Streaming self-scaling histograms with stability and optimality guarantees

Hervé Brönnimann[1] and Joannès Vermorel[2]⋆

[1] Computer and Information Science, Polytechnic University, Six Metrotech Center, Brooklyn, NY 11201, USA. hbr@poly.edu

[2] École normale supérieure, 45 Rue d'Ulm, 75230 Paris, Cedex 5, France. joannes@vermorel.com

Abstract. Histograms provide discrete efficient representation of continuous data and can be constrained to fit various purposes. We provide the first offline sub-quadratic algorithm which applies to any *sub-additive* histogram cost functions. Sub-additive cost functions includes in particular all the ℓ_p norms. This algorithm outputs a histogram with at most $2m$ buckets and cost at most $(1+\varepsilon)$ the optimal cost achievable with m buckets. Then we show how to adapt our algorithm in a streaming context but restricted for a certain cost function that bounds the maximum error for one-dimensional aggregate range queries. An advantage of our approach is that is only uses $\mathcal{O}(m)$ space and $\mathcal{O}(\log(m))$ per item processing time, m being the number of buckets, independent from N the number of values processed. We also prove that the histogram outputs of our streaming algorithm are very stable. While reading the streams, the output histogram undergoes at most $\mathcal{O}(m\log(N))$ boundary displacements. To our knowledge, this concept of *stability* has never been studied for streaming histograms. Stability implies that extra properties associated to the buckets don't have to be reconstructed too often, strengthening their quality and accuracy. Our empirical evaluations confirm the good behavior of our streaming algorithm in practice.

1 Introduction

Histograms provide discrete efficient representation of continuous data. Over the last decades, they have been mainly used for cost-based query optimization [11,12,22] and for approximate query answering [6,15,20,21].

The problem of central interest in approximating distributions with histograms can be stated as follow: *"For a given number m of buckets, how should the boundaries be chosen in order to obtain the most accurate histogram for a particular task?"*. Note that the dual problem *"For a given level of accuracy and for a particular task, how many buckets are needed?"* has also been studied in [13,19], but little has been done in this perspective. In addition, we will focus in this paper in computing such optimal or near-optimal histograms in a streaming context (see [2,18] for recent surveys of data streams and applications).

⋆ Work by the second author was performed while doing summer research at Polytechnic University, on leave from École normale supérieure, Paris, France.

Notation and cost functions. Let $\mathbf{V} = (v_1, v_2, ..., v_N) \in \mathbb{R}^N$ be a set of real values, supposed to model a numerical attribute in a database, or a value arriving on a data stream. Under data stream context, we will assume N very large. See the excellent survey [18] for algorithms to compute various quantities of a stream (e.g., ℓ_p norms, quantiles, etc.) under streaming models. A histogram H is a collection of *buckets*, designed by B in the following. Each bucket is characterized by its *length*, the distance between the bucket boundaries noted $l(B)$ and its *weight*, the number of values contained by the bucket noted $w(B)$.

A common way of defining the quality of a histogram is to provide a *cost function* c that associates a cost to each bucket. The quality of the overall histogram H is then measured by the *total cost* $c(H) = \sum c(B_i)$. The optimal histogram for $\mathbf{V}$ with m buckets, designed by H_m^{opt}, is the histogram that minimizes the total cost.

The *squared bucket count* cost function is defined by $c_{count}(B) = w(B)^2$. The total cost $c(H)$ is clearly minimized when all the buckets have the same weight. The more complicated cost functions considered in the literature are the *intra-bucket dispersion* $c_1(B) = \sum_{v \in B} |v - \text{AVG}[B]|$ and the *intra-bucket variance* $c_2(B) = \sum_{v \in B}(v - \text{AVG}[B])^2$. Histograms that minimize the total intra-bucket variance C_2 are often called *V-optimal histograms*. The *rangesum* histogram quality criterion is defined as the expected value of $|\mathbf{V}[\ell, r) - H[\ell, r)|^2$ over all half-open query intervals $[\ell, r)$ is also considered in [20], but this criterion does not quite fit into the simple framework presented here. All these cost functions verify the simple following property.

Definition 1 (Sub-additivity). *If c is a cost function, we will say that c is* sub-additive *if, for all B_1, B_2 and B buckets such that $B_1 \cap B_2 = \emptyset$ and $(B_1 \cup B_2) \subset B$ then we have $c(B_1) + c(B_2) \leq c(B)$.*

In this paper, we also have a particular interest for a very simple cost function, the *freedom* cost function $c_{freedom}(B) = l(B)w(B)$. It is sub-additive and was introduced by Greenwald in [7] under the name of the *available error* along with the EquiError algorithm. It admits a very simple explanation in terms of mechanics: assuming the elements v are particles free to take any value in B (in the absence of any information about the distribution of particles in a bucket), then $c_{freedom}(B)$ represents the momentum, or quantity of motion, of all the particles $v \in B$. A more geometric insight of the freedom cost function in given in [7]. When looking at the cumulative distribution function (CDF) of $\mathbf{V}$, the bucket appears as rectangular constraints of dimension $l(B) \times w(B)$ fitted on the curve, and all that is know is that the CDF goes inside those rectangles. Minimizing the freedom therefore minimizes the area the CDF curve can lie in, or equivalently the uncertainty on the area below the curve.

