Optimality and Scalability in Lattice Histogram Construction *

Panagiotis Karras
School of Computing
National University of Singapore
lastname@comp.nus.edu.sg

ABSTRACT

The Lattice Histogram is a recently proposed data summarization technique that achieves approximation quality preferable to that of an optimal plain histogram. Like other hierarchical synopsis methods, a lattice histogram (LH) aims to approximate data using a hierarchical structure. Still, this structure is not defined a priori; it consists an *unknown*, not a given, of the problem. Past work has defined the properties that an LH needs to obey and developed general-purpose approximation algorithms for the construction thereof. Still, two major issues remain unaddressed: First, the construction of an *optimal* LH for a given error metric is a problem unsolved to date. Second, the proposed algorithms suffer from too high space and time complexities that render their application in real-world settings problematic. In this paper, we address both these questions, focusing on the case that the target error metric is a *maximum* error metric. Our algorithms treat both the error-bounded LH construction problem, in which the space occupied by an LH is minimized under an error constraint, as well as the classic space-bounded problem. First, we develop a dynamic-programming scheme that detects an optimal LH under a given maximum-error bound. Second, we propose an efficient, practical, greedy algorithm that solves the same problem with much lower time and space requirements. Then, we show how both our algorithms can be applied to the classic space-bounded problem, aiming at minimizing error under a bound on space. Our experimental study with real-world data sets shows the effectiveness of our methods compared to competing summarization techniques. Moreover, our findings show that our greedy heuristic performs almost as well as the optimal solution in terms of accuracy.

1. INTRODUCTION

Data summarization is the problem of representing a large data set by a compact *synopsis* that can be constructed quickly and is characterized by high accuracy. The need for such a representation arises oftentimes in applications such as distributed stream monitoring [39], approximate query answering [2, 36, 22, 5], query optimization [32], OLAP/DSS systems [41], time-series indexing [6], and data mining [31]. An overview of the area is given in [10, 19, 20].

The summarization problem can be formulated in two variants: in the *space-bounded* problem, the goal is to minimize an approximation error over the summarized data within a space budget; in the *error-bounded* problem, the aim is to minimize the space occupied by a synopsis under a given error bound. These problems have conventionally been treated by two major approaches. The former, histogram-based techniques [18, 21, 38, 37, 23, 22, 36, 11, 16, 14, 40, 12], summarize the data by dividing it into consecutive intervals, or *buckets*; typically, a bucket is assigned a single representative value that approximates the data therein; variations to this theme aim to optimize the data representation within a bucket [30, 4, 42]. The latter methods utilize a *predefined* hierarchical tree structure such as that defined by the Haar wavelet decomposition [32, 41, 5, 8, 9, 25, 35, 12, 13] or alternatives that follow a similar pattern [39, 26, 24].

The question of combining the advantages of these two methodologies into a more general structure was posed in [19, 20] and has recently received focused attention. An attempt to insert features of a hierarchical structure into a histogram was made in [4], but its scope was constrained to enhancing individual buckets independently of each other. The concept of a *hierarchical binary histogram* was used in [7], but the hierarchy in this case is a hierarchy of divisions in the multidimensional space; it is not a hierarchy among buckets in which a bucket may *contain* another. Hierarchical histograms based on a *containment* hierarchy were proposed in [3] and [39]. [3] builds buckets using query feedback; hence it does not provide an accuracy guarantee. The techniques of [39] provide accuracy guarantees, but they are constrained by a *predefined* hierarchical tree structure. Most recently, [28] proposed *Lattice Histograms*: hierarchical histograms based on an arbitrary, non-predefined, containment hierarchy. The algorithms proposed by [28] detect a most suitable hierarchy to employ for lattice histogram (LH) construction. Still, despite this progress, these algorithms suffer from two major drawbacks: First, they do not construct a provably *optimal* LH for a given error or space bound. Second, they impose high time and space requirements, limiting their applicability on practical summarization problems.

Thus, two cardinal research questions emerge: (i) a question about the computational feasibility of the *optimal*-LH construction; and (ii) the question of efficiency and scal-

*Work supported by NUS AcRF grant T1 251RES0807.

