arXiv:2304.07652v1 [cs.DS] 15 Apr 2023

通过学习插值实现更优的流式分位数近似及最坏情况保证

Nicholas Schiefer* JustinY. Chen† Piotr Indyk† Shyam Narayanan†
Sandeep Silwal† Tal Wagner‡

摘要

对于包含 n 个输入的数据流， ε 近似分位数草图可以近似任意查询点 q 的秩——即小于 q 的输入点数量——其加性误差不超过 εn；通常，该保证以至少 $1-1/\text{poly}(n)$ 的概率成立，同时占用 $o(n)$ 空间。尽管 Karnin、Lang 和 Liberty 提出的著名 KLL 草图针对最坏情况数据流实现了可证明最优的分位数近似算法，但它在实践中的近似效果往往远非最优。事实上，实践中最常用的技术是 Dunning 的 t-digest，它在真实世界数据上的近似效果往往远胜 KLL，但已知在最坏情况下可能产生任意大的误差。我们将插值技术应用于流式分位数问题，力求在保持类似最坏情况保证的同时，在真实世界数据集上取得优于 KLL 的近似效果。

1 引言

分位数近似问题是流式计算模型中最基本的问题之一，也是实践中最重要的流式问题之一。给定一组项目 x_1, x_2、.....、 x_n以及一个查询点 q， q 的秩记为 $R(q)$，即满足 $x_i \leq q$的 $\{x_i\}_{i=1}^n$ 中的项目数量。 ε-近似分位数草图是一种数据结构：通过访问数据流元素的一次遍历，它能够同时近似所有查询点的秩，且加性误差至多为 εn。

鉴于其核心重要性，数据流分位数问题一直受到理论研究者和实践工作者的广泛研究。Manku、Rajagopalan 和 Lindsay [10] 给出了一种随机化解法，所需空间为 $O((1/\varepsilon)\log^2(n\varepsilon))$；他们的技术还可以直接改造为一种确定性解法，并达到相同的界 [14]。随后，Greenwald 和 Khanna [4]开发了一种确定性算法，所需空间仅为 $O((1/\varepsilon)\log(n\varepsilon))$。更近期，Karnin、Lang 和 Liberty (KLL) [7]开发了随机化 KLL 草图，该草图以 $1-\delta$ 的概率在所有点上都能成功，并使用 $O((1/\varepsilon)\log\log(1/\delta))$ 的空间，同时给出了匹配的下界。

与此同时，数据库、计算机系统和数据科学领域的从业者也对流式分位数估计抱有浓厚兴趣，并对这一问题进行了研究。最值得一提的是，Dunning [3] 提出了广受赞誉的 t-digest，这是一种基于一维 k 均值聚类的启发式分位数估计技术，已被众多系统采用，包括 Influx 、 Apache Arrow 和 Apache Spark。尽管 t-digest 在许多真实世界数据集上都具有出色的准确性，但已知它在最坏情况下可能产生任意大的误差 [2]。

为了说明这一核心权衡，图 1 展示了 SOSD基准测试中图书数据集的秩函数 [8, 11], , 以及 KLL 和 t-digest 的近似结果；在数据集经过随机打乱时，这些近似方法占用相同的空间；此外，图中还展示了同一数据集按对抗性顺序输入数据流时的结果。我们发现，这种顺序会导致 t-digest 的性能特别差。

机器学习的最新进展推动了学习增强算法 的发展。这类算法旨在利用输入分布的经验特性，改进经典算法问题的解法 [12]。通常，学习增强算法保留与经典算法相近的最坏情况保证，同时在实际应用中常见的良好结构输入上表现更佳。我们或许可以将类似的技术用于分位数估计。

事实上，该领域的一项开创性成果研究了学习型索引结构这一相关问题。索引是一种数据结构，可将查询点映射到其秩。针对这一学习问题，人们尝试了多种模型系列，包括神经网络，以及成功的递归模型索引（ RMI）；它们能够定义分段线性近似 [9]。

尽管学习型索引旨在回答秩

*Anthropic。在 MIT 工作期间完成。
†麻省理工学院
‡Amazon AWS。在微软研究院工作期间完成。

版权所有 ©2023 本文版权归作者所有

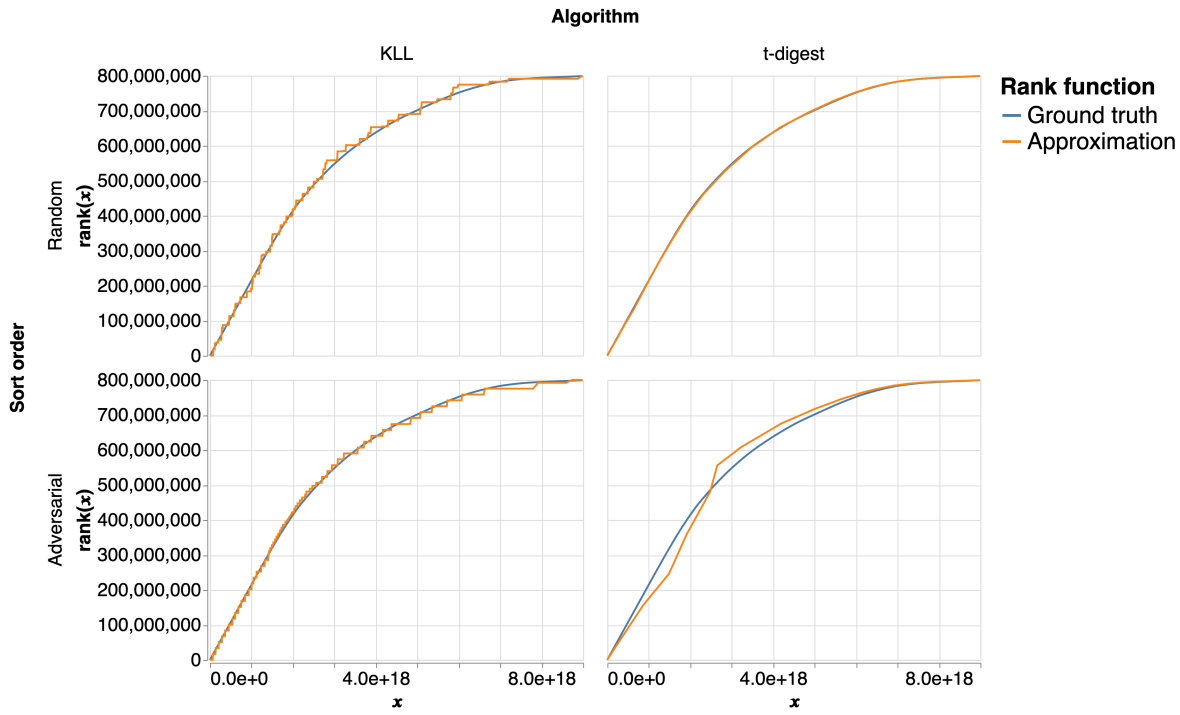

图 1：SOSD 图书数据集的真实秩函数与近似秩函数，分别使用 KLL 和 t-digest 摘要结构进行近似。对于近似处理，数据分别按随机打乱的顺序（上）和对抗性顺序（下）输入。在我们发现的对抗性情形下，t-digest 的表现远逊于 KLL，这表明了最坏情况正确性的价值。