Freedom is also the most natural bound on range query errors: if we use a histogram to answer range queries and know nothing else about the buckets other than their bounds, width, and weight, then the maximum error to answer a query whose endpoints fall into two buckets B_{left} and B_{right} is given exactly by $c_{freedom}(B_{left}) + c_{freedom}(B_{right})$. Thus the freedom cost of a histogram is the best criteria to minimize the maximum range query error, and is also an upper bound on the total intra-bucket dispersion c_1.

Our results. We introduce Echap a simple offline algorithm, which applies to all sub-additive cost functions. Echap outputs a histogram with at most $2m$ buckets, and costs at most $(1+\varepsilon)$ the optimal cost achievable with m buckets[3]. To our knowledge, Echap is the first algorithm with a sub-quadratic complexity that provides guaranteed histograms for any sub-additive cost functions.

Then we show how to adapt our algorithm Sechap in a streaming context with $\mathcal{O}(\log(m))$ processing time per incoming value in the case of the freedom cost function. Unfortunately, we are not able to provide the same theoretical guarantees for this streaming variant of Echap. But we prove that the algorithm maintains an upper bound for the actual cost, even though we cannot show that this bound is always within a constant factor of optimal. We nevertheless give empirical evidence of the good behavior of Sechap in practice.

As with [9], our algorithms rely on a sketch used to construct the output histogram. Instead of a wavelet, our sketch is simply a histogram with more buckets. Our sketch improves the previously known results in using only a space $O(m)$ independent of N. It is also simple and practical. The histogram evolves by mutations, where a mutation consist either of merging two adjacent buckets or of splitting a bucket in two parts.

We also prove that the histogram evolution, while reading the stream, involves only $O(m \log N)$ mutations in the worst case. To our knowledge, this is the first result on histograms that constrains the "amount of change" of the histograms when reading an additional value. In addition, empirical evaluations indicate that convergence is even faster in practice. Since histograms could be used to capture further properties of the data (maintaining some aggregate value per bucket for example), stability implies that those properties won't have to be reconstructed too often, strengthening their accuracy and improving the overall computational efficiency.

Related work on histograms. Histograms whose bucket boundaries are determined by the input are called *self-scaling* (or also *self-tuning*). The *equi-width histograms*, also called trivial histograms, are the simplest histograms where all buckets have equal length. They are not self-scaling, and in practice they perform poorly [22]. The *equi-depth histograms* are defined as histograms where all buckets have the same weight. The optimal equi-depth histogram could be computed in $O(N^2)$ time with the P^2 algorithm as presented in [14]. Heuristic algorithms are presented in [1, 3]. Data stream algorithms have recently been given to maintain the quantiles of a distribution, an equivalent problem [4,8,16]. These solutions typically offer probabilistic or deterministic guarantees on the approximate rank errors they compute of $O(N/m)$ for a storage $O(m \log(N/m))$.

The freedom cost function is introduced in [7] under the name of the *available error* along with the EquiError algorithm. EquiError is very similar to

[3] Some authors call such a histogram a $(2, 1+\varepsilon)$-optimal histogram.

SECHAP. The main differences between both algorithms are where the bucket is split, and how they maintain an estimate of the weights of each of the new buckets. SECHAP splits the buckets in half and keeps separate under- and over-counts, trying to maintain a fixed ratio between the min and the max freedom. Instead, EQUIERROR attempts to maintain a uniform error per bucket, hence the addition of a single element may force all the bucket boundaries to change based on a cubic spline approximation of the CDF, and propagation of this may force the algorithm to require $\mathcal{O}(m)$ processing time per item. In addition, the paper [7] does not prove any of the error bounds we present here. However, it makes very good points about why self-scaling algorithms (of which are all the algorithms discussed above) are more desirable for capturing and representing statistical distributions and offers valuable practical experience.

Recent results have been obtained for the ℓ_p *optimal histograms*, where the sum for every bucket of a weighted moment is minimized (see the next section for a formal definition). Typically the norm of interest are ℓ_1 (average absolute error) [5] and ℓ_2 (root mean square error) [5, 9, 10, 13, 17]. A static dynamic programming algorithm that compute the ℓ_2-optimal histogram is presented in [13], it takes $O(mN^2)$ using $O(mN)$ space. This dynamic programming algorithm could easily be modified to return optimal histograms for any sub-additive cost function with the same complexities. Recent work using wavelets has been done in the streaming context in order to find proved ε-approximation of the optimal histogram [5, 9, 10]. Those wavelet-based solutions typically consist of two algorithms: a sketching algorithm that preprocess the input to some data structure (a certain number of wavelet coefficients) and a reconstruction algorithm that performs some computation on that structure to output the final approximation (usually through a dynamic programming algorithm). Gilbert et al. [5] present an algorithm that computes an ϵ-approximation for either the ℓ_1 or ℓ_2 norm with time to process a single update, the time to reconstruct the histogram and the size of the sketch all bounded by $poly(m, \ln(N), \ln(\|\mathbf{V}\|), 1/\varepsilon)$. This result is improved in [9] with processing time $O(1)$ per source value, a sketch space of $m(\ln N \ln \|\mathbf{V}\| \varepsilon)^{O(1)}$ and a histogram reconstruction in time $O((m \ln N \ln \|\mathbf{V}\|/\varepsilon)^{O(1)})$.

Spline-based histograms, where the maximum absolute difference between a value and the average of the values in its buckets in minimized, have also been studied [22]. The dynamic programming algorithm proposed in [13] could be easily adapted to find the optimal histogram in $O(mN^2)$, but to our knowledge, no online algorithm has been proposed so far.

2 Offline histograms

In this section, we introduce our ECHAP algorithm to compute histograms with optimality guarantees for any sub-additive cost function. These results provide the basis for our streaming variant in the next section. First we provide some results about equi-cost histograms. The main result is the ECHAP algorithm of the second subsection.

2.1 Near Equi-Cost Histograms

The main tool behind our analysis is the notion of *equi-cost histogram*: this is a histogram H^{eq}_m in which every bucket has the same cost $C(H^{\mathrm{eq}}_m)/m$. We can relax this notion by introducing a *δ-near equi-cost histogram*, in which the most expensive bucket B_M costs at most $(1+\delta)C(B_\mu)$ where B_μ is the cheapest bucket. As defined by, e.g., say [20], we say that a histogram is (α,β)-*optimal* if it has at most αm buckets and cost $\beta C(H^{\mathrm{opt}}_m)$.

Theorem 1. *Let C be a sub-additive cost function, and $k>0$ a parameter (not necessarily integral). If $H^{eq}_{\lceil (k+1)m\rceil}$ is the equi-cost histogram with $\lceil (k+1)m\rceil$ buckets, then $H^{eq}_{\lceil (k+1)m\rceil}$ is $(k+1, 1+\frac{1}{k})$-optimal. More generally, a δ-near equi-cost histogram $H^{\delta eq}_{\lceil (k+1)m\rceil}$ with $\lceil (k+1)m\rceil$ buckets is $(k+1, 1+\frac{1+\delta}{k-\delta})$-optimal.*

Proof: Let us write $H^{\delta\mathrm{eq}}$ for short, and let B_μ and B_M be the cheapest and most expensive buckets (respectively) out of $H^{\delta\mathrm{eq}}$. It suffices to prove the second inequality, since the first follows with $\delta=0$.

At most m buckets of $H^{\delta\mathrm{eq}}$ may contain a boundary of H^{opt}_m (let's call them *crossing* buckets, and denote them by $H^{\delta\mathrm{eq}}_{cross}$), the other buckets of $H^{\delta\mathrm{eq}}$ are entirely contained inside those of H^{opt} (*enclosed* buckets, denoted by $H^{\delta\mathrm{eq}}_{encl}$). The cost of the crossing buckets is at most $mC(B_M)$. The (near) equi-cost property implies $C(B_M)\le(1+\delta)C(B_\mu)$. Since $H^{\delta\mathrm{eq}}$ has $\lceil (k+1)m\rceil$ buckets, we have

$$C(B_\mu)\le\frac{1}{\lceil (k+1)m\rceil}C(H^{\delta\mathrm{eq}})\le\frac{1}{(k+1)m}C(H^{\delta\mathrm{eq}}).$$

From these two inequalities, we get $C(H^{\delta\mathrm{eq}}_{cross})\le\frac{1+\delta}{k+1}C(H^{\delta\mathrm{eq}})$. By the sub-additivity of the cost function, the total cost of the enclosed buckets cannot add up to more than $C(H^{\mathrm{opt}}_m)$. Overall, we have

$$C(H^{\delta\mathrm{eq}})=C(H^{\delta\mathrm{eq}}_{cross})+C(H^{\delta\mathrm{eq}}_{encl})\le\frac{1+\delta}{k+1}C(H^{\delta\mathrm{eq}})+C(H^{\mathrm{opt}}_m).$$

The result on δ-near equi-cost histograms follows by simple algebra. □

Given $\varepsilon>0$, and an algorithm to compute a δ-near equi-cost histogram $H^{\delta\mathrm{eq}}$, one may pick $k=\delta+\frac{1+\delta}{\varepsilon}=O(1/\varepsilon)$ to obtain a $(O(\frac{1}{\varepsilon}),1+\varepsilon)$ approximations for any $\varepsilon>0$. Unfortunately, $\lceil (k+1)m\rceil$ is quite a lot of buckets to obtain a lower bound on the cost of the m-bucket histogram H^{opt}_m. We may resolve this shortcoming by using the $\lceil (k+1)m\rceil$ boundaries of $H^{\delta\mathrm{eq}}$ for constructing a near-optimal histogram $H^\star_{2m}$ with $2m$ buckets, using dynamic programming [10, 13]. The previous lemma implicitly proves that this yields a $(2,1+\varepsilon)$ scheme. Indeed, combining all the buckets enclosed in a bucket of H^{opt}_m into a single bucket will not increase the cost, by the sub-additivity property. Such a histogram will have at most $2m$ buckets. Of course, we do not know H^{opt} but since there is such a histogram, the best histogram $H^\star_{2m}$ with $2m$ buckets will also match the same cost.

Corollary 1. *Let C be a sub-additive cost function, and $k > 0$ a parameter (not necessarily integral). If H_m^{opt} is an optimal histogram with m buckets, $H^{\delta eq} = H^{\delta eq}_{\lceil (k+1)m \rceil}$ a δ-near equi-cost histogram with $\lceil (k+1)m \rceil$ buckets, and $H^\star_{2m}$ the cheapest histogram with $2m$ buckets whose bucket boundaries are chosen among those of $H^{\delta eq}$, then*

$$C(H^\star_{2m}) \leq \Big(1 + \frac{1+\delta}{k-\delta}\Big) C(H_m^{opt}).$$

Note that this approach would not work for reducing any histogram, but it works specifically because we start with a (near) equi-cost histogram. Also, there is an inherent limitation in the proof here above: we cannot obtain similar result for $H^\star_m$ instead of $H^\star_{2m}$. Indeed, $H^\star_m$ will be resulting from the merge of buckets crossing the boundaries of the optimal histogram. Our approach would require a "reverse" inequality like $C(B_1) + C(B_2) \geq \alpha C(B)$ (see the sub-additivity definition), but this inequality is not true in general except for $\alpha = 0$.

2.2 The Echap algorithm

The overall approach, at least in spirit, is thus to compute a δ-near equi-cost histogram $H^{\delta \mathrm{eq}}_{\lceil (k+1)m \rceil}$ and from it extract the best histogram $H^\star_{2m}$. Unfortunately, we can't quite do that, but we compute a histogram H with $2\lceil (k+1)m \rceil$ buckets whose cost is similar to that of the equi-cost histogram.

Our algorithm, dubbed Echap (for Equi-Cost Histogram APproximation), is simple and can be best explained in the non-streaming context. It only uses $O(m)$ space. Its pseudo-code is presented in Figure 1. The first step is to provide an arbitrary initialization for H. Intuitively, each step of the algorithm will consist in merging the two neighboring buckets with the smallest costs and in splitting in two the bucket with the highest cost. The algorithm terminates when some condition similar to the near-equi-cost condition is met. The algorithm works for any sub-additive cost function, provided one knows how to split a bucket into two buckets of costs at most half (by the sub-additivity property, this is always possible). The proposed adaptation in the streaming context, presented and analyzed in the next section, works only for the freedom cost function.

Theorem 2. *If the cost function is sub-additive, then upon return from* Echap*, the histogram $H = H_{2\lceil (k+1)m \rceil}$ has a cost at most $4C(H^{eq}_{\lceil (k+1)m \rceil})$ and the histogram $H^\star_{2m}$ has a cost at most $\left(1 + \frac{2}{k}\right) C(H_m^{opt})$. Moreover, the algorithm performs at most $O(km \log_2(C(H^{init})/C(H_m^{opt})))$ iterations in line 4.*

A straightforward consequence of Theorem 2 is that Echap requires only $\mathcal{O}(N \log m)$ as computation time to output a guaranteed histogram for any sub-additive cost function, assuming we keep the buckets in a priority queue to speed up processing of line 8 in logarithmic time and that bisection in line 5 can be done in constant time.. This result improves the previously known $\mathcal{O}(mN^2)$ required

```
Algorithm: ECHAP($m$, $k$, $\{v_i\}_i$)
Output A histogram $H^\star_{2m}$ with $2m$ buckets such that $C(H^\star_{2m}) \leq \left(1 + \frac{2}{k}\right) C(H^{\mathrm{opt}}_m)$
 1: Initialize $H = H^{\mathrm{init}}_{2\lceil (k+1)m \rceil}$, with $2\lceil (k+1)m \rceil$ buckets, arbitrarily.
 2: Let $M$ such that $c(B_M) = \max_i\{c(B_i)\}$.
 3: Let $\mu$ such that $c(B_\mu \cup B_{\mu+1}) = \min_i\{c(B_i \cup B_{i+1})\}$.
 4: while $c(B_M) > 2c(B_\mu \cup B_{\mu+1})$ do
 5:     Let $\{B_{M_1}, B_{M_2}\}$ be a partition of $B_M$
        with $c(B_{M_1}) \leq c(B_M)/2$ and $c(B_{M_2}) \leq c(B_M)/2$.
 6:     Merge $B_\mu$ and $B_{\mu+1}$.
 7:     Split $B_M$ in $B_{M_1}$ and $B_{M_2}$.
 8:     Update $M$ and $\mu$ as in lines 2 and 3.
 9: end while
10: Return the best $H^\star_{2m}$ constructed with the same boundaries as $H$ by dynamic
    programming [10,13].
```

Fig. 1. The ECHAP algorithm.

to compute the optimal histogram with a dynamic programming algorithm as presented in [13]. ECHAP also improves the memory requirement with a $\mathcal{O}(m)$ memory versus $\mathcal{O}(mN)$ for the dynamic programming algorithm.

Proof: First, let remark that a partition as defined at the line 5 of ECHAP exists because the sub-additivity of the cost function. As in Theorem 1, let us write H^{eq} for short instead of $H^{\mathrm{eq}}_{\lceil (k+1)m \rceil}$.

As soon as the property "$\max_i c(B_i) = c(B_M) \leq 2 \min_i c(B_i \cup B_{i+1})$" is true, then the algorithm exits the while loop of line 4. We claim that $\min_i c(B_i \cup B_{i+1})$ is smaller than the cost of a bucket of H^{eq}. Indeed, since H contains $2\lceil (k+1)m \rceil$ buckets, the Dirichlet principle grants that at least two neighboring buckets of H are entirely contained in the same bucket of H^{eq}. The claim follows from the sub-additivity of the cost function. In particular, for B_M the most expensive bucket of H, we have

$$c(B_M) \leq \frac{2}{\lceil (k+1)m \rceil} C(H^{\mathrm{eq}}).$$

From this, it follows immediately that $C(H) \leq 2\lceil (k+1)m \rceil c(B_M) \leq 4C(H^{\mathrm{eq}})$.

As we did for Corollary 1, we can also consider the histogram H' with $2m$ buckets obtained from H by merging all the buckets contained in a bucket of H^{opt}_m. Those buckets have a total cost of at most $C(H^{\mathrm{opt}}_m)$ by the sub-additivity property. The m other crossing buckets have a cost of at most

$$mc(B_M) \leq m \frac{2}{\lceil (k+1)m \rceil} C(H^{\mathrm{eq}}) \leq \frac{2}{k+1} C(H^{\mathrm{eq}}).$$

By Theorem 1, we have $C(H^{\mathrm{eq}}) \leq \frac{k+1}{k} C(H^{\mathrm{opt}}_m)$. In total, we obtain that $C(H')$, and hence $C(H^\star_{2m})$, is at most $(1 + \frac{2}{k}) C(H^{\mathrm{opt}}_m)$.

Let us now prove that the maximum number of ECHAP iterations in line 4 is at most $2\lceil (k+1)m \rceil \log_2(\lceil (k+1)m \rceil C(H^{\mathrm{init}})/C(H^{\mathrm{eq}}))$, which together with

Theorem 1 will imply the theorem. Remember that by our claim, the bucket created in step 6 has a cost $c(B_\mu \cup B_{\mu+1})$ at most $C(H^{\text{eq}})/\lceil (k+1)m \rceil$. Note that when the biggest bucket is split into two buckets at each step, the respective costs of the created buckets are at most the half of the cost of the original bucket. There are $2\lceil (k+1)m \rceil$ buckets, hence after $2\lceil (k+1)m \rceil$ iterations, we are sure that the highest bucket cost (if still greater than $C(H^{\text{eq}})/\lceil (k+1)m \rceil$) has been halved. The claim follows. $\square$

In the case where the cost function is the freedom cost, the Echap algorithm can be slightly changed in order to obtain a better bound on the total histogram cost. Indeed, for any bucket B, a partition in the middle into B_1 and B_2 yields $s(B) \geq 2(s(B_1) + s(B_2))$. Therefore for any distribution, we have $2C(H_{2m}^{\text{opt}}) \leq C(H_m^{\text{opt}})$. So computing Echap($2m$,$k = 2$,$\{v_i\}_i$) gives $H_{4m}^\star$ with $4m$ buckets and cost at most $2C(H_{2m}^{\text{opt}}) \leq C(H_m^{\text{opt}})$.

3 Online histograms

In this section, we adapt the Echap algorithm for the data stream context in the case of the freedom cost function. The reason we do it for that function is that in that case, the split in line 5 of Echap can be computed easily by splitting a bucket in its spatial median.

For each bucket B, the Sechap algorithm (Streamed Equi-Cost Histogram APproximation) records the boundaries of B and the width $w(B)$. Since the data is given in a data stream, when splitting a bucket we cannot be sure of the amount of data that falls in each half. Hence we keep two bounds for the number of points that fall into B, the number of guaranteed occurrences $w^g(B)$, and the number of uncertain occurrences $w^u(B)$. We define the under-cost $s^-(B)$ of the bucket B by $s^-(B) = l(B)w^g(B)$, and the over-cost by $s^+(B) = l(B)(w^g(B) + w^u(B))$. The Sechap algorithm ensures that $s^-(B) \leq s(B) \leq s^+(B)$.

The Sechap algorithm also requires two dictionaries T_M and T_μ. The dictionary T_M contains all the buckets sorted according to s^-, the under-cost. It will be used to access to the bucket of highest cost. The other dictionary, T_μ, contains all the neighboring buckets (B_i, B_{i+1}) ordered by the over-cost $s^+(B_i \cup B_{i+1}) = (l(B_i) + l(B_{i+1}))(w^g(B_i) + w^u(B_i) + w^g(B_{i+1}) + w^u(B_{i+1}))$. It will be used to access the neighboring buckets of lowest cost. The dictionaries T_M and T_μ can be seem as pessimistic estimation of the highest and lowest bucket costs.

The details of the algorithm can be found in Figure 2. As with Echap the memory space requirement is only $O(m)$. We will now give two guarantees on the behavior of the Sechap algorithm: one guarantee on the accuracy of the cost associated to the histogram returned by Sechap, and one guarantee on the evolution rate of the histogram.

Theorem 3 (SECHAP). *The overcost associated to the histogram returned by the* Sechap *algorithm is an upper bound for the actual cost. The total number of split/merge operations when treating the first n values is less than $m \log_2 n$.*

```
Algorithm: SECHAP($m$, $k$, $\{v_i\}_i$)
 1: Initialize $H$ with the $2\lceil(k+1)m\rceil$ first values, with exactly one bucket per point.
 2: Set $w^g(B) \leftarrow 1$, $w^u(B) \leftarrow 0$ for each bucket $B$.
 3: Initialize $T_M$ and $T_\mu$ according to $H$.
 4: for all incoming $v_i$ do
 5:   $w^g(B) \leftarrow w^g(B) + 1$ for the bucket $B$ with $v_i \in B$
 6:   Update $T_M$ and $T_\mu$, select $B_M = \max(T_M)$ and $B_\mu = \min(T_\mu)$
 7:   if $s^-(B_M) > 2s^+(B_\mu \cup B_{\mu+1})$ then
 8:     Merge $B_\mu$ and $B_{\mu+1}$ into $B^*_\mu$ with
        $l(B^*_\mu) \leftarrow l(B_\mu) + l(B_{\mu+1})$,
        $w^g(B^*_\mu) \leftarrow w^g(B_\mu) + w^g(B_{\mu+1})$,
        $w^u(B^*_\mu) \leftarrow w^u(B_\mu) + w^u(B_{\mu+1})$
 9:     Split $B_M$ into $B_{M_1}$ and $B_{M_2}$ with
        $l(B_{M_1}) \leftarrow l(B_{M_2}) \leftarrow l(B_M)/2$
        $w^g(B_{M_1}) \leftarrow w^g(B_{M_2}) \leftarrow 0$,
        $w^u(B_{M_1}) \leftarrow w^u(B_{M_2}) \leftarrow w^g(B_M) + w^u(B_M)$
10:     Update $T_M$ and $T_\mu$
11:   end if
12: end for
13: Return $H$
```

Fig. 2. The SECHAP algorithm.

Proof: The first part of the theorem can be proved by induction. Assume that the overcost is always an upper bound for the real cost after treating n values. When the $n+1^{th}$ value is treated, if no split/merge occurs then it is clear that the property remains true. If a split/merge occurs, then the value given to the variables w^g and w^u (lines 8 and 9) guarantees that the overcosts of those particular buckets remain upper bounds of the actual costs. Therefore the property is true at the step $n+1$.

In order to show the second part of the theorem, we need to remark that a split operation on a bucket B requires $s^-(B) > 0$ since $s^+ \geq 0$. Therefore, B should verify $w^g(B) > w^u(B)$. This implies that the $w^u(B)$ is at least multiplied by two for each split. We also remark that the merge does not decrease $w^u(B)$. Since $w^g(B)$ could not be increased by more than one for every incoming value, it is clear that the total number of split/merge operations in the first n values is less than $m \log_2 n$. □

4 Experimental results

4.1 Algorithms and datasets

The purpose of these experiments is first to answer a question left open in the previous section: how good in practice are the histograms provided by the SECHAP algorithm? This section provides empirical evaluations of two algorithms: the

EquiError *algorithm*[4] as presented in [7] and the Sechap algorithm introduced here above. When the number of incoming values is less than 10,000 they are compared against the optimal histogram algorithm as defined in [13], which has a time complexity in $O(mN^2)$. (Unfortunately, it is not feasible to compute the optimal cost with more than 10,000 values). Note also that since the output histograms returned by EquiError or Sechap depend on the order of the incoming values, all the results presented in the following have been established on the basis of 1000 random permutations of the two datasets for less than 10,000 values, and 10 random permutations for more than 10,000 values (for lack of time) but will be redone with 1000 permutations for the final version..

In order to have reproducible experiments, we have chosen to present the results obtained with two publicly available numerical datasets from the *UCI Knowledge Discovery in Databases Archive*, as well as one synthetic dataset:

1. A randomly generated attribute with a normal distribution (Gaussian centered at 0 with standard deviation 1).
2. The variable `Elevation` from the `Forest cover type` database[5]. This dataset has been chosen for being an example of a smooth (continuous) distribution.
3. The variable `dst_bytes` from the `KDD Cup 99` database[6] . This dataset has been chosen for being an example of a peaky (discontinuous) distribution.

When we include the optimal cost using the algorithm of [13], the two datasets are restricted to their 10000 first values. Otherwise, all three datasets have approximately 500,000 values.

4.2 Cost optimality study

The first series of results are relative to the costs of the histograms returned by EquiError and Sechap. Those results are gathered in Tables 1 and 2 with m as the number of buckets. The **Sechap** and **EqErr** indicate the corresponding algorithm. In addition, next to the actual cost of the Sechap histogram, we include the over-cost between brackets to indicate the goodness of the guarantee returned by Theorem 3. The unit used to express the costs is such that the cost of the optimal histogram with m buckets is always 1, hence the test was done only on the first 10,000 values of each dataset.

The results include the dispersion cost function but note that neither EquiError nor Sechap have been tuned for this cost function. The two algorithms have been used *as such*. The motivation is to evaluate the empirical behavior of EquiError or Sechap when used to minimize the dispersion cost function.

Both tables show that both EquiError and Sechap provide good approximations of the optimal histograms. In particular, Sechap is as good as EquiError in case of a smooth distribution, but slightly outperformed by

[4] The open source implementation of the *Equi-Error algorithm* available at `http://www-dsg.stanford.edu/MichaelGreenwald.html` has been used for the experiments.

[5] http://kdd.ics.uci.edu/databases/covertype/covertype.html

[6] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

Normal distribution			Forest cover type database			KDD Cup 99 database		
m	**Sechap**	**EqErr**	m	**Sechap**	**EqErr**	m	**Sechap**	**EqErr**
8	1.135 (1.157)	1.248	8	1.135 (1.157)	1.248	8	1.135 (1.157)	1.248
16	1.117 (1.148)	1.181	16	1.117 (1.148)	1.180	16	1.117 (1.148)	1.181
32	1.105 (1.147)	1.139	32	1.105 (1.148)	1.138	32	1.105 (1.148)	1.139
64	1.121 (1.182)	1.131	64	1.121 (1.183)	1.131	64	1.121 (1.183)	1.131
128	1.161 (1.255)	1.156	128	1.161 (1.255)	1.156	128	1.161 (1.255)	1.156
256	1.240 (1.384)	1.229	256	1.240 (1.385)	1.229	256	1.240 (1.385)	1.229
512	1.393 (1.623)	1.358	512	1.393 (1.623)	1.358	512	1.393 (1.623)	1.358

Table 1. Cost results for the freedom cost function.

Normal distribution			Forest cover type database			KDD Cup 99 database		
m	**Sechap**	**EqErr**	m	**Sechap**	**EqErr**	m	**Sechap**	**EqErr**
8	1.041 (5.168)	1.041	8	1.014 (4.858)	0.966	8	1.071 (5.697)	1.115
16	1.062 (4.776)	1.051	16	1.053 (4.642)	1.024	16	1.135 (6.207)	1.098
32	1.067 (4.619)	1.067	32	1.066 (4.485)	1.107	32	1.227 (7.508)	1.046
64	1.074 (4.631)	1.067	64	1.068 (4.464)	1.032	64	1.714 (12.221)	1.189
128	1.086 (4.787)	1.093	128	1.067 (4.534)	1.027	128	2.419 (19.218)	1.547
256	1.129 (5.187)	1.124	256	1.086 (4.739)	1.045	256	3.952 (34.599)	2.425
512	1.200 (5.871)	1.205	512	1.130 (5.202)	1.144	512	6.736 (64.458)	5.151

Table 2. Cost results for the dispersion cost function.

EquiError for a more peaky distribution, for both cost functions. Note also that for a peaky distribution and the dispersion cost function, the optimality ratio seems to diverge with a larger number of buckets. For all other cases, it remains close to 1 although the divergence would show for many more buckets.

We also see that the overcost maintained by the algorithm is a good estimate of the actual cost (which is unknown to the algorithm), for the freedom function. The results for the dispersion cost functions are not as good, however, and there we see that the overcost is a direct result of the larger number of mutations, especially for the peaky example distribution of KDD Cup 99 (as shown in the stability study.)

4.3 Sechap cost evolution study

Since we only process the first 10,000 values in the previous tables, we now look at the evolution of the cost as more values are examined for the three data types. In Figure 3, we display the cost for 64 buckets, the results for other numbers of buckets are similar. We display both the actual cost of the histogram (which can be computed from the original values) and the overcost maintained by the algorithm. We notice a linear dependence (an almost perfect one for the smooth distribution) on the cost per element. As in the previous experiment, we also note that the overcost is very close to the cost all throughout, a little bit less so for the peaky distributions.

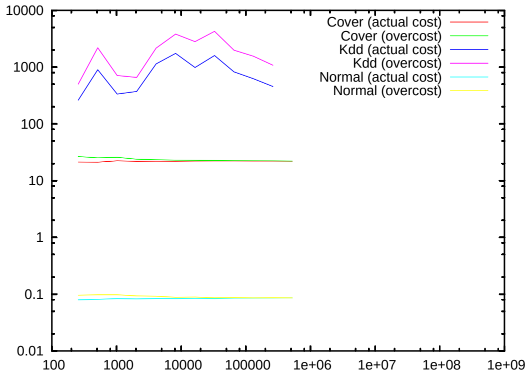

Fig. 3. Cost evolution results *per element* on the full databases streaming in random order, as a function of the number of values seen so far.

4.4 Sechap stability study

The last series of results is relative to evolution rate of the SECHAP histogram while reading every value of the stream. The results are gathered in Figure **??**. The number of buckets is fixed to 64, and columns indicates the number of elements processed, as given in the first line. For each subsequent line, the (abridged) dataset name is followed by values to be read in column labeled x as the number of split/merge that occurs between the step x and $2x$. These results were averaged over 10 permutations, for lack of time, but will be redone with 1000 permutations for the final version.

Figure **??** shows that for all three distributions the convergence is very fast: the number of split/merge decreases rapidly, faster than the bound established in Theorem 3 in the previous section. Note that there are no split/merge until at least m elements have been seen as these are used to initialize the histogram. The number of split/merge is much higher for a peaky distribution, as expected, and the convergence there is slower. Yet, the number of mutations of the histogram is still very small in that case.

One could expect that the sorted distribution should yield the worse convergence since the bucket boundaries would be constantly shifting to adjust to the growing range (as would be the case with EQUIERROR). In Figure 5, we display the convergence of SECHAP when the data arrives on the stream in sorted order. In fact, the convergence is *accelerated* if the data is sorted! It can be explained easily, of course, but we thought it worth pointing out. Note that KDD Cup 99 does not change at all until the last half of the data comes in, since at least half the values are identical to the peak of the distribution.

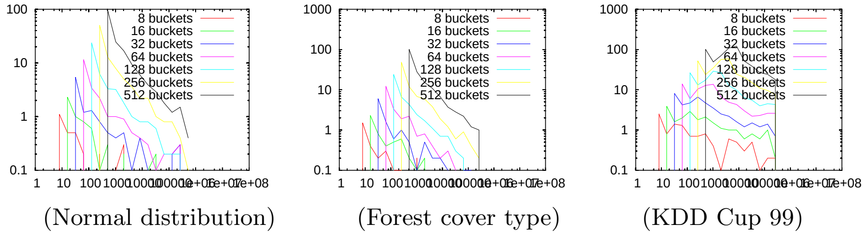

Fig. 4. Evolution results on the full databases streaming in random order.

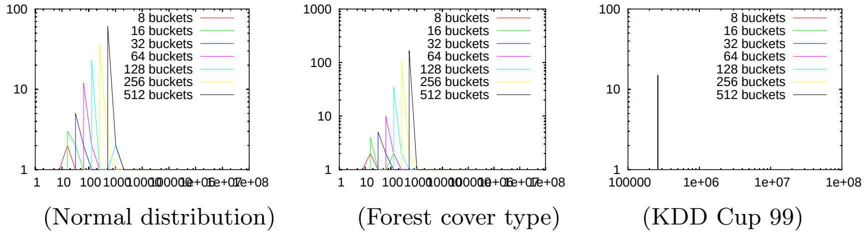

Fig. 5. Evolution results on the full databases streaming in sorted order.

5 Conclusion

The Echap algorithm introduced in this paper is, to our knowledge, the first sub-quadratic algorithm that provides theoretical guaranties on the returned histograms for all sub-additive cost functions. Its online variant Sechap is also the first algorithm that provides theoretical guarantees on the *mutation rate* of the returned histograms. Sechap guarantees that the number of buckets whose boundaries are modified decreases exponentially according the number of incoming values. Empirical evaluations indicate that Sechap is in general slightly outperformed in quality by a previously introduced algorithm called EquiError. Nevertheless, Sechap requires only a $\mathcal{O}(\log(m))$ per element processing time for handling an histogram with m buckets, and so is still interesting in practical situations when one cannot afford the $\mathcal{O}(m)$ time per element required by EquiError.

References

1. Ashraf Aboulnaga and Surajit Chaudhuri. Self-tuning histograms: building histograms without looking at data. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 181–192, 1999.
2. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proc. of the 22nd Annual ACM Symp. on Principles of Databases Systems*, 2002.

3. Phillip B. Gibbons, Yossi Matias, and Viswanath Poosala. Fast incremental maintenance of approximate histograms. In *Proc. 23rd Int. Conf. Very Large Data Bases, VLDB*, pages 466–475. Morgan Kaufmann, 1997.
4. Anna Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proc. 28th Int. Conf. Very Large Data Bases, VLDB*, pages 454–465. Morgan Kaufmann, 2002.
5. Anna Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 389–398. ACM Press, 2002.
6. Anna Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Optimal and approximate computation of summary statistics for range aggregates. In *Symposium on Principles of Database Systems*, 2001.
7. Michael Greenwald. Practical algorithms for self scaling histograms or better than average data collection. *Perform. Eval.*, 27-28:19–40, 1996.
8. Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 58–66. ACM Press, 2001.
9. Sudipto Guha, Piotr Indyk, S. Muthukrishnan, and Martin Strauss. Histogramming data streams with fast per-item processing. In *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 681–692. Springer, 2002.
10. Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-streams and histograms. In *ACM Symposium on Theory of Computing*, pages 471–475, 2001.
11. Yannis E. Ioannidis and Stavros Christodoulakis. Optimal histograms for limiting worst-case error propagation in the size of join results. *TODS*, 18(4):709–748, 1993.
12. Yannis E. Ioannidis and Viswanath Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 233–244. ACM Press, 1995.
13. H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 275–286, 1998.
14. Raj Jain and Imrich Chlamtac. The P^2 algorithm for dynamic calculation of quantiles and histograms without storing observations. pages 1076–1085, October 1985.
15. Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. Optimal histograms for hierarchical range queries (extended abstract). In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 196–204. ACM Press, 2000.
16. Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 251–262. ACM Press, 1999.
17. Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Dynamic maintenance of wavelet-based histograms. In *The VLDB Journal*, pages 101–110, 2000.
18. S. Muthukrishnan. Data streams: Algorithms and applications. invited talk at SODA 2003. Available on request by email to `muthu@research.att.com`.
19. S. Muthukrishnan, Viswanath Poosala, and Torsten Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. *Lecture Notes in Computer Science*, 1540:236–256, 1999.

20. S. Muthukrishnan and Martin Strauss. Rangesum histograms. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 233–242. Society for Industrial and Applied Mathematics, 2003.
21. Viswanath Poosala, Venkatesh Ganti, and Yannis E. Ioannidis. Approximate query answering using histograms. *IEEE Data Engineering Bulletin*, 22(4):5–14, 1999.
22. Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. Improved histograms for selectivity estimation of range predicates. pages 294–305, 1996.