VLDB '09, August 24-28, 2009, Lyon, France

ability in LH construction algorithms. In this paper, we address both these questions and propose solutions to the ensuing problems. We focus our solutions on the case the target error function or error bound is expressed by means of a *maximum-error* metric; synopses tailored for such metrics prove more robust in accurate data reconstruction [8, 9]. First, we show that it is indeed computationally feasible to construct optimal solutions to both the error-bounded and space-bounded LH construction problems on maximum-error metrics. Second, we develop an efficient and scalable greedy algorithm that constructs LHs in practicable time and space. In addition, we provide experimental evidence to the effect that the latter algorithm achieves almost as high accuracy as the optimal solution, while its performance is far superior in terms of time and space requirements. Besides, this algorithm can operate in a data stream context for the error-bounded LH construction problem.

Outline The remainder of this paper is structured as follows. Section 2 presents background on Lattice Histograms. Section 3 introduces our algorithm that optimally solves the error-bounded LH construction problem with a maximum-error metric. Section 4 develops our greedy algorithms for scalable error-bounded LH construction. In Section 5, we explain how both of these solutions can be utilized for solving the space-bounded LH construction problem. Experimental results on real-world data sets are outlined in Section 6. In Section 7 we discuss our results and in Section 8 we outline our conclusions.

2. BACKGROUND

Research on summarization and data approximation has until recently followed two divergent paradigms. An approximate data representation, and the associated pattern discovery in data, could either follow a segmentation-orientated approach, or a hierarchy-oriented one. In the former case, there is ample freedom on the choice of non-overlapping segments or *buckets* of consecutive data values into which to partition the underlying data and select a single representative for each; still, non-local pattern and relationships are lost. The resulting representation is called a *histogram*. In the latter case, a more global view of the data is gained from the vantage point of a *hierarchical data structure*. Still, local interrelationships may be lost in this approach, as the employed hierarchical structure is predefined and cannot adapt itself to the data at hand.

Both approaches aim to represent an n-size data vector $\mathbf{D} = \langle d_0, d_1, \ldots, d_{n-1} \rangle$ via an approximation $\hat{\mathbf{D}}$ that uses at most B space units, and achieves as low approximation error as possible. Such error is measured by an appropriate *error metric*. Most popular metrics are instances of a *normalized*, weighted Minkowski-norm distance between the original vector and its approximation, $\mathcal{L}_p^w(\hat{\mathbf{D}}, \mathbf{D}) = \left(\sum_i \frac{(w_i |\hat{d}_i - d_i|)^p}{n} \right)^{\frac{1}{p}}$, where $\hat{d}_i$ is the approximate reconstructed value for d_i and w_i an associated weight that signifies the importance of d_i; in the case of a relative-error-based metric, $w_i = \frac{1}{|d_i|}$.

Several attempts have been made towards extending and generalizing the applicability of these approaches. Some have focused on providing general-purpose algorithmic techniques that are applicable on all of them. Thus, [12] has offered a generic model for achieving space-efficiency in synopsis construction algorithms; this model is applicable on several summarization models. Likewise, [29] provided a general-purpose technique for efficiently constructing synopses optimized for a maximum-error metric, as opposed to an aggregate one. Works like [23] and [16] have provided histogram construction algorithms for several error metrics, going beyond the simple heuristics of [21, 38, 37]. Another stream of research has attempted to provide hierarchical summarization algorithms applicable on general error metrics [8, 9, 25, 35, 13, 39, 27]. Still, other works have striven to define histograms in multidimensional spaces, and even build hierarchies over them at that, allowing overlaps and containment relationships between buckets [34, 37, 17, 1, 3]. However, each technique in this genre relies on one or more of the following: (a) heuristics, (b) query feedback, (c) predefined hierarchical patterns and/or grids; thus, they do not provide reliable approximation error guarantees and do not aim to discover patterns in an unbiased fashion.

2.1 The Lattice Histogram

The tradeoff between histogram and hierarchical structures, and the ensuing need for its investigation, were first noted by Ioannidis [19, 20]. On the one hand, a histogram is not constrained by restrictions on bucket *position* or *size*, yet it carries an implicit *locality* assumption. On the other hand, hierarchical structures are advantaged by their capacity to exploit non-local interrelations and gain in compactness therewith; still, they are constrained by the predefined character of the hierarchies they utilize. Indeed, as [28] observed, any *predefined* hierarchy used at the task of summarization imposes an arbitrary constraint on the problem; the most suitable hierarchical pattern for a given data approximation task should not be a *given* of the problem, but rather an *unknown* that needs to be discovered as part of the solution itself. In an attempt to overcome this tradeoff, [28] proposed the Lattice Histogram (LH), a synopsis data structure that aims to discover and utilize any arbitrary hierarchy in the data. Benefited from both a freedom in bucket position and size, as well as a capacity to exploit non-local, hierarchical interrelations, Lattice Histograms can outperform other hierarchical index-based as well as histogram-based techniques in terms of accuracy.

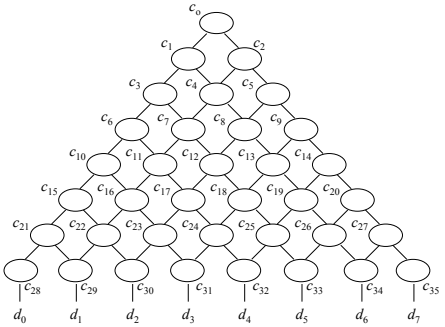

Figure 1: The LH Structure

An LH structure that can approximately represent a data vector of $n = 8$ elements, $\{d_0, \ldots, d_7\}$, is shown in Figure 1. This LH structure includes $N = \frac{n(n+1)}{2} = 36$ nodes, $\{c_0, \ldots, d_{35}\}$, arranged in a lattice of n levels. The ℓ^{th} level (from the top) contains ℓ nodes, each of them *affecting* a range $\mathcal{R}_i$ of length $n - \ell + 1$. The index f_k of the first node c_{f_k} in level k is $f_k = \frac{k(k-1)}{2}$. In effect, the level $\ell(i)$ in which a node c_i resides is $\ell(i) = \lfloor \frac{1+\sqrt{8i+1}}{2} \rfloor$. The *left child* of a node c_i at level ℓ is $c_{i+\ell}$; its *right child* is $c_{i+\ell+1}$. An approximated

data item d_j stands below node c_{N-n+j} in the last level of the lattice. In order to construct an LH-based synopsis of a data vector $\mathbf{D}$ using a space budget of B, we need to assign non-zero values to, i.e. *occupy*, B LH nodes. One of the ranges $\mathcal{R}_i$, $\mathcal{R}_j$ affected by two occupied nodes c_i, c_j is allowed to *contain* the other; however, the two ranges may *not overlap* without having a clear *containment* relationship [28]. For example, if we occupy node c_4 in Figure 1, then it is allowed to occupy any of its *descendant* nodes, as well as nodes that either fully contain, or are disjoint from, range $\mathcal{R}_4$, i.e., nodes c_0, c_1, c_2, c_{28} and c_{35}. The approximation of a data value d_i represented by an LH is constructed as the value of the lowest occupied node affecting d_i, by means of an *interval tree*; hence, data reconstruction requires $O(\log B)$ time (as for other summarization techniques [32, 16, 26, 24]). An *optimal* LH synopsis of $\mathbf{D}$ in space B should achieve the minimum error ϵ^* achievable in B space for the employed error metric. For example, the LH synopsis of the data vector $\mathbf{D}=\{4,3,5,10,12,11,11,4\}$ that minimizes metrics $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_\infty$, occupies the nodes $c_0=4$ and $c_{13}=11$; thus, it approximates the data as $\hat{\mathbf{D}}=\{4,4,4,11,11,11,11,4\}$ and achieves errors $\mathcal{L}_1=0.5$, $\mathcal{L}_2=\sqrt{.5}$, and $\mathcal{L}_\infty=1$. This approximation is better than can be achieved using traditional summarization methods [28].

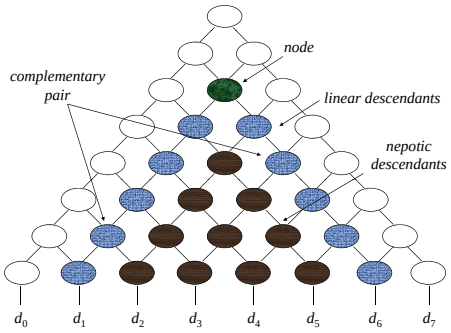

Figure 2: Groups of descendants of an LH node

Plain histograms [21, 23, 16] as well as *Compact Hierarchical Histograms* (CHHs) [39] constitute *special cases* of LHs [28]. According to the terminology in [28], in Figure 1, c_{13} is the *nepot* of c_5 (i.e., common child of its two children), while nodes c_{16} and c_{25} are *complementary with respect to* node c_7 (i.e., their ranges are disjoint and, when combined, form the range of c_7). The descendants of a node are divided into two groups: *linear* and *nepotic* descendants. These groups for a selected node, as well as an example of a pair of *complementary* nodes among the linear descendants, are illustrated in Figure 2. A node c_i in level ℓ has $n-\ell$ pairs of complementary linear descendants, one pair for each level below. As [28] has shown, it is redundant to occupy the linear descendants (hence, symmetrically, the linear ancestors) of an occupied node. Based on this observation, [28] constructed a dynamic-programming approximation scheme for LH construction.

Still, the introduction of the LH structure was accompanied neither by an algorithm capable to construct an *optimal* LH for a given problem, nor by a scalable algorithm suitable for efficient LH construction. Only a high-complexity approximation scheme was provided. Thus, two cardinal questions on the potential of LH-based data approximation remain open. In the next sections, we proceed to provide optimal-quality and heuristic solutions to the LH construction problem. Table 1 summarizes the notation we employ.

Symbol	Meaning
$\mathbf{D}$	Summarized data vector
c_i	Node in Lattice Histogram
$c_{i_L^k}$	Left node in k^{th} complementary pair for c_i
$c_{i_R^k}$	Right node in k^{th} complementary pair for c_i
$\mathcal{R}_i$	Range of data items affected by node c_i
ℓ	Level of the lattice
d_j	Approximated data value
w_j	Weight associated with d_j
T_j	Tolerance interval of d_j
v	Incoming value to c_i
z	Value assigned to c_i
$S(i,v)$	Space required at c_i with v
s_i^*	Minimum value of $S(i,v)$ at c_i
$s_{i_L^k}^*$	Minimum value of $S(i,v)$ at $c_{i_L^k}$
$s_{i_R^k}^*$	Minimum value of $S(i,v)$ at $c_{i_R^k}$
s_i^k	Minimum space using k^{th} complementary pair
$\mathbf{I}_i$	Set of incoming values v yielding s_i^* at c_i
$\mathbf{L}_i^k$	Set of v yielding $s_{i_L^k}^*$ at $c_{i_L^k}$
$\mathbf{R}_i^k$	Set of v yielding $s_{i_R^k}^*$ at $c_{i_R^k}$
$\mathbf{I}_i^k$	Set of v yielding s_i^k with k^{th} pair at c_i
ϵ_i^*	Best achievable error at c_i with s_i^* space
ε^*	Best achievable error in a subinterval of $\mathbf{I}_i$

Table 1: Employed notation

3. OPTIMAL LATTICE HISTOGRAM CONSTRUCTION

We start out our investigation by providing an algorithm that achieves an *optimal* solution to the *error-bounded* LH construction problem. Later, in Section 5, we will show how this solution can be applied towards an equally effective solution to the dual, *space-bounded* problem. We focus on the problems for *maximum* error functions, which provide intuitive *deterministic error guarantees* for independent approximate values [8, 9, 25, 35]. A *maximum-error* bound, as opposed to an *aggregate error* bound, has to be individually satisfied by each approximate value. As we will show, the computation of an optimal solution is rendered tractable and elegant thanks to this property. Moreover, we define a *strong* version of the problem under consideration as follows:

PROBLEM 1. *Given a data vector* $\mathbf{D}$ *and an error bound* ϵ *for a (weighted) maximum-error metric* $\mathcal{L}_\infty^w$*, construct an LH* $\mathbf{L}$ *that produces an approximation* $\hat{\mathbf{D}}$ *of* $\mathbf{D}$*, such that* $\mathcal{L}_\infty^w\left(\left\|\mathbf{D}-\hat{\mathbf{D}}\right\|\right)\leq\epsilon$ *and the number of used LH nodes* B^* *in* $\mathbf{L}$ *is minimized. Moreover, of all* B^**-term LH representations satisfying* ϵ*, select one of minimal actual error* $\epsilon^*\leq\epsilon$*.*

This version of the problem is deemed to be named *strong* due to the *secondary* optimization requirement to choose an error-optimal representation among those that satisfy the given error bound in the minimum space. To solve this problem, we need to determine the minimal set of LH nodes that need to be occupied and the values assigned to them. Thanks to the fact that the error bound needs to be individually satisfied at each approximated data value, we can break down this problem into a separate subproblem for each specific LH node. Thus, for each LH node c_i, we need to determine the minimum amount of nodes that need to be occupied among itself and its descendants. This amount depends on the value v assigned at the lowest occupied ancestor of c_i; we call v the *incoming value* at c_i; this is assumed to be 0 at the root of the lattice. Thus, we need to analyze how this required amount of space depends on v.

3.1 Computation of the Space Function

Let $S(i,v) \in \mathbb{N}$ be the minimum number of occupied LH nodes among c_i and its descendants required to satisfy the given $\mathcal{L}_\infty^w$-error bound ϵ with incoming value v to node c_i in level ℓ. We calculate the solution by means of a bottom-up recursive process that computes the *value range* of $S(i,v)$ on each node c_i. After $S(0,0)$ is established, the exact choices of LH nodes can be traced back for the extraction of the optimal LH. For any i, $S(i,v)$ is defined for every $v \in \mathbb{R}$ and takes values in $\mathbb{N}$. Its value range can be delimited by the following theorem.

THEOREM 1. *Let $s_i^* \in \mathbb{N}$ be the minimum value of $S(i,v)$ on an LH node c_i, $v \in \mathbb{R}$. Then, $\forall v$, $S(i,v) \in \{s_i^*,\ s_i^*+1\}$.*

PROOF. Let $\bar{v}$ be an incoming value to c_i with which the minimum value of $S(i,v)$ is obtained: $\forall v, S(i,v) \geq S(i,\bar{v}) = s_i^*$. If node c_i were assigned a non-zero value z^* in the LH corresponding to $S(i,\bar{v})$, then this very value z^* would make a preferred incoming value to c_i, allowing for an equivalent LH in which node c_i would be unoccupied, i.e. $S(i,z^*) = S(i,\bar{v}) - 1$; this contradicts our assumption that $\bar{v}$ achieves minimum space. Thus, by reductio ad absurdum, it follows that node c_i has to be *unoccupied* for any incoming value $\bar{v}$ with which the optimal $S(i,v)$ is achieved. Then, for any other incoming value $v' \in \mathbb{R}$ to c_i, we may simply assign the value $\bar{v}$ itself to c_i and maintain the same solution as with incoming value $\bar{v}$ among the descendants of c_i otherwise. The assignment of $\bar{v}$ to c_i increases the number of occupied nodes among c_i and its descendants by one unit. In conclusion, $S(i,v)$ obtains exactly two values in the whole domain of $\mathbb{R}$, hence $\forall v \in \mathbb{R}, S(i,v) \in \{s_i^*,\ s_i^*+1\}$. ∎

According to Theorem 1, we can divide all incoming values $v \in \mathbb{R}$ to a node c_i into two groups: (i) the set of values with which the optimal, minimum space $S(i,v) = s_i^*$ is achieved, and (ii) the rest, for which $S(i,v) = s_i^* + 1$. In the following, we examine how the division of the domain of $S(i,v), v \in \mathbb{R}$ into these two sets and the respective value range $\{s_i^*,\ s_i^*+1\}$ can be inductively calculated from raw data.

At the bottom LH level, the division of the domain of $S(i,v)$ is computed from the approximated data values and the given error bound. A data item d_j with an associated error weight w_j defines a *tolerance interval*, $T_j = [d_j - \frac{\epsilon}{w_j}, d_j + \frac{\epsilon}{w_j}]$; any approximate value $\hat{d}_j \in T_j$ satisfies the given error bound ϵ for d_j.

An LH node c_i at the next-to-bottom LH level has to approximate two data items, say left-side item d_L and right-side item d_R; these two items define two tolerance intervals, say $T_L = [a,b]$ and $T_R = [c,d]$. We distinguish two cases, depending on the nature of the intersection of T_L and T_R. If $T_L \cap T_R \neq \emptyset$, then the minimum value of $S(i,v)$ is 0, achieved for any $v \in T_L \cap T_R$. Then, according to Theorem 1, the worst-case value of $S(i,v)$ is 1, obtained for $v \notin T_L \cap T_R$; this value can namely be rectified towards the optimal solution by assigning a single non-zero value to c_i. Otherwise, if $T_L \cap T_R = \emptyset$, then the minimum value of $S(i,v)$ is 1; this optimal case is obtained with any incoming value $v \in T_L \cup T_R$; such incoming values require the assignment of a single non-zero value in one of the two children nodes of c_i in order for the error bound ϵ to be satisfied. In this case, according to Theorem 1, the worst-case value of $S(i,v)$ is 2, required for incoming values $v \notin T_L \cup T_R$.

For the sake of illustration and distinction, both cases of the state of affairs in which $S(i,v) = 1$ at a next-to-bottom-level node c_i with incoming value v, are depicted in Figure 3. The left side of the figure depicts the case in which $T_L \cap T_R = \emptyset$, and $v \in T_L \cup T_R$, hence a single non-zero value assigned to one of the children of c_i suffices to satisfy the error bound ϵ for both approximated data values. In the particular depicted case, such a value is assigned to the right child c_{2i+1}, since the incoming value lies specifically within the tolerance interval of the left child. On the other hand, the right side of the figure presents a case in which the intersection of the two tolerance intervals is not empty, i.e., $T_L \cap T_R \neq \emptyset$, yet the incoming value v does not belong to this intersection, i.e., $v \notin T_L \cap T_R$. In this case, a single non-zero value $z \in T_L \cap T_R$ assigned to c_i suffices for data approximation within the error bound.

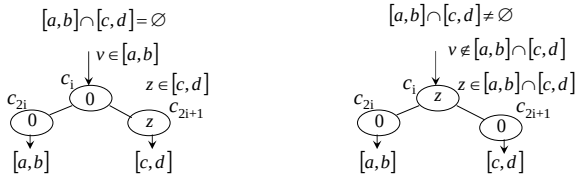

Figure 3: Two cases of $S(i,v) = 1$ in a next-to-bottom-level LH node c_i.

Putting all our observations together, the value of $S(i,v)$ at a next-to-bottom-level LH node c_i is expressed as:

$$S(i,v) = \begin{cases} 0, & T_L \cap T_R \neq \emptyset \ \wedge \ v \in T_L \cap T_R \\ \hline 1, \vee & T_L \cap T_R \neq \emptyset \ \wedge \ v \notin T_L \cap T_R \\ & T_L \cap T_R = \emptyset \ \wedge \ v \in T_L \cup T_R \\ \hline 2, & T_L \cap T_R = \emptyset \ \wedge \ v \notin T_L \cup T_R \end{cases} \quad (1)$$

From Equation 1 we deduce that, in order to fully represent the division of the domain of $S(i,v), v \in \mathbb{R}$ at a next-to-bottom-level LH node c_i, we only need to store the set $\mathbf{I}_i$ such that $v \in \mathbf{I}_i \Leftrightarrow S(i,v) = s_i^*$, where s_i^* is the minimum value of $S(i,v)$ at c_i. Thus, if $T_L \cap T_R \neq \emptyset$, then $\mathbf{I}_i = T_L \cap T_R$ and $s_i^* = 0$; otherwise, if $T_L \cap T_R = \emptyset$, then $\mathbf{I}_i = T_L \cup T_R$ and $s_i^* = 1$. These relations are collectively expressed as follows:

$$\mathbf{I}_i = \begin{cases} T_L \cap T_R, & T_L \cap T_R \neq \emptyset \\ T_L \cup T_R, & T_L \cap T_R = \emptyset \end{cases} \quad (2)$$

$$s_i^* = \begin{cases} 0, & T_L \cap T_R \neq \emptyset \\ 1, & T_L \cap T_R = \emptyset \end{cases} \quad (3)$$

In effect, $\mathbf{I}_i$ is expressed as a union of at most two distinct v-value intervals. We can store this union as an *ordered* series of intervals. Henceforward, we use the term *series of intervals* to distinguish this storage method from a mathematical union. The storage of this series suffices for the full representation of the value domain of $S(i,v)$; it is namely inferred that, if $v \notin \mathbf{I}_i$, then $S(i,v) = s_i^* + 1$.

For a node c_i at a subsequent, higher LH level ℓ, we need to divide the problem among one pair of complementary linear descendants of c_i. The optimal pair to choose itself depends on the incoming value v. Let $c_{i_L^k}$ and $c_{i_R^k}$ be the k^{th} pair of complementary linear descendant nodes of c_i, on which the respective minima of the $S(i,v)$ function are $s_{i_L^k}^*$ and $s_{i_R^k}^*$. Furthermore, let $\mathbf{L}_i^k = \bigcup_{j=1}^{l} L_j^k$, $\mathbf{R}_i^k = \bigcup_{j=1}^{m} R_j^\ell$ be the corresponding subsets of the domain of $v \in \mathbb{R}$ (i.e., mathematical unions of continuous intervals L_j^k and R_i^k,

stored as ordered series of intervals) in which these minima of $S(i,v)$ are achieved. Finally, let $\mathbf{I}_i^k$ be the subset (union of intervals) of the domain of $v \in \mathbb{R}$ in which the minimum-space solution using this k^{th} pair of complementary linear descendants of c_i is achieved, and s_i^k be this minimum space. Then, the actual form of $\mathbf{I}_i^k$ depends on whether $\mathbf{L}_i^k$ and $\mathbf{R}_i^k$ have a non-empty intersection or not. That is, if the intersection of $\mathbf{L}_i^k$ and $\mathbf{R}_i^k$ is non-empty, then $\mathbf{I}_i^k$ is this intersection itself. Otherwise, $\mathbf{I}_i^k$ is the union of $\mathbf{L}_i^k$ and $\mathbf{R}_i^k$, again stored as an ordered series of intervals. Thus, by analogy to, and extension of, the state of affairs at the the next-to-bottom LH level, $\mathbf{I}_i^k$ is expressed as follows.

$$\mathbf{I}_i^k = \begin{cases} \mathbf{L}_i^k \cap \mathbf{R}_i^k, & \mathbf{L}_i^k \cap \mathbf{R}_i^k \neq \emptyset \\ \mathbf{L}_i^k \cup \mathbf{R}_i^k, & \mathbf{L}_i^k \cap \mathbf{R}_i^k = \emptyset \end{cases} \quad (4)$$

In more detail, in the former case, incoming values $v \in \mathbf{L}_i^k \cap \mathbf{R}_i^k$ achieve a minimum-space solution under the scope of both nodes in the k^{th} pair of complementary linear descendants of c_i. Thus, such values achieve the minimum space $s^*_{i^k_L} + s^*_{i^k_R}$ for node c_i itself. In the latter case, any value $v \in \mathbf{L}_i^k \cup \mathbf{R}_i^k$ can achieve an optimal-space solution under the scope of one of the two complementary nodes (i.e., the node to whose series of intervals it particularly belongs), and requires one additional space unit at the other complementary node. Thus, s_i^k is expressed as follows.

$$s_i^k = \begin{cases} s^*_{i^k_L} + s^*_{i^k_R}, & \mathbf{L}_i^k \cap \mathbf{R}_i^k \neq \emptyset \\ s^*_{i^k_L} + s^*_{i^k_R} + 1, & \mathbf{L}_i^k \cap \mathbf{R}_i^k = \emptyset \end{cases} \quad (5)$$

We emphasize that Equations 4 and 5 also cover the next-to-bottom case, in which there is only a single pair to be examined. Thus, Eq. 2 and 3 are special cases of Eq. 4 and 5, respectively. These are conceptually the end-cases of the recursive computation of optimal spaces and the subsets of value space within such optimal spaces are achieved.

We have now shown how to calculate the optimal space achieved with a certain pair of complementary linear descendants of c_i and determine the subset of the incoming value domain where it is achieved. Still, in order to derive the overