.py` 7d4e6ea
[2] `https://github.com/edoliberty/streaming-quantiles/blob/master/kll.py` 73f927e
[3] `https://github.com/domestomas/JaggedSketch/blob/main/jaggedSketchSimple.py`
[4] `https://github.com/domestomas/JaggedSketch/blob/main/jaggedSketchImproved.py`

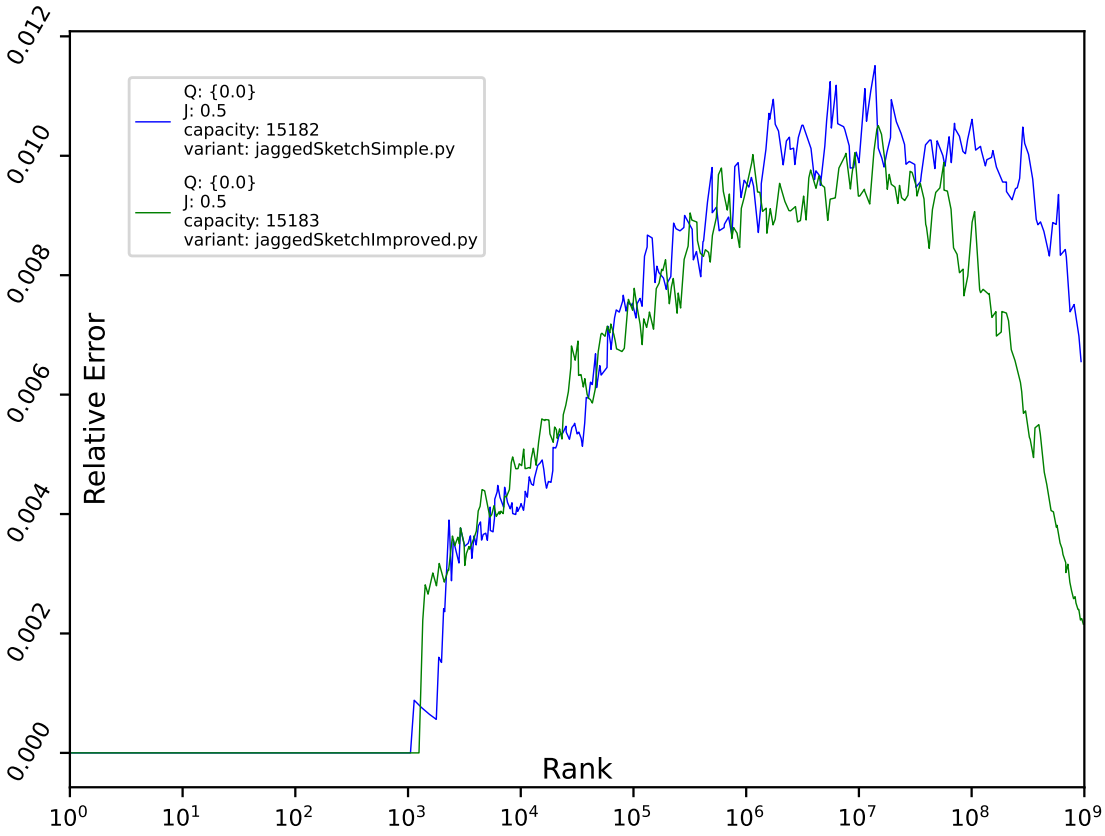

Figure 3.1 Comparison of `jaggedSketchSimple.py` and `jaggedSketchImproved.py` for $Q = \{0\}$ and $J = 0.5$

minimal item present in the level is smaller than y. We have $\mathrm{R}(y, O_\ell) = 0$ as no item from O_ℓ can be smaller than the minimum item from compactor $\ell + 1$.

Note that we cannot do this in the static version, as we need to set the important levels before reading the stream.

The power of P

The parameters C and K are originally defined such that immediately after a schedule reset we have $K = C/(4 \log P)$. This is important, as the number of sections of the right half of the buffer is $S = C/(2K) = 2 \log P$. The number of compactions to the next reset is then $2^{S-1} = 2^{2 \log P - 1} = P^2/2$. We determined by trial and error on the simple implementation that Jagged Sketch performs better if we have $P^{1.5}$ instead of P^2. Thus, we set $K = C/(3 \log P)$. The analysis from Chapter 2 stays valid as long as the power is larger than 1.

The laziness

Ivkin et al. [Ivk+22] proposed several improvements to the KLL sketch that are applicable also to ReqSketch and Jagged Sketch. Cormode et al. [Cor+23] used some of them in their implementation and we follow their example. For a detailed explanation of the improvements, see [Ivk+22].

The first of the applied improvements is laziness. In the lazy version of the sketch, we let the compactors to be overfilled as long as the size of the whole sketch does not exceed the sum of the capacities of all the compactors. This

means that the compaction is not triggered by the first compactor being overfilled but by the whole sketch being overfilled.

We can distinguish partial and full laziness. With full laziness, we stop performing the compactions as soon as the whole sketch is not overfilled, meaning that more compactors can stay overfilled over time. With partial laziness, if we compact the first compactor, we perform all other compactions as in the original version of the algorithm, thus only the first compactor can stay overfilled over time.

We mentioned above that the authors of ReqSketch switched full laziness for partial laziness to improve amortized update time. We used full laziness for both ReqSketch and Jagged Sketch as we do not care about time complexity for now. We verified that the simple implementation performs better with full laziness.

The randomness reduction

Another improvement by Ivkin et al. [Ivk+22], also used in the implementation of ReqSketch, is reducing randomness via anticorrelation. For each compactor, we remember the random bit used for choosing odd/even indexed items in the last compaction. If the number of compactions P on the given compactor is odd, we generate a new random bit and use it. If P is even, we just use the negation of the bit we used the last time. This strategy yields a decrease of the error by a constant factor in the analysis (and it is also intuitively a good idea). Again we verified its practical utility by adding this modification to the simple implementation.

The random shift

A similar improvement, again by Ivkin et al. [Ivk+22] consists of shifting the compacted part of the buffer to the left by 1 item independently with probability $\frac{1}{2}$ during each compaction. If this shift happens, the largest item stays in the buffer (and the number of compacted items stays the same). Before this improvement, for an item y which is smaller than the maximum and larger than the minimum of the compacted items, the compaction could be always important (if we are unlucky). Now, the compaction is important with probability $\frac{1}{2}$ as the shift changes the parity of the rank of y among the compacted items.

We note that we can use anticorrelation also for this improvement – in odd compactions we set the new random bit for the parity of the deleted items and in even compactions we set the new random bit for the random shift, thus the bits are still independent as one of them is always chosen at random. By our measurements, the utility of this improvement is very small, but we added it nonetheless as it requires only a small change.

Removing the improvement for high ranks

We explained in Section 1.3 that our error improvement for high ranks is based on the fact that compared to the implementation of ReqSketch we increase the size of the high compactors which does not change the asymptotic size of the sketch but it decreases asymptotically the error for high ranks. However, the improvement definitely increases the size of the sketch somehow. Hence, for a fixed size of the sketch, the error improvement for high ranks costs us an increase of the error for

lower ranks, which can be undesired. Thus, we added the possibility to switch the improvement off. In this case, C becomes $\sqrt{\delta^{-1}\log(P)}H^{\min(1,J)}/(\varepsilon F)$ instead of $\sqrt{\delta^{-1}H}H^{\min(1,J)}/(\varepsilon F)$ and the number of sections stays the same (thus K equals $C/(3\log P)$ in both cases).

Our experiments suggest that removing the improvement decreases the error for lower ranks only by a small amount and thus we do not recommend this feature for real-world applications (see Section 3.3.3).

The fixed-space version

The last improvement that we considered is the fixed-space version of the sketch. We came up with the idea ourselves, but we do not expect it to be novel. The main point of the modification is setting a space bound instead of ε. Any time we add a new compactor or increase a capacity of existing one, the ε is increased in such a way that the sum of capacities of all the compactors does not exceed the space bound. This can be more practical for the users of the algorithm if they have a memory limit to fit in (e.q. the memory of their machine) and it can also increase the performance of the sketch as we are using the whole space from the beginning.

This modification introduces two problems. The first problem is that the capacity of all the compactors can change whenever the capacity of any compactor increases. However, we can observe that in this situation all the capacities decrease. If we only scale the parameters C and K by the new ε and do not take into account the changes in P (so that the number of sections does not change), it is not hard to show that the analysis of the compaction schedule stays valid.

The second problem is that for large enough N we can get ε larger than 1. This can be elegantly solved by setting both space bound and ε_0. While ε is smaller than ε_0, the sketch fills the given space and when we would violate the bound given by ε_0, we fix $\varepsilon = \varepsilon_0$ and the sketch starts growing beyond the space bound.

We tested this improvement with the simple version and it yields better error. However, if we applied it together with the other changes explained above, the change was no longer significant. Hence, we decided not to use it in our implementation, as it makes the code more complicated and it does not give us smaller error. We would definitely recommend using this modification for real-world applications of the sketch.

3.3 The results

Finally, let us present the results of the experiments. First, we show the different settings of parameters Q and J, then we compare Jagged Sketch to ReqSketch and KLL, and finally, we show the effect of removing the improvement for high ranks.

3.3.1 Versatility

The advantage of Jagged Sketch is that it enables the users to express their priorities by the setting of parameters Q and J. Let us recall that Q says which

ranks are important and J gives their importance relative to other ranks – the larger J the more important the ranks given by Q.

Different settings of J for $Q = \{0\}$

We expect the most common setting of Q to be $Q = \{0\}$ in practice, thus we first fix $Q = \{0\}$ and test the behaviour of Jagged Sketch for different settings of J. In Figures 3.2 to 3.4 we show the error for $J \in [0, 0.5]$, $J \in [0.5, 1]$ and $J \geq 1$ respectively.

We can see in Figure 3.2 that the sketch works as expected. For items of a small rank, there is zero error, as they fit to the first compactor. With $J = 0$ the (relative) error is worst at the beginning and gets better for higher ranks, which is the result of the improvement for high ranks (we discuss turning the improvement off in Section 3.3.3). With larger J the error gets better for low ranks and worse for high ranks.

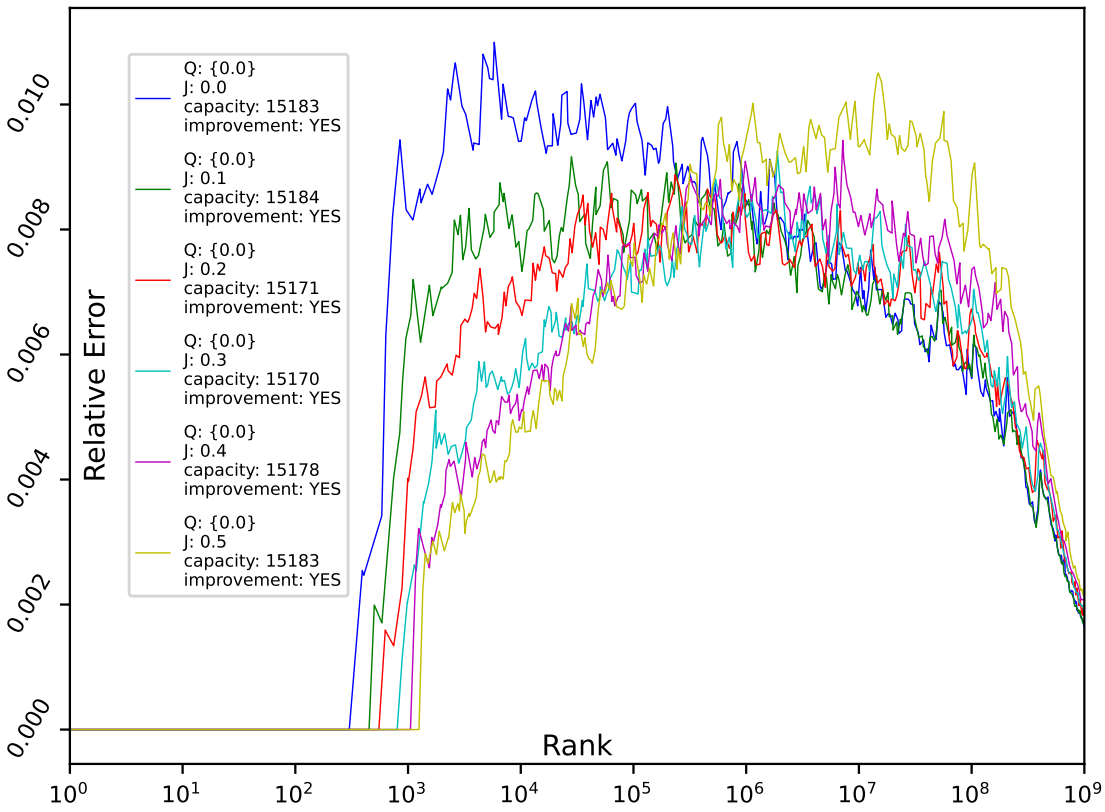

Figure 3.2 Jagged Sketch with $Q = \{0\}$ and $J \in [0, 0.5]$

In Figure 3.3 we can see the same effect, but in this range of J the improvement for small ranks gets less significant and the worsening for high ranks is bigger. From this, we conclude that the most universal setting of J is around $J \in [0.3, 0.5]$, although the optimal setting of course always depends on our priorities.

In Figure 3.4 the situation is less clear. It is still true that with growing J we get better error for small ranks. For a bit larger ranks the error gets worse with larger J. However, for even larger ranks, the error gets worse for J up to 1.4 and for larger J it gets better again. To explain this phenomenon, let us recall that $F = \overline{\min(H, d^J)}$. The larger J, the sooner we reach the situation where $H < d^J$ as we go up the levels. From this point upwards the sketch behaves similarly as

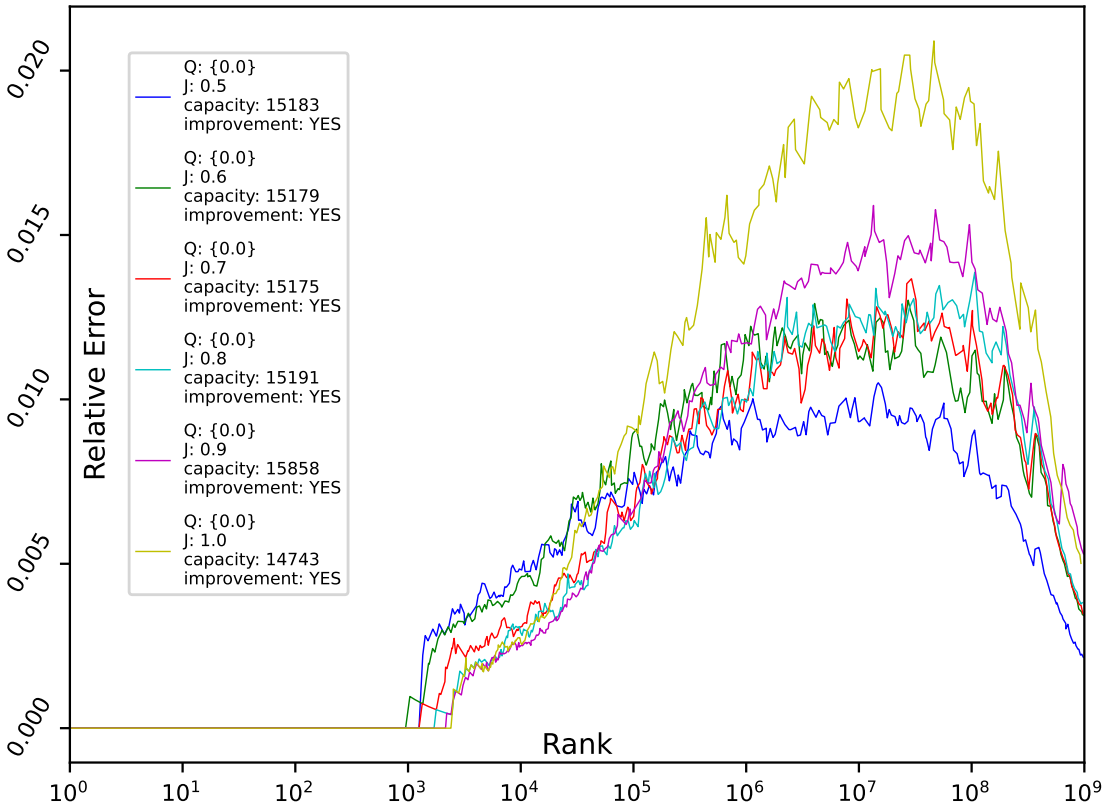

Figure 3.3 Jagged Sketch with $Q = \{0\}$ and $J \in [0.5, 1]$

with $J = 0$. Recall that this was exactly the intuition behind the definition, as we explained in Section 1.4. This can be clearly seen in the graph, as with increasing J the boundary between the "jagged" and "unjagged" behaviour moves to the left. With large enough J, the sketch looks like with $J = 0$ with the exception of level 0 which is H times larger. This corresponds to the fact that there is only a small difference between $J = 5$ and $J = 20$ in the graph and we observed no changes for $J > 20$.

Different settings of Q for $J = 0.5$

Let us now fix $J = 0.5$ (as it seems to be a reasonable choice from the data shown above). The different settings of parameter Q are shown in Figure 3.5. We can see that the sketch works in principle as expected, but the points with the smallest error are shifted a bit to the right (see the arrows). We are not sure about the reason[5] for this phenomenon, but we expect this could be compensated by adding appropriate additive constant to the definition of important level.

3.3.2 Comparison to other sketches

With $Q = \{0\}$, we can beat ReqSketch for all the ranks for multiple different values of J, as shown in Figure 3.6. This corresponds to the fact that for $0 < J < 1$ and $Q = \{0\}$ Jagged Sketch has asymptotically the same error for the ranks in the middle and better error both for the low and high ranks. One would expect some

[5]The reason is not the alternative way of determination of important levels from Section 3.2. With the original algorithm, this effect is the same or stronger.

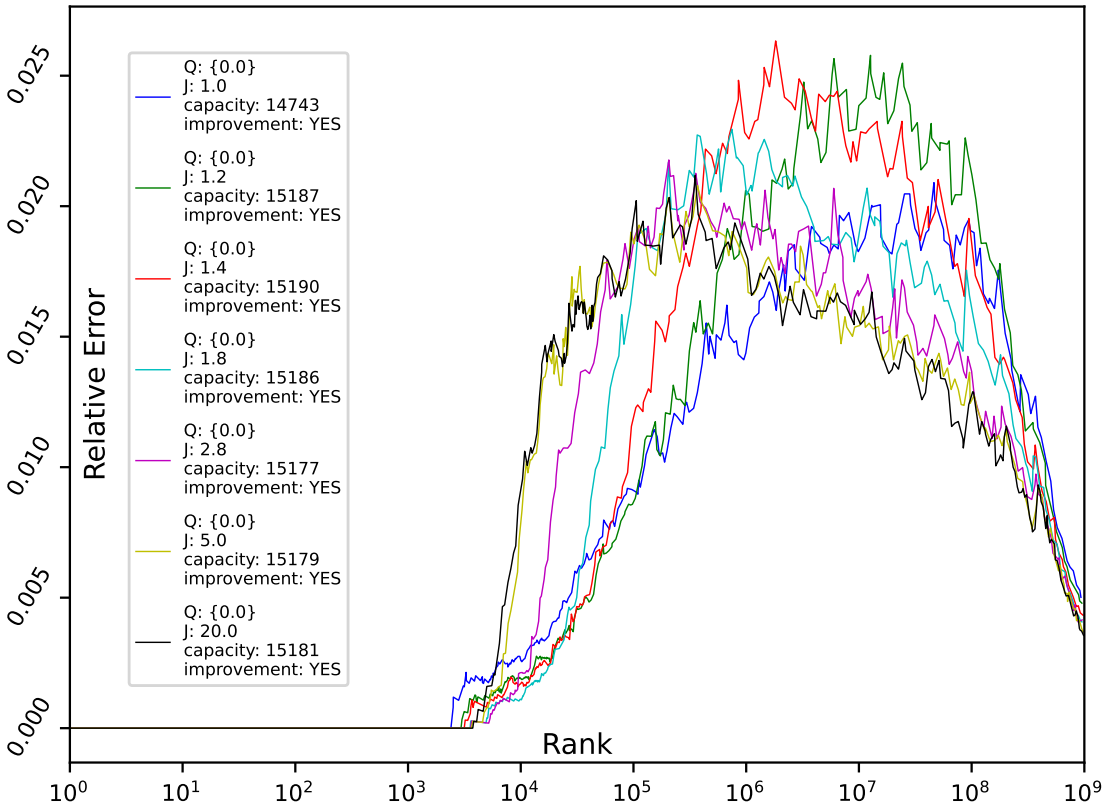

Figure 3.4 Jagged Sketch with $Q = \{0\}$ and $J \geq 1$

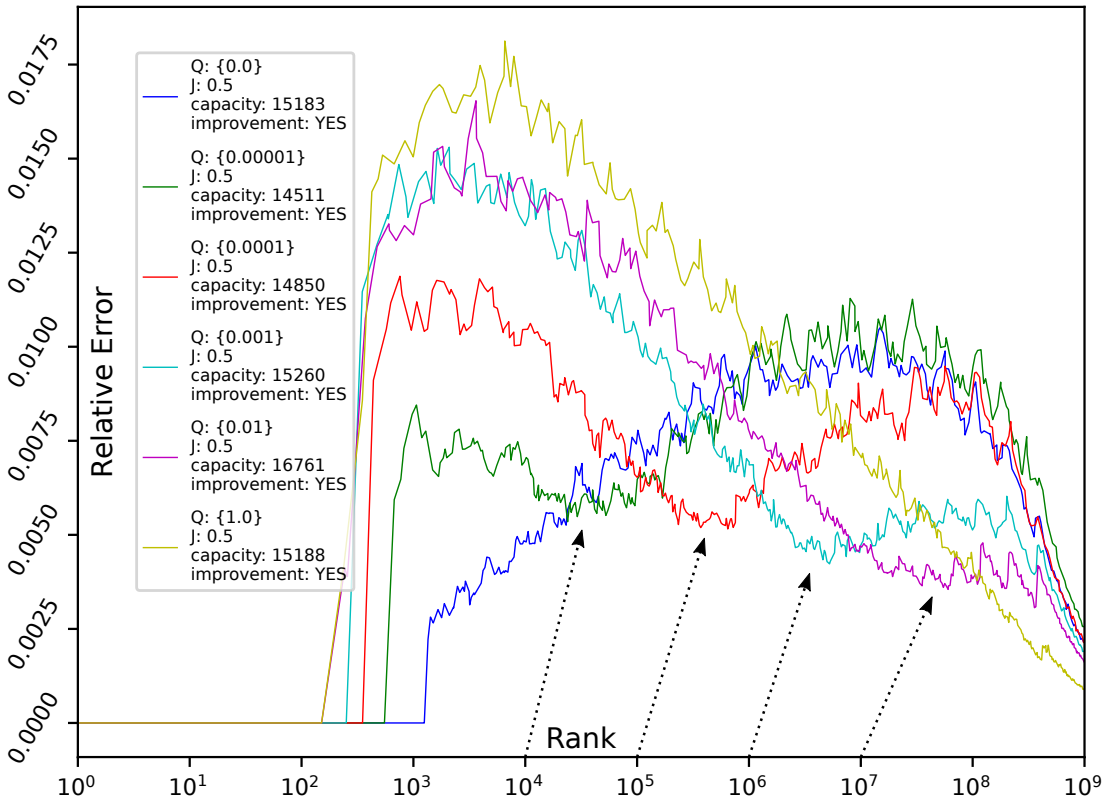

Figure 3.5 Jagged Sketch with $Q = \{q\}$ for $q \in \{0, 0.00001, 0.0001, 0.001, 0.01, 1\}$ and $J = 0.5$

worsening constant factor for the ranks in the middle, however, it is apparently less significant than the effect of changing the power of P from 2 to 1.5 explained in Section 3.2, which is the only important implementation improvement not present in the ReqSketch implementation.

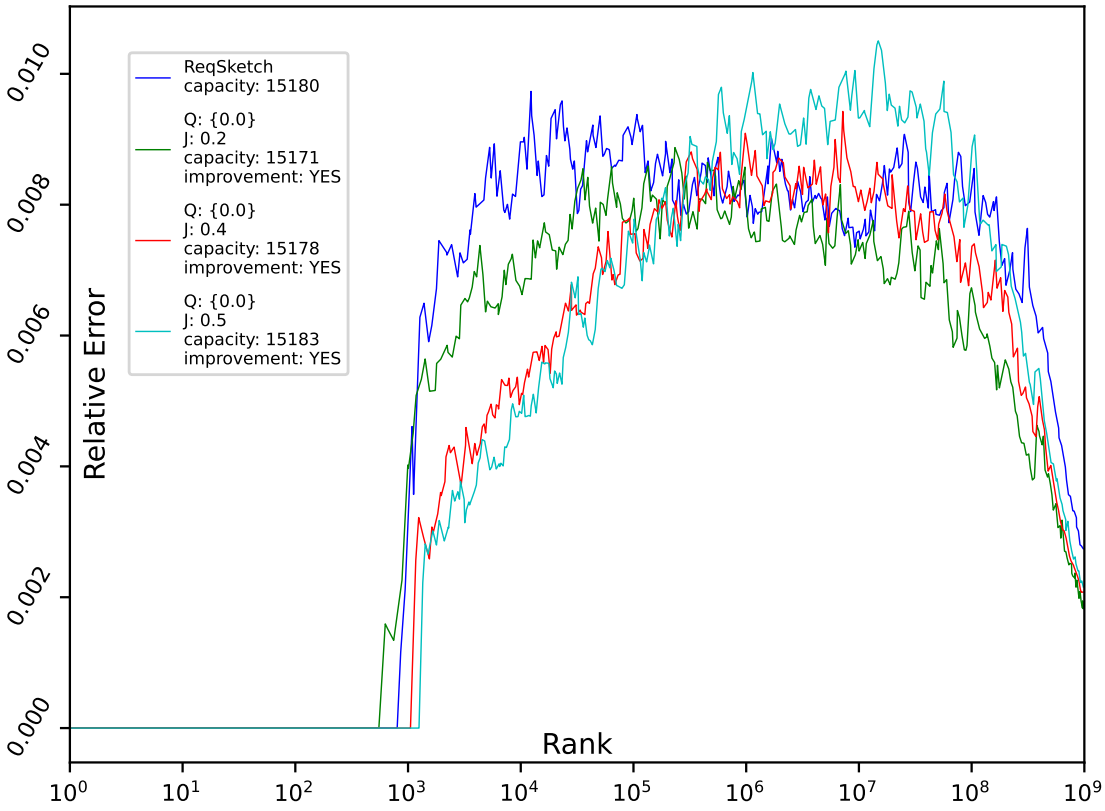

Figure 3.6 Jagged Sketch with $Q = \{0\}$ compared to ReqSketch

To compare Jagged Sketch with KLL, we plot the additive error (for sample point s we have $b(s)/N$ instead of $b(s)/s$ on the y axis). The result is shown in Figure 3.7. We have chosen $Q = \{1\}$ and $J = 20$ for Jagged Sketch, as the relative error of KLL is naturally best for the highest ranks.[6]

As we explained above, this comparison is only illustrative as there exist better versions of KLL with smaller error. For this simple KLL implementation, the additive error of Jagged Sketch is about four times larger for most ranks (note that the x-axis is logarithmic), but for the low ranks the error is naturally much smaller. This corresponds to the fact that in this setting we have asymptotically the same error for large ranks and a much smaller error for low ranks compared to KLL.

[6]Note that the curve for KLL looks a bit weird as our method of choosing sample points is not suitable for KLL – the low-ranked items are often not present in the KLL sketch and so there are no sample points in the left part of the logarithmic graph. We add some ad hoc sample points which is the reason for the strange shape of the curve. We did not try to invent any better method to get a nicer curve as the specific value of the error of KLL is of no importance to us.

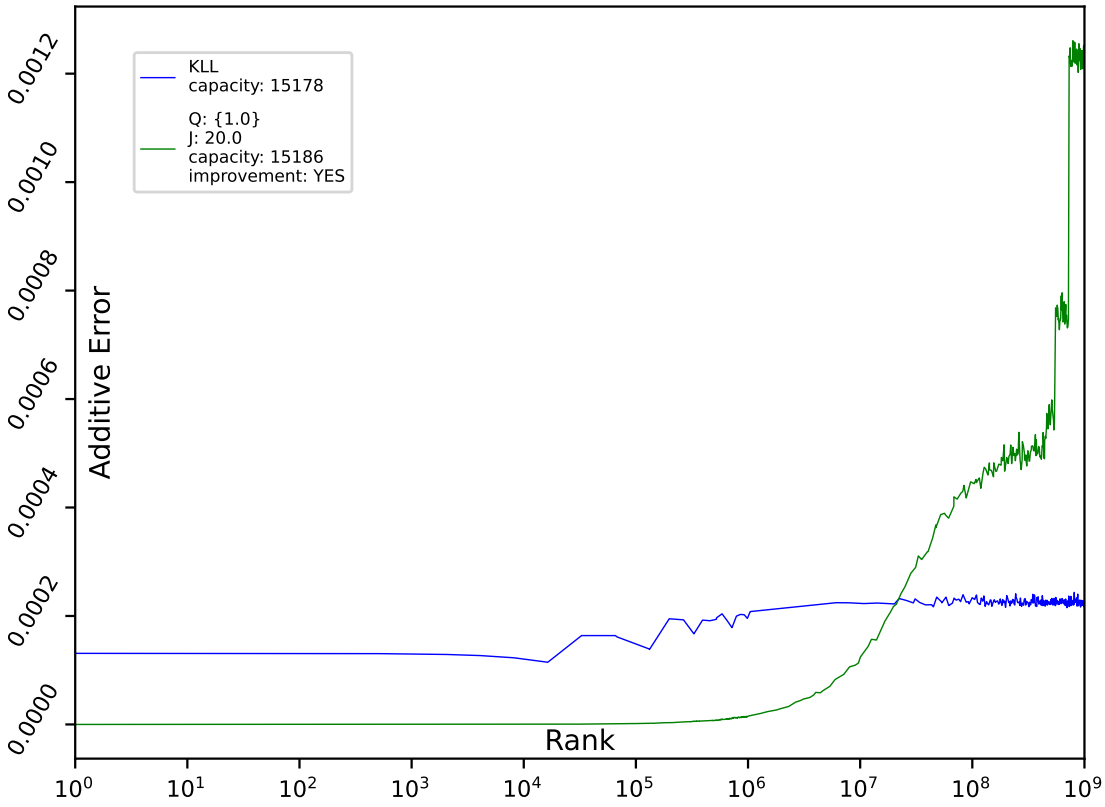

Figure 3.7 Jagged Sketch with $Q = \{1\}$ and $J = 20$ compared to KLL

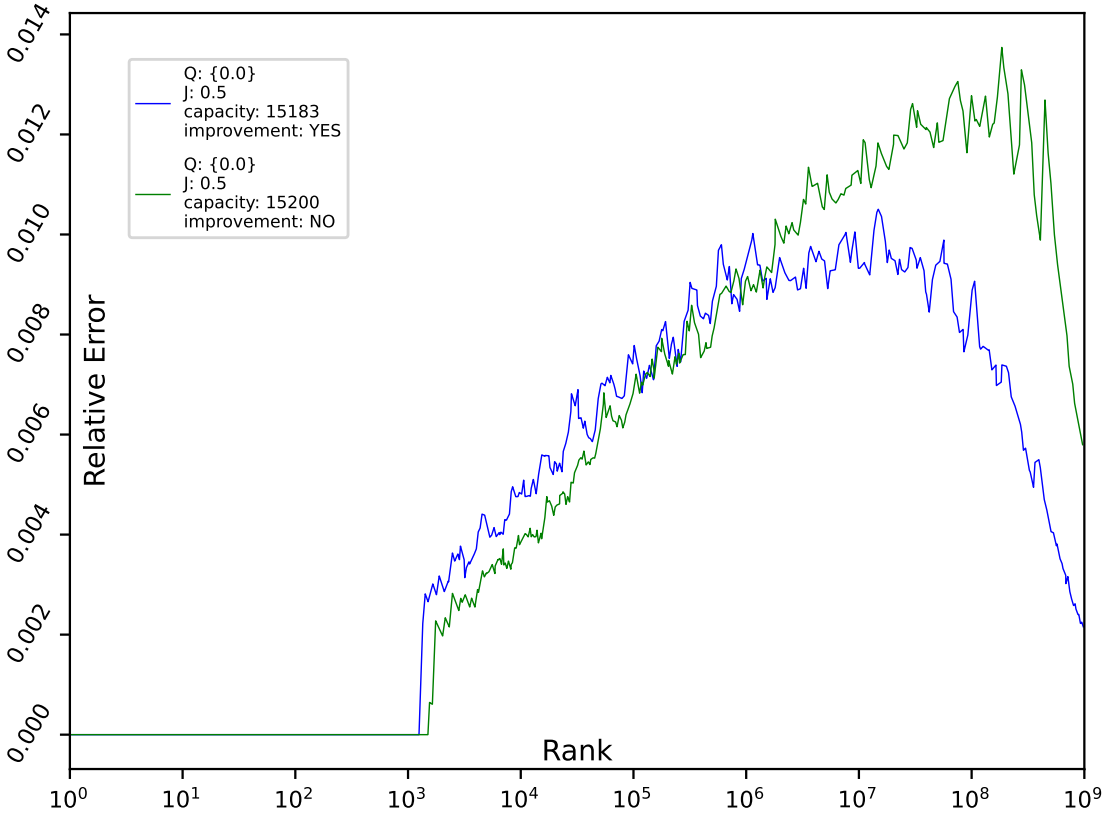

Figure 3.8 Comparison of Jagged Sketch with and without the improvement for high ranks; $Q = \{0\}$ and $J = 0.5$

3.3.3 Removing the improvement for high ranks

Finally, let us discuss removing the improvement for high ranks. Our motivation for the removal was getting better error for lower ranks. However, Figure 3.8 (with $Q = \{0\}$ and $J = 0.5$) shows that the difference in the error for low ranks is negligible compared to the difference for high ranks (again note that the x-axis is logarithmic so the difference is even larger than it looks like on the first sight). This is good news – it means that the improvement for high ranks works fine.

A skeptic could object that the improvement for high ranks can be in practice simulated by adding 1 to Q. In Figure 3.9 we can see that it is not the case – Jagged Sketch with $Q = \{0\}$ and the improvement (at least for $J = 0.5$) yields better error than Jagged Sketch with $Q = \{0, 1\}$ and without the improvement.

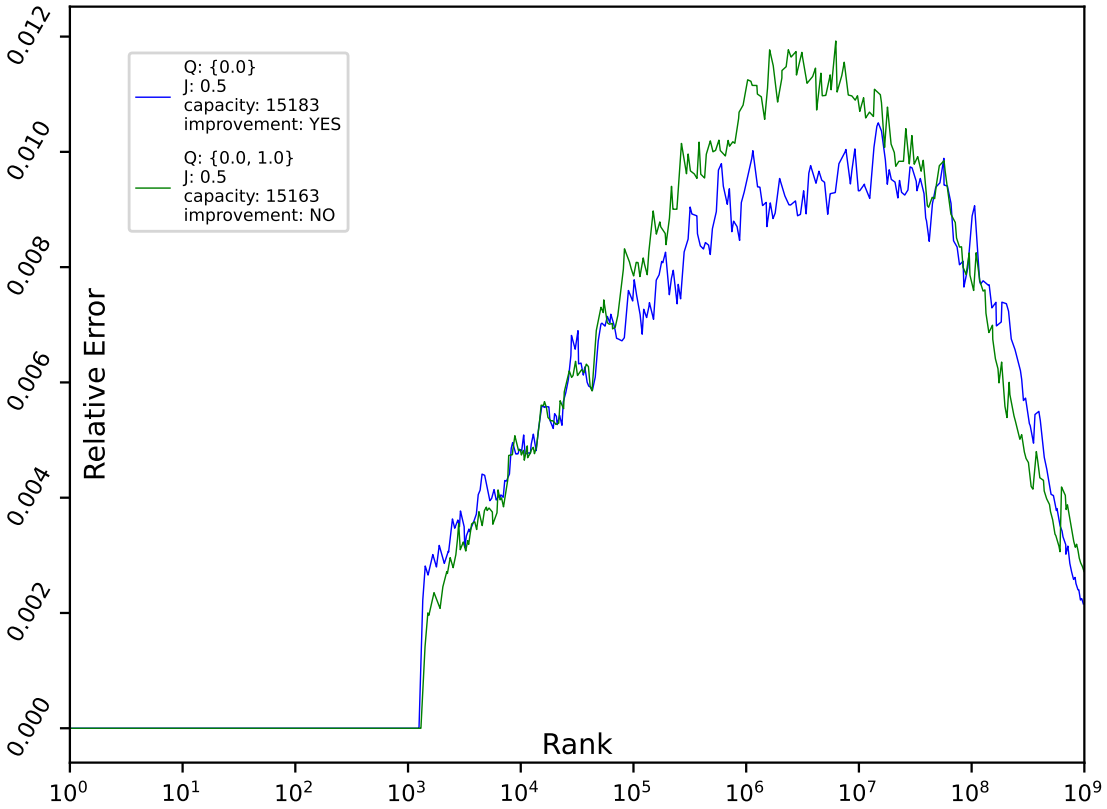

Figure 3.9 Comparison of Jagged Sketch with the improvement and $Q = \{0\}$ and Jagged Sketch without the improvement and $Q = \{0, 1\}$; $J = 0.5$ in both cases

Removing the improvement could be useful with setting $J = 0$ if we need to minimize the maximum relative error over all ranks (which is the standard relative error guarantee). However, it turns out it suffices to use the normal version of Jagged Sketch with $J = 0.4$ – see Figure 3.10. Hence we conclude that the ability to turn off the improvement is not necessary.

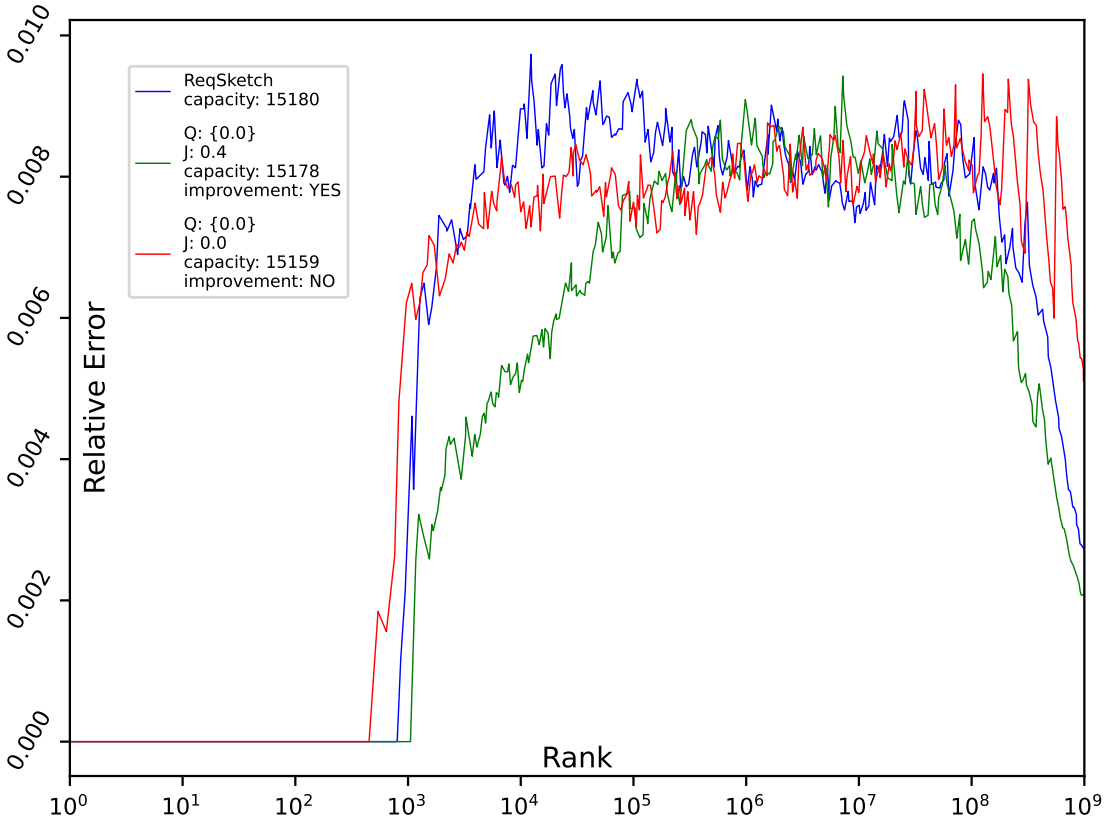

Figure 3.10 Comparison of ReqSketch and Jagged Sketch with and without the improvement for $J = 0$

Conclusion

We have developed an improved version of the ReqSketch algorithm which is more versatile and yields a better error both in theory and in practice. Unlike the original ReqSketch, our algorithm assumes the foreknowledge of N and is not mergeable.

Further research

The main unresolved task is naturally making Jagged Sketch fully mergeable, which would also remove the assumption of the foreknowledge of N. The original proof of mergeability of ReqSketch is already very complicated and it would become even more so with the added modifications. We came up with a new version of the compaction operation which we hope would simplify the analysis and we plan to use it to make Jagged Sketch fully mergeable.

Another possible direction of research is adding other enhancements developed for KLL, most importantly the ability to handle weighted input stream by Ivkin et al. [Ivk+22] and interpolation techniques by Schiefer et al. [Sch+23].

There is also still the $\sqrt{\log N}$ gap between ReqSketch and the lower bound on relative error by Cormode et al. [Cor+23]. However, the lower bound holds even for offline non-comparison-based algorithms, thus it is easily possible that the lower bound is not tight for streaming comparison-based algorithms and ReqSketch already achieves optimal space for relative error.

Finally, if there is a gap between optimal comparison-based and non-comparison-based sketches for relative error, it would be interesting to develop a non-comparison-based version of Jagged Sketch, possibly by using some of the techniques from q-digest [Shr+04], from the relative error version of q-digest by Cormode et al. [Cor+06] or from the new paper by Gupta et al. [GSW24] (preprint).

List of Notation

Notation	Meaning
item	arbitrary item u from a fixed universe $\mathcal{U}$ with a total order
input stream $\mathcal{S}$	a sequence of items from a fixed universe $\mathcal{U}$ with a total order
N	the length of the input stream; $N = \lvert\mathcal{S}\rvert$
$\mathrm{R}(y, s)$	the rank of item y in a sequence s
$\mathrm{R}(y)$	the rank of item y in the input stream, equivalent to $\mathrm{R}(y, \mathcal{S})$
$\hat{\mathrm{R}}(y)$	the estimate on $\mathrm{R}(y)$ returned by the sketch on rank query y
$\mathrm{Err}(y)$	the error of the rank query y; $\mathrm{Err}(y) = \left\lvert \mathrm{R}(y) - \hat{\mathrm{R}}(y) \right\rvert$
$\log(x)$	the binary logarithm of x; $\log_2(x)$
$\ln(x)$	the natural logarithm of x; $\log_e(x)$
$\overline{x}$	$\max(x, 1)$
$\overline{\log}(x)$	$\max(\log x, 1)$
dynamic version	the version of Jagged Sketch without the foreknowledge of the stream length N
static version	the version of Jagged Sketch with the foreknowledge of the stream length N
time t	state of the sketch after performing t update operations
X^t	the state of any variable X at time t; for example $N^t = t$
R	the set of important ranks given by users; we have $\lvert R\rvert \in \mathcal{O}(1)$ and $R \subset \{1, \ldots, N\}$
Q	the set of important ranks given by users for the dynamic version, at any time t we have $R^t = \{\overline{\lceil N^t q \rceil} \mid q \in Q\}$
δ	the probability of exceeding the error bound
ε	the parameter that controls the size and accuracy of the sketch
jaggedness J	the parameter $J \geq 0$ expresses the priority of important ranks in R relative to other ranks; $J = 0$ means all ranks have the same priority (rendering R irrelevant)
H	the height of the sketch; the number of compactors
level	the index of a given compactor in the sketch; the first compactor is at level 0
$\mathcal{L}(r)$	important level for given important rank $r \in R$
$H(y)$	critical level; the first level h such that $\mathrm{R}(y) \leq 2^{h-2} C_h$
`SPACE`	the space occupied by the whole sketch

Notation	Meaning
protected items	the items that stay in the buffer and do not participate in the compaction
y-important item	item i is important w.r.t. item y if $\mathrm{R}(i) \leq \mathrm{R}(y)$
y-important compaction	compaction is important w.r.t. item y if the rank of y among the compacted items is odd
m_ℓ	the number of important compactions on level ℓ with respect to some item y
special compaction	a compaction that happens immediately before the schedule reset
recalculations	a situation in the dynamic version of Jagged Sketch when the sketch grows and all the parameters are recalculated
full compaction	a compaction in the dynamic version of Jagged Sketch used before recalculation; the whole right half of the buffer is compacted
C_h	the capacity of the compactor at level h
I_h	the input stream of the compactor at level h
O_h	the output stream of the compactor at level h
B_h	the buffer of the compactor at level h
F_h	scaling factor of the compactor at level h, depending on the closest important level
P_h	the number of compactions already performed on the compactor at level h
L_h	the compaction schedule of the compactor on level h
S_h	the number of sections of the right part of the buffer of the compactor on level h

Bibliography

[GSW24] Gupta, Meghal; Singhal, Mihir; Wu, Hongxun. *Optimal quantile estimation: beyond the comparison model* (preprint). 2024. Available from arXiv: 2404.03847.

[Cor+23] Cormode, Graham; Karnin, Zohar; Liberty, Edo; Thaler, Justin; Veselý, Pavel. *Relative Error Streaming Quantiles*. Journal of the ACM. 2023. Available from arXiv: 2004.01668.

[Sch+23] Schiefer, Nicholas; Chen, Justin Y.; Indyk, Piotr; Narayanan, Shyam; Silwal, Sandeep; Wagner, Tal. *Learned Interpolation for Better Streaming Quantile Approximation with Worst-Case Guarantees*. SIAM Conference on Applied and Computational Discrete Algorithms (ACDA23). 2023. Available from arXiv: 2304.07652.

[Ivk+22] Ivkin, Nikita; Liberty, Edo; Lang, Kevin; Karnin, Zohar; Braverman, Vladimir. *Streaming Quantiles Algorithms with Small Space and Update Time*. Sensors. 2022. Available from arXiv: 1907.00236.

[Cor+21] Cormode, Graham; Mishra, Abhinav; Ross, Joseph; Veselý, Pavel. *Theory meets Practice at the Median: a worst case comparison of relative error quantile algorithms*. KDD '21: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. 2021. Available from arXiv: 2102.09299.

[Dun21] Dunning, Ted. *The t-digest: Efficient estimates of distributions*. Software Impacts. 2021. Available from doi: 10.1016/j.simpa.2020.100049.

[CV20] Cormode, Graham; Veselý, Pavel. *Tight Lower Bound for Comparison-Based Quantile Summaries*. Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. 2020. Available from arXiv: 1905.03838.

[KLL16] Karnin, Zohar; Lang, Kevin; Liberty, Edo. *Optimal quantile approximation in streams*. 2016 IEEE 57th annual symposium on foundations of computer science (FOCS). 2016. Available from arXiv: 1603.05346.

[Rig15] Rigollet, Philippe. *High dimensional statistics: Lecture notes*. 2015. Available also from: https://ocw.mit.edu/courses/18-s997-high-dimensional-statistics-spring-2015/pages/lecture-notes/.

[ZW07] Zhang, Qi; Wang, Wei. *An efficient algorithm for approximate biased quantile computation in data streams*. Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management. 2007. Available from doi: 10.1145/1321440.1321601.

[Cor+06] Cormode, Graham; Korn, Flip; Muthukrishnan, S.; Srivastava, Divesh. *Space- and time-efficient deterministic algorithms for biased quantiles over data streams*. Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. 2006. Available from doi: 10.1145/1142351.1142389.

[Shr+04] Shrivastava, Nisheeth; Buragohain, Chiranjeeb; Agrawal, Divyakant; Suri, Subhash. *Medians and beyond: new aggregation techniques for sensor networks*. Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems. 2004. Available from doi: 10.1145/1031495.1031524.

[GK01] Greenwald, Michael; Khanna, Sanjeev. *Space-efficient online computation of quantile summaries*. SIGMOD Rec. 2001. Available from doi: 10.1145/376284.375670.

[MRL99] Manku, Gurmeet Singh; Rajagopalan, Sridhar; Lindsay, Bruce G. *Random sampling techniques for space efficient online computation of order statistics of large datasets*. SIGMOD Rec. 1999. Available from doi: 10.1145/304181.304204.

[Vit85] Vitter, Jeffrey S. *Random sampling with a reservoir*. ACM Trans. Math. Softw. 1985. Available from doi: 10.1145/3147.3165.