查询，但它们并不能解决数据流分位数估计问题，因为它们并非针对数据流中的数据进行处理。例如，训练神经网络或拟合 RMI 模型需要数据流中的 $O(n)$ 个元素同时驻留在内存中，或者需要多次遍历数据流。

1.1 我们的贡献。我们提出了一种用于流式分位数问题的算法，在真实世界数据集上的误差远低于 KLL 草图，同时保留了相似的最坏情况保证。我们将该算法称为线性压缩器草图，它以线性插值 取代了 KLL 草图中的部分机制。直观而言，当真实累积分布函数相对平滑时，这种线性插值能够更好地近似该函数；许多真实世界数据集的累积分布函数通常都具有这一特性。

在理论方面，我们证明线性压缩器草图能达到与 KLL 草图相近的最坏情况误差。也就是说，线性压缩器草图以概率 $1-\delta$，使用空间 $O((1/\varepsilon)\log^2\log(1/\delta))$，计算单个元素秩的 ε 近似。这与已知下界以及与之匹配的（相当复杂的）KLL 草图版本 ε 相比，差距在 $1/\delta$ 的多重对数级因子以内。

我们的证明对原始 KLL 草图的分析做了相对直接的修改，这是因为两种算法总体上十分相似。事实上，我们可以将自己的算法视为利用了 KLL 草图分析中的一个空隙：算法设计中留有一些“余量”。

在实验中，我们展示了线性压缩器草图在 SOSD 基准测试库 [8, 11] 中的多种基准数据集上，以及在会导致 t-digest 等其他算法表现不佳的各种输入顺序下，其误差都显著低于 KLL 草图。在许多情况下，线性压缩器草图实现了与 t-digest 相当的空间-误差权衡，同时还保留了最坏情况保证。

2 理解 KLL 草图

实现最优空间复杂度的完整 KLL 草图结构较为复杂：它涉及多种不同的数据结构，包括一个用于替代顶部 $O(\log\log(1/\delta))$ 个压缩器的 Greenwald-Khanna（GK）摘要结构。这里，我们介绍一个更简单的 KLL 草图版本，其空间占用为 $O((1/\varepsilon)\log^2\log(1/\delta))$，仅比最优值高出一个 $O(\log\log(1/\delta))$ 的因子，并且通常

版权所有 ©2023 本文版权归作者所有

在实践中实现 [5], , 见 [7] 中的定理 4。在本文余下部分，我们将这一草图称为非 *GK KLL* 草图。

2.1 非 GK KLL 草图。基本的 KLL 草图由一系列压缩器组成。每个 H 压缩器都有一个容量 k，用于定义它可以存储的项目数。每个项目还关联一个（可能是隐式的）权重，表示它在草图中代表输入流中的点数。同一压缩器中的所有点具有相同的权重。

当压缩器达到其容量时，就会对其进行压缩。压缩操作首先对项目进行排序。然后，从压缩器中选取偶数位置或奇数位置的元素，并丢弃未选中的项目。丢弃偶数位置或奇数位置项目的选择概率相同。随后，将选中的项目放入层级结构中的下一个压缩器，并将这些点的权重都设为原来的两倍。这种通用结构常见于许多流式分位数草图 [10, 7]。

要预测查询点 q的秩，我们将所有压缩器中不大于 q 的所有点的权重相加。

KLL 草图的一项关键贡献是为不同的压缩器设置不同的容量。我们将数据流中的点最先到达的压缩器称为高度 0，而每个后续压缩器的高度都比其下方的压缩器高 1，因此最顶部压缩器的高度为 $H-1$。在 KLL 中，高度为 h 的压缩器的容量为 $k_h = \max(kc^{H-h}, 2)$，其中 k 是用于定义最高压缩器容量的空间参数，c 是缩放参数，通常设为 $c = 2/3$。

2.2 非 GK KLL 草图的分析。这里，我们给出非 GK KLL 草图分析的一个略作简化的版本，以便聚焦于关键细节。考虑上文所述的非 GK KLL 草图，它以 H 个不同的压缩器结束。高度为 h 的项目的权重为 $w_h = 2^h$。令 m_h 表示高度为 h 的压缩器中的压缩操作次数。

考虑高度为 h 的压缩器中的一次压缩操作，以及当时该压缩器中的一个点 x。如果 x 是压缩器中的偶数元素之一，那么其左侧的总权重（用于定义其秩）不会因压缩而改变。如果 x 是压缩器中的奇数元素之一，那么总权重要么增加 w_h（如果选择了奇数项），要么减少 w_h（如果选择了偶数项）。对于层级 h 上第 i 次压缩操作，令 $X_{i,h}$ 在选择奇数项时为 1，在选择偶数项时为 -1。注意，$\mathrm{E}[X_{i,h}] = 0$ 和 $|X_{i,h}| \le 1$。因此，层级 h 上所有压缩操作引入的总误差为 $\sum_{i=1}^{m_h} w_h X_{i,h}$。考虑数据流中的任意点 x。截至固定层级 H'，所有层级上的压缩操作在 $R(x)$ 中引入的误差因此为 $\sum_{i=0}^{H'-1}\sum_{i=1}^{m_h} w_h X_{i,h}$。

对该误差应用双尾 Hoeffding 界，我们得到

$$\begin{aligned}
&\Pr[\text{error is} > \varepsilon n] \\
&= \Pr\left[\left|\sum_{i=0}^{H'-1}\sum_{i=1}^{m_h} w_h X_{i,h}\right| > \varepsilon n\right] \\
&\le 2\exp\left(-\frac{\varepsilon^2 n^2}{2\sum_{i=0}^{H'-1}\sum_{i=1}^{m_h} w_h^2}\right).
\end{aligned}$$

这涵盖了截至 H' 的所有层所引入的误差。请注意，如果我们设定 $H' = H$，那么误差界将主要由最高层压缩器的权重项决定。为了解决这一问题，非 GK KLL 草图将最后 $s = O(\log\log(1/\delta))$ 个压缩器的容量设为固定常数 k，并对其进行单独分析：假定每次压缩产生其可能的最大误差 w_h。这是 KLL 分析中的关键引理，也是线性压缩器草图的出发点。

3 线性压缩器草图

我们提出了一种流式分位数近似算法，该算法结合了我们关于如何改进 KLL 的经验性和理论性观察。我们不改变非 GK KLL 草图的基本架构。与最优 KLL 草图类似，后者用 Greenwald-Khanna 草图替换顶部 $O(\log\log(1/\delta))$ 个压缩器；我们也用另一种数据结构替换其中一些顶部压缩器。在我们的方案中，我们将顶部 $t = O(1)$ 个压缩器替换为一种称为线性压缩器的结构。

线性压缩器。一个线性压缩器 是由元素组成的有序列表，每个元素都是数据流中的一个项目及其权重构成的对。与 KLL 一样，权重表示该项目所代表的数据流项目数量；与 KLL 不同的是，列表中各元素的权重各不相同，而且可以是任意浮点数，而非 2 的幂。与 KLL 压缩器一样，线性压缩器具有一个容量，我们将其固定为 tk，即它所替代的（固定大小）压缩器的总容量。超过该容量时，它会进行压缩 ，并且只保留一半元素。

高度为 h 的 KLL 压缩器 C_h 隐式表示——

版权所有 ©2023 本文版权归作者所有

一个分段常数函数 f：具体来说，

$$f(q) = \sum_{x \in C_h : x \le q} w_h.$$

该函数表示此压缩器对查询点 q 的近似秩所作的贡献。线性压缩器隐式表示一个分段线性函数，该函数也会对 q的秩作出贡献。给定一个线性压缩器 $L = \{(y_1, w_1), (y_2, w_2) \ldots, (y_k, w_k)\}$，其中 $y_1 \le y_2 \le \cdots \le y_k$，$L$ 对 q 的秩的贡献为

$$f_L(q) = \underbrace{\sum_{i=1}^{i^*-1} w_i}_{\text{KLL-style term}} + \underbrace{w_{i^*} \frac{q - y_{i^*-1}}{y_{i^*} - y_{i^*-1}}}_{\text{interpolation term}} \tag{3.1}$$

其中，i^* 是满足 $y_{i^*} > q$的最小索引。实际上，我们将 y_{i^*} 的权重以均匀密度分布在 y_{i^*-1} 和 y_{i^*}之间的整个区间上，而不是将其视为恰好位于y_{i^*} 处的点质量。由此得到的贡献 $f_L(q)$是一个单调的分段线性函数，正如预期。

向线性压缩器添加点。我们的线性压缩器接收来自 KLL 风格压缩器中最后一个压缩器的点，每个点的权重固定为w_{H-t-1}。不能仅仅通过拼接数组来合并这些点及其权重。为说明这一点，考虑向一个包含两个点 a 和 c 的压缩器中添加一个单位权重的点 b，其中 $a < b < c$，且 c 的权重为 w。压缩后 c的权重不应为 w，因为添加之前 c 的权重应均匀分布在整个区间 $[a, c]$上。

相反，我们通过将两列点合并为一列并排序，得到列表 $z_1 < z_2 < \cdots < z_{m+n}$，从而向现有点集 $x_1 < x_2 < \cdots < x_n$ 中添加一组新点 $y_1 < y_2 < \cdots < y_m$。接下来，我们将 $w(z_1)$ 设为原列表中 z_1 的权重，并递归地计算新的权重。假设 $z_i = x_i$，不失一般性，我们设

$$w(z_i) = w(x_i)\frac{x_i - z_{i-1}}{x_i - x_{i-1}} + w(y_*)\frac{x_i - z_{i-1}}{y_* - y_{*-1}}$$

其中，y_* 是满足 $y_i > z_i$的第一个 y_i。

等价地，我们使用公式 3.1 将每个权重函数转换为秩函数，将它们相加，然后计算有限差分以得到最终的权重函数。

压缩线性压缩器。最后，我们描述压缩线性压缩器的过程。给定参数 $\alpha \in [0, 1]$ 和一个包含 n 个点的线性压缩器 C，我们希望获得一个包含 αn个点的新线性压缩器 C'，并满足以下性质：

- 中的点是 C中点的子集。
- 两个压缩器中各点的总权重相同，因此 $\sum_{x \in C} w(x) = \sum_{x \in C'} w(x')$。
- 对于每个点 $x \in C'$，秩 $f_C(x) = f_{C'}(x)$。
- 压缩引入的“误差”尽可能小。也就是说，对于某个损失函数 L，我们希望 $\sum_{x \in C} L(f_{C'}(x)\ f_C(x))$ 尽可能小。

本文使用 $\alpha = 1/2$，不过原则上也可以使用其他值。

高效完成这一过程非常重要。在我们的实验中，主要采用上确界（ℓ_∞）损失 $L(x, x') = \sup_x |x - x'|$。可以使用 [6] 提出的一种动态规划技术将其最小化。

4 分析

我们对算法进行最坏情况分析，其结果与非 GKKLL 草图版本的最坏情况分析相匹配：

THEOREM 4.1. 第 3 节介绍的线性压缩器草图，以空间复杂度 $O((1/\varepsilon)\log^2 \log(1/\delta))$，以概率 $1 - \delta$ 对单个元素的秩进行 ε 近似。

我们采用两种技术来分析每个压缩器引入的误差。为了分析线性压缩器草图中 KLL 风格压缩器的误差，我们证明它们引入的误差与在相同数据流上运行的非 GK KLL 草图完全相同。接着，我们应用非 GK KLL 草图的两部分分析，分别分析前 $H - s$ 个压缩器，以及第 $(H - s)$ 个到第 $(H - s + t)$ 个压缩器。为了分析顶部线性压缩器的误差，我们分析每次压缩引入的误差。随后，我们分析线性压缩器的压缩次数，从而得出线性压缩器引入的总误差。

考虑数据流 $X = x_1, x_2, \ldots\ldots, x_n$。设 $S(X)$ 是在该数据流上计算得到的非 GK KLL 草图，它以 H 个压缩器结束；设 $S_b(X)$ 是 $S(x)$ 的第 b 个压缩器。同样，设 $S'(X)$ 是在该数据流上计算得到的线性压缩器草图，其中包含 $H - t$ 层 KLL 风格压缩器，并在第 $H - t + 1$ 层包含一个线性压缩器。设 $S'_b(X)$ 是 $S'(X)$ 的第 b 个压缩器。

接下来，令 $R(S, x, h)$ 表示在数据流结束时，草图 S 中高度不超过 $h' \le h$ 的所有压缩器内点 x 的秩。为方便起见，我们将 $R(x, 0)$ 设为 x 的真实秩

版权所有 ©2023 本文版权归作者所有

在输入流中。令 $\mathrm{err}(S,\ x,\ h)\ =\ R(S,\ x,\ h)\ -R(S,\ x,\ h-1)$ 表示层级 h 的压缩器导致 x 的近似秩发生的总变化。总误差可分解为每个压缩器产生的误差之和，即 $\sup_x |R(x,\ 0)\ - S'(x)| = \sum_{h=1}^{H} \mathrm{err}(S',\ x,\ h)$。

分析 KLL 压缩器。在 S 和 S' 中，流元素只会从较低层的压缩器移动到较高层的压缩器；处理数据流过程中，任意时刻层级为 b 的压缩器，完全由截至该时刻的更低层压缩器决定。因此，对于所有 $b < H - t$，$S'_b(X) = S(X)$。

在 KLL 草图中，最低层的压缩器容量都恰好为 2。正如作者所指出的，一串容量均为 2 的 H'' 个压缩器本质上就是一个采样器：每 $2^{H''}$ 个元素中，它们会均匀地选取一个，并以权重 $2^{H''}$ 输出。这意味着，在 KLL 草图和线性压缩器草图中，这些压缩器都可以用 $O(1)$ 空间实现。

为了处理其他 KLL 压缩器，我们使用 [7] 中的一个定理作为关键引理：

Theorem 4.2. (Theorem 3 in [7]) 考虑非 *GK KLL* 草图 $S(X)$，其高度为 H，且层级 h 的压缩器容量为 $k_h \geq kc^{H-h}$。令H'' 表示压缩器大小大于 *2* 的层级（即压缩器不只是进行采样的层级）的高度。对于任意 $H' > H''$，有

$$\Pr\left[\sum_{h=1}^{H} \mathrm{err}(S,x,h) > 2\varepsilon n\right] \leq 2\exp\left(-c\varepsilon^2 k 2^{H-H''}/32\right) + 2\exp\left(-C\varepsilon^2 k^2 2^{2(H-H')}\right).$$

分析线性压缩器。如前所述，我们将逐次分析线性压缩器压缩引入的误差。具体而言，我们分析线性压缩器草图在一次压缩结束到紧接着的下一次压缩结束之间的变化。在此期间，共有 d 个权重为2^{H-t} 的项目被添加到线性压缩器中；如果线性压缩器从未进行过压缩，则$d = tk$，否则$d = tk/2$。

令 f 为压缩前完整线性压缩器的分段线性秩函数，其端点集 Z 由 $z_1 < z_2 < \cdots < z_{tk}$ 构成，权重函数为 w。令 f' 为压缩后线性压缩器的分段线性秩函数，其端点集为 $Z' \subset Z$，权重函数为 w'，且 $|Z'| = |Z|/2$。

线性压缩器的压缩过程会移除线性压缩器中的一些项目。连续段是指在排序顺序中彼此相邻的一系列被移除元素。我们证明，所引入的误差由线性压缩器引入的误差不超过移位权重最大的连续段。

Lemma 4.1. 将 $Z \setminus Z'$ 整理为由相邻被删除元素构成的连续段，并令 F_i 为第 i 个连续段的总权重。则 上确界$_{z\in Z} |f(z) - f'(z)| \leq$ 最大值 $_i F_i$。

证明。固定一个端点为 a 和 b 的连续段，令其总权重为 $F = \sum_{i=a+1}^{b-1} w(z_i)$。考虑该连续段中的任意点 z_j，使得 $a < j < b$。它原来的秩为 $f(z_j) = \sum_{i=1}^{j} w(z_i)$，而根据构造，它现在的秩为 $f'(z_j) = \underline{\sum_{i=1}^{a} w(z_i)} + {}^{F+w_{(z_b)}z_b - z_a}(z_j - z_a)$。因此，

$$\begin{aligned}
|f(z_j) - f'(z_j)| &= \left|\sum_{i=a+1}^{j} w(z_i) - \frac{F + w(z_b)}{z_b - z_a}(z_j - z_a)\right| \\
&= \left|\sum_{i=a+1}^{j} w(z_i) - \sum_{i=a+1}^{b} w(z_i)\frac{z_j - z_a}{z_b - z_a}\right| \\
&\leq \sum_{i=a+1}^{b-1} w(z_i) \\
&= F. \qquad \square
\end{aligned}$$

接下来，我们证明线性压缩步骤引入的较大误差出现在某个被丢弃的端点处：

Lemma 4.2. 存在某个 $z_i \in Z \setminus Z'$，使得上确界 $_{x\in[z_1,z_{tk}]} |f(x) - f'(x)| = |f(z_i) - f'(z_i)|$。

证明。考虑任意一点 $x \in [z_1, z_d]$。如果 x 是压缩后保留的端点之一 $z_j \in Z'$，那么根据构造，$f(x) = \sum_{i\leq j} w(z_i) = f'(x)$。我们的论断与 x 是否为 $Z \setminus Z'$ 中的一个端点时的误差无关。

于是，假设 x 不在原始端点集 Z中。令 z_a 和 z_b 分别为 Z中 x 的左邻点和右邻点。根据线性压缩器的定义，

$$\begin{aligned}
f(z_a) &= \sum_{i=1}^{a} w(z_i), \\
f(x) &= \sum_{i=1}^{a} w(z_i) + \frac{w(z_b)}{z_b - z_a}(x - z_a), \\
f(z_b) &= \sum_{i=1}^{b} w(z_b).
\end{aligned}$$

令 $z_{a'}$ 和 $z_{b'}$ 分别为 Z'中 x 的左邻点和右邻点。根据定义，权重 $W := w'(z_{b'}) = \sum_{i=a'+1}^{b'} w(z_i)$，因此我们有

$$f'(x) = \sum_{i=1}^{a'} w(z_i) + \frac{W}{z_{b'} - z_{a'}}(x - z_{a'}).$$

版权所有 ©2023 本文版权归作者所有

因此，

$$f(x) - f(x') = \sum_{i=a'+1}^{a} w(z_i) + \left(\frac{w(z_b)}{z_b - z_a} - \frac{W}{z_{b'} - z_{a'}}\right)(x - z_{a'}).$$

注意，该表达式在区间 $[z_a, z_b] \subset [z_{a'}, z_{b'}]$ 上的极值出现在 z_a 或 z_b处，具体取决于 $D = {}^{w(}_{z_b - z_a} - {}^{W}_{z_{b'} - z_{a'})}$ 的符号。无论哪种情况，$|f(x) - f(x')|$ 都会在端点 z_a 或 z_b之一处取得最大值，证毕。□

我们用一个简单的计数论证来界定线性压缩器中大部分权重的大小：

LEMMA 4.3. 考虑一个刚完成第 c 次压缩的线性压缩器。线性压缩器中至少有一半的端点 Z 的权重不超过 $(2c+3)2^{H-t}$。

证明。每个点进入线性压缩器时的权重均为 2^{H-t}。经过 c 次压缩后，共有 $(2+c)tk/2$ 个这样的点进入压缩器。压缩操作会保持点的总权重不变，因此压缩器的总权重为 $(2+c)2^{H-t}tk/2$。

假设当前压缩器中的 tk 个点有超过一半的权重至多为 T。这些点的总权重大于 $Ttk/4$，而其余点的权重各自至少为2^{H-t} ，因此总权重至少为 $2^{H-t}tk/4$。所以总权重为 $(T + 2^{H-t})tk/4$。该权重不能超过守恒的总权重 $(2+c)2^{H-t}tk/2$，因此我们有

$$\frac{(T + 2^{H-t})tk}{4} \le \frac{(2+c)2^{H-t}tk}{2}.$$

重新整理后，我们得到结果对任意$T \le (2c+3)2^{H-t}$都成立。□

结合这些引理，我们得到单次压缩步骤所引入误差的一个上界。

THEOREM 4.3. 假设所研究的压缩是第$(c+1)$次压缩。在这一步压缩中引入的误差为 $\sup_x |f(x) - f'(x)| \le (c+2)2^{H-t+1}$。

证明。 我们按如下方式构造一种特定的压缩后权重分布。令 f'' 为该压缩后状态的秩函数。在此期间，有 $tk/2$ 个权重为 2^{H-t} 的点首次被加入线性压缩器。

此外，还有 $tk/2$ 个点是上一次线性压缩后留下的。我们对 $tk/2$ 个新点排序，每四个点保留一个，其余点则丢弃，并将它们的权重重新分配给下一个秩最高的保留点（无论其属于哪种类型）。根据引理 4.3，线性压缩器中至少存在 $tk/4$ 个权重不超过 2^{H-t+c} 的已有点。我们对这些点排序，并每隔一个丢弃一个。总计，我们丢弃了所需的 $tk/2$ 个点。

注意，此次压缩中可能出现的最长连续段由一个已有点和三个（从四个新点中取出的）被丢弃的点组成。根据引理 4.1，此次压缩在任一原始端点上引入的误差不超过该连续段内各点权重之和：在本例中，该和为 $2^{H-t} + 3 \cdot (2c+3)2^{H-t} \le (c+2)2^{H-t+1}$。根据引理 4.2，我们发现， f''引入的误差为 $\sup_x |f(x) - f''(x)| \le (c+2)2^{H-t+1}$。

我们已经给出了线性压缩中优化问题的一个特定可行解。我们的实际算法会在所有此类可行解中找出使该误差函数最小的解；因此

$$\sup_x |f(x) - f'(x)| \le \sup_x |f(x) - f''(x)| \le (c+2)2^{H-t+1}. \qquad \square$$

结合 KLL 与线性压缩器。最后，我们结合对 KLL 和线性压缩器的分析，得到总体误差界，并证明定理 4.1。我们的分析与 [7] 中定理 4 的证明形式高度一致。

证明。 [定理 4 的证明。1] 首先，我们分析高度至多为 $H - t$ 的压缩器，包括采样压缩器。这些都是 KLL 风格的压缩器；根据定理 4.2，只要$\varepsilon k 2^s \ge c'\sqrt{\log(2/\delta)}$ 对足够小的 c' 成立，这些压缩器贡献的误差以概率 $1-\delta$ 至多为 εn。其次，我们分析顶部的 $s - t$ 个压缩器。这些压缩器引入的误差由等价的非 GK KLL 草图的误差界定，其中顶部有完整的s 个等大小压缩器。该误差进一步由 $\sum_{h=H-s+1}^{h} m_h w_h = \sum_{h=H-s+1}^{H} n/k = sn/k$ 界定，其中 m_h 是层级 h 的 KLL 压缩器被压缩的次数，而 $w_h = 2^h$ 是与该压缩器关联的权重；只要 $s \le k\varepsilon$，该误差至多为εn。按照 KLL 中的设定，取 $k = O((1/\varepsilon)\log\log(1/\delta))$ 和 $s = O(\log\log(1/\delta))$，即可同时满足这两个条件。

最后，我们分析大小为 tk 的单线性压缩器，它取代了顶部的 $t < s$ 个 KLL 压缩器。设 M 为线性

版权所有 ©2023 本文版权归作者所有

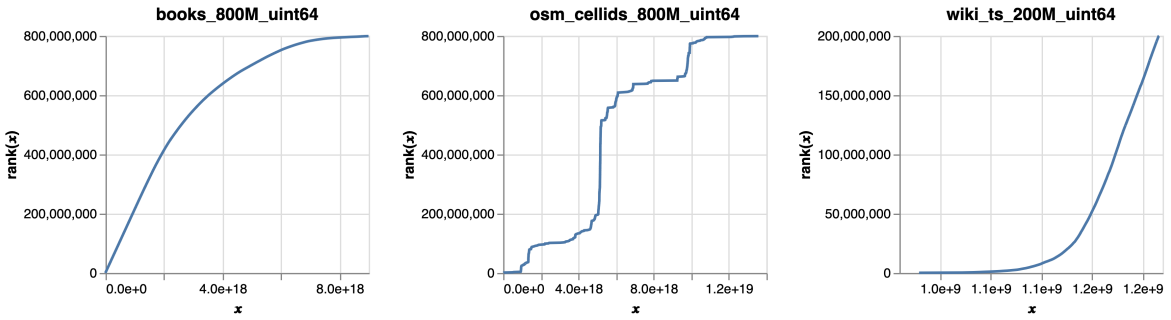

图 2：我们实验中使用的三个 SOSD 数据集的秩函数。这三个数据集的秩函数形状各不相同，因此我们能够在多种设置下比较这些算法。

压缩器的压缩次数。注意，由于线性压缩器每次压缩之间都会向其中添加 $tk/2$ 个条目，每个条目的权重为 2^{H-t}，因此 $M \leq 2n/(tk2^{H-t})$。应用定理 4.3，并对每次压缩引入的误差求和，总误差为

$$\begin{aligned}\sum_{c=1}^{M}(c+2)2^{H-t+1} &= M2^{H-t} + 2^{H-t}M(M+1) \\ &\leq 2^{H-t+1}M^2 \\ &\leq \frac{8n^2}{t^2k^2 2^{H-t}}.\end{aligned}$$

我们的压缩器在每个层级的大小设置方式与非 GK KLL 草图相同。与 KLL 分析一样，对于某个常数 $0 < c < 1$，我们有 $H \leq \log(n/ck) + 2$。因此，我们的误差有如下界：

$$\frac{8n^2}{t^2k^2 2^{\log(n/ck)+2-t}} \leq \frac{8ckn^2}{t^2k^2n2^{2-t}} = \frac{cn2^{t+1}}{t^2k}.$$

对于常数 t 和任意 $k = O((1/\varepsilon)\log\log(1/\delta))$，与 KLL 中一样，该值至多为 εn。因此，草图的总误差为 $O(\varepsilon n)$，符合要求。

草图的各个部分都会占用一定空间。KLL 压缩器的大小按几何级数递增，因此草图中 KLL 部分所用的空间主要由顶部的 $s-t$ 个压缩器决定，占用 $O(sk) = O((1/\varepsilon)\log^2\log(1/\delta))$ 空间。线性压缩器草图的每个元素所占空间是 KLL 压缩器的两倍，总计占用 $O(tk) = O(k)$ 空间，因此总空间使用量为 $O((1/\varepsilon)\log^2\log(1/\delta))$。

□

5 实验

我们为该算法编写了高性能实现，并在广泛的空间参数范围 k 和多个线性压缩器高度 t下评估了其经验误差。我们的实验基于近期用于学习型索引结构的 SOSD 基准测试套件 [8, 11]开展。每个 SOSD 基准测试都包含大量（通常为 2 亿至 8 亿个）64 位无符号整数值。我们尤其关注 books、osm cellid 和 wiki ts 数据集，因为如图 2 所示，这三个数据集的秩函数形状截然不同。

参数化。 该算法由 KLL 空间参数 k 和 t 参数化：前者决定最大压缩器和线性压缩器的大小，后者表示由线性压缩器替换的 KLL 压缩器数量。我们的最坏情况界适用于任意常数 t，但该界关于 t 呈指数增长。在实践中，我们尝试了各种较小但非零的取值（$t = 1$, 2, 3）。我们认为 t 是一个可调参数，可根据所需的实测性能和最坏情况保证进行调整，并预计它会针对不同应用分别进行适当选择。

实现细节。我们用 C++ 实现了算法，并提供 Python 绑定来管理实验和进行数据分析。我们的实现性能相当不错：在非正式实验中，其吞吐量仅比经过高度优化、可用于生产环境的 KLL 实现低约三倍。这种高性能实现使我们能够处理 SOSD 数据集的全部内容；在初步工作中，我们发现，许多有前景的算法只有在规模适中的数据集（少于一百万个点）上才会优于 KLL。我们的实现支持任意整数 $t \geq 0$：当 $t = 0$ 时，我们的实现与常见的 KLL 实现变体完全相同，但不包含 Greenwald-Khanna 草图。

基线方法。 我们的算法最自然的比较对象是 KLL，因为 KLL 草图可以视为一种不包含线性压缩器的线性压缩器草图。我们使用自己的（非 GK）KLL 实现进行了实验（通过设置 $t = 0$），并使用一个开源实现验证了这些结果——

版权所有 ©2023 本文版权归作者所有

该实现来自 Facebook 的 Folly 库 [1]。与大多数已实现的 KLL 草图版本一样，这两种实现都不包含最终的 Greenwald-Khanna 草图，而要达到空间最优性则必须包含它。

除了具有最坏情况保证的非 GK KLL 草图外，我们还对 t-digest [3]，进行了实验。t-digest 在实践中很常用，但已知其最坏情况下的性能可能任意差[2]。我们使用了 digestible 库 [13]中的 t-digest C++ 实现。

数据流顺序。我们发现，许多不具备最坏情况保证的流式分位数近似算法，在输入流采用特定顺序时误差远低于 KLL 草图，但在其他输入顺序下误差却很高。例如，图 1 显示，即使输入是秩函数平滑的固定数据集（books 数据集），也存在一种对抗性顺序，会导致 t-digest 的近似误差很高。这一观察或许本身就具有独立的研究价值。

我们针对每个数据集的多种输入顺序，对线性压缩器草图和基线方法进行了评估：

- 随机：使用固定种子对数据进行随机打乱。
- 排序：按排序顺序提供数据。
- 前半部分按顺序排列，后半部分逆序排列：数据流的前半部分按顺序包含排序后数据的前半部分；后半部分则按逆序呈现排序后数据的后半部分。
- 交替顺序：数据流依次包含最小元素、最大元素、第二小元素、第二大元素，以此类推。这是图 1 中的对抗性顺序。

5.1 实验结果与讨论。我们从实验中获取洞见的主要工具是空间—误差权衡曲线，它展示了草图所需总空间与草图定义的近似值和精确秩函数之间的经验误差的关系。我们针对 SOSD 的三个不同数据集、四种不同的排序顺序以及四种不同的算法绘制了这些曲线，结果如图 4所示。我们使用平均 L1 误差，其定义为对于数据集 X，$\sum_{x \in X} |f(x) - f'(x)|$。从定性角度看，线性压缩器草图从未显著逊于 KLL，即使面对交替顺序等对抗性输入顺序也是如此；而且它的表现往往可与 t-digest 相媲美，甚至更胜一筹。差异在图书数据集上最为明显，该数据集的累积分布函数平滑，线性压缩器草图的分段线性表示能够极其准确地近似它。

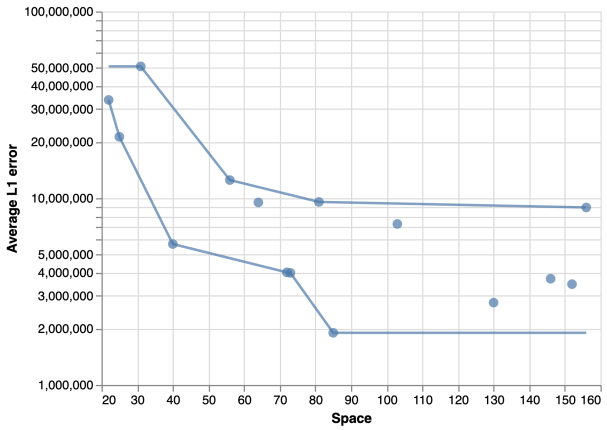

图 3：在 $t = 2$ （图书数据集，随机顺序）条件下，线性压缩器草图的空间-误差数据点“包络”或凸包示例。它展示了在给定空间用量时，用户可以预期线性压缩器草图产生的误差范围。

为了更定量地了解线性压缩器草图相较于 KLL 和 t-digest 的性能，我们绘制了“可能误差比凸包”图，如图5所示。要得到这样的凸包，我们首先确定了图4中每种算法对应数据点的上边界和下边界。这些边界构成一个“包络”或凸包，涵盖每个数据集的所有数据点：图3展示了一个此类凸包的示例。随后，我们对包络进行插值以得到平滑曲线（如图3所示），并计算相对于另一种算法包络的比值（在上边界/下边界和下边界/上边界两组之间），从而生成一个凸包，展示两种算法在“最坏情况”和“最佳情况”性能之间的表现范围。我们看到，线性压缩器草图的误差比 KLL $3\times$ 更差、$10\times$ 更好，且比 t-digest $10\times$ 更差、$20\times$ 更好。

致谢

Justin Y. Chen 获得了 MathWorks 工程奖学金、GIST-MIT 合作研究基金以及美国国家科学基金会 CCF-2006798 项目的资助。Justin Y. Chen、Shyam Narayanan 和 Sandeep Silwal 获得了美国国家科学基金会研究生研究奖学金，资助编号 1745302。Nicholas Schiefer、Justin Y. Chen、Piotr Indyk、Shyam Narayanan 和 Sandeep Silwal 获得了西蒙斯研究员奖的资助。Piotr Indyk 获得了美国国家科学基金会 TRIPODS 计划的资助（项目编号 DMS-2022448）。

我们感谢 SylviaH¨urlimann、Jessica Balik 和匿名评审者提出的宝贵建议。

版权所有 ©2023 本文版权归作者所有

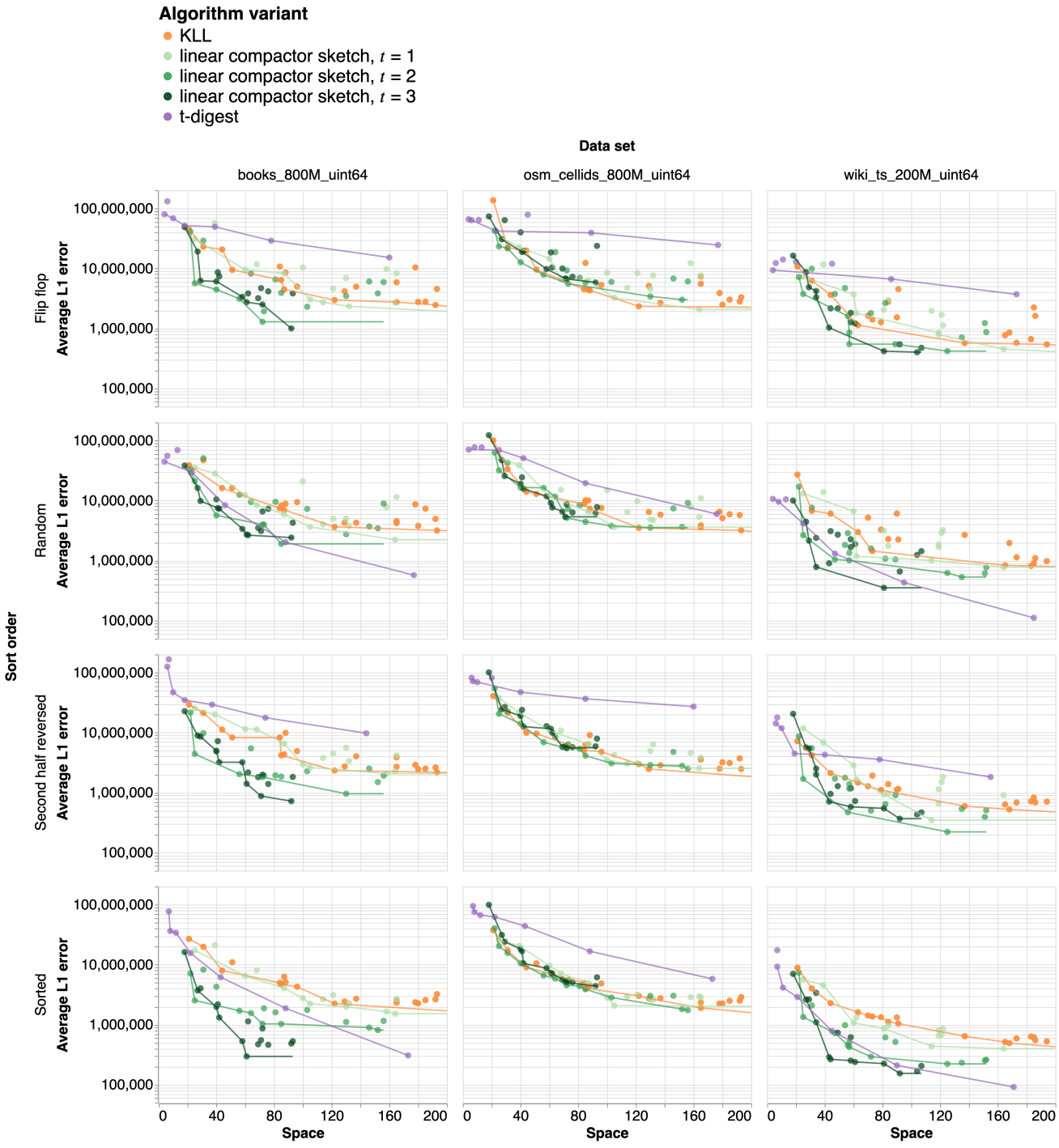

图4：上文所述的 SOSD 中三个不同数据集和四种不同排序顺序下，基线方法与线性压缩器草图的空间-误差权衡曲线。标记表示各个草图，曲线则表示观测到的可能性下边界（即上文所述的下包络），以突出各算法的总体能力。曲线越靠左下方，算法越好，表示在给定空间用量下误差更低。

版权所有 ©2023 本文版权归作者所有

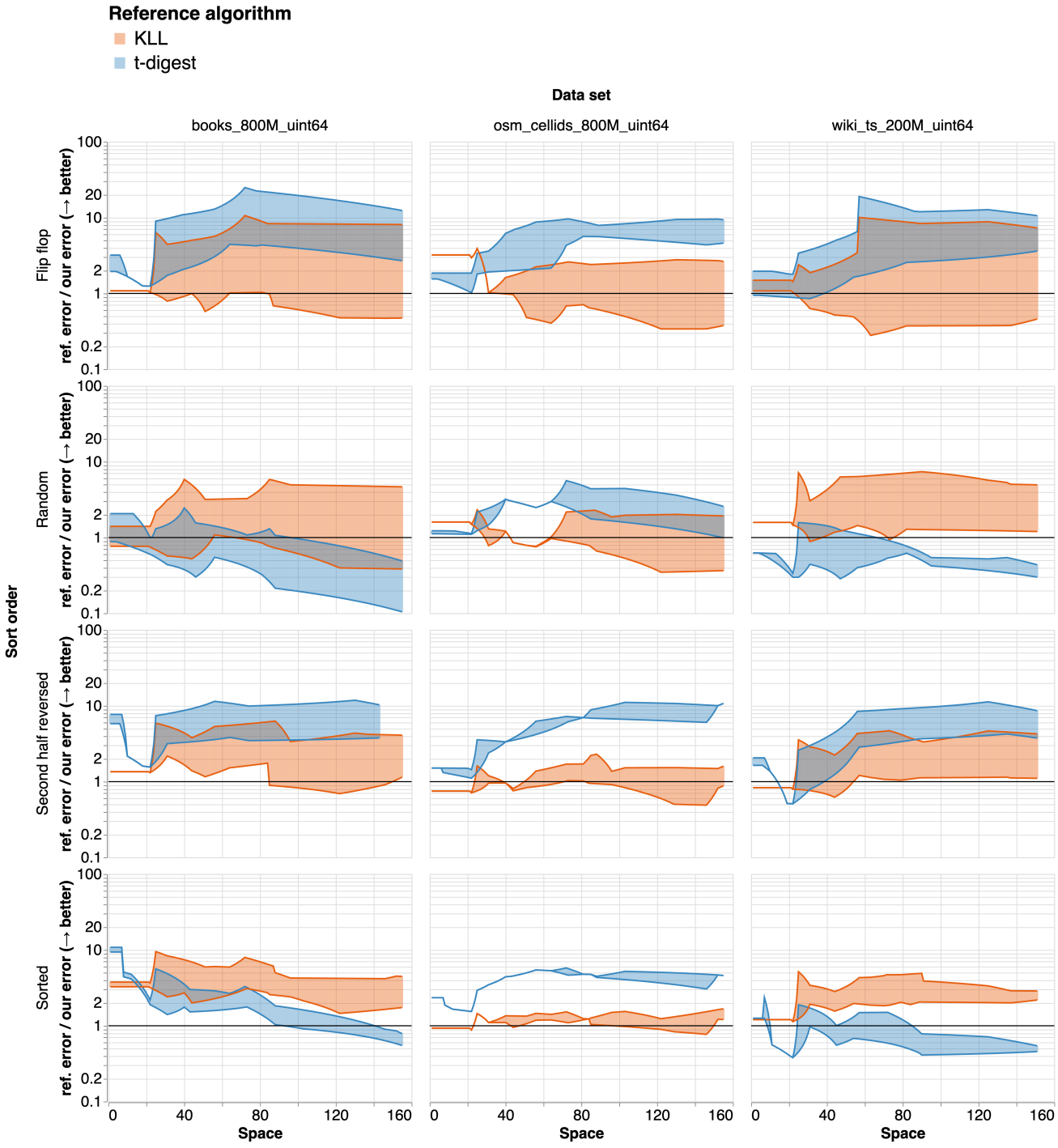

图5：表示参考算法（KLL 或 t-digest）所达到的误差与采用 $t = 2$ 的线性压缩器草图所达到的误差之比可能取值的凸包。填充区域表示与图4中的实验结果相符的比值范围。我们看到，线性压缩器草图的误差始终不比 KLL 的误差差 $3\times$，而且通常能达到与 t-digest 相当、甚至有时远低于 t-digest 的误差。

版权所有 ©2023 本文版权归作者所有

参考文献

[1] Folly：Facebook 的开源库。https://github.com/facebook/folly.[2] Graham Cormode、Abhinav Mishra、Joseph Ross 和 Pavel Vesel`y。《中位数上的理论与实践：相对误差分位数算法的最坏情况比较》。载于 *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*，第 2722–2731 页，2021年。[3] Ted Dunning。《t-digest：高效估计分布》。 *Software Impacts*，7:100049，2021年。[4] Michael Greenwald 和 Sanjeev Khanna。《节省空间的分位数摘要在线计算》。*ACM SIGMOD Record*，30(2):58–66，2001年。[5] Nikita Ivkin、Edo Liberty、Kevin Lang、Zohar Karnin 和 Vladimir Braverman。《小空间、低更新时间的流式分位数算法》。 *Sensors*，22 24 2022( )，。[6] Hosagrahar Visvesvaraya Jagadish、Nick Koudas、S Muthukrishnan、Viswanath Poosala、Kenneth C Sev- cik 和 Torsten Suel。《具有质量保证的最优直方图》。载于 *VLDB*，第 98 卷，第 24–27 页，1998年。[7] Zohar Karnin、Kevin Lang 和 Edo Liberty。《数据流中的最优分位数近似》。载于 *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)* 71 78, 2016 ，第 – 页。[8] Andreas Kipf、Ryan Marcus、Alexander van Renen、Mi- hail Stoian、Alfons Kemper、Tim Kraska 和 Thomas Neumann。《SOSD：学习型索引基准测试》。*NeurIPS Workshop on Machine Learning for Systems*，2019年。[9] Tim Kraska、Alex Beutel、Ed H. Chi、Jeffrey Dean 和 Neoklis Polyzotis。《学习型索引结构的优势》。载于 *Proceedings of the 2018 International Conference on Management of Data*，SIGMOD '18，第 489–504 页，纽约，NY，美国，2018年。美国计算机协会。[10] Gurmeet Singh Manku、Sridhar Rajagopalan 和 Bruce G Lindsay。《单遍处理且使用有限内存的近似中位数及其他分位数》。*SIGMOD Record* 27 2 426 435, 1998 *ACM* ，( ): –。[11] Ryan Marcus、Andreas Kipf、Alexander van Renen、Mihail Stoian、Sanchit Misra、Alfons Kemper、Thomas Neumann 和 Tim Kraska。《学习型索引基准测试》。*Proc. VLDB Endow.*，14(1):1–13，2020年。[12] Michael Mitzenmacher 和 Sergei Vassilvitskii。 *Algorithms with Predictions* 646 662 ，第 – 页。剑桥大学出版社，2021年。[13] Dan Morton。digestible：一种现代 C++ 实现，用于合并型 t-digest 数据结构。https: // github.com/SpirentOrion/digestible。[14] Lu Wang、Ge Luo、Ke Yi 和 Graham Cormode。《数据流上的分位数：一项实验研究》。载于 *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*，SIGMOD '13，第 737–748 页，纽约，NY，美国，2013年。美国计算机协会。

版权所有 ©2023 本文版权归作者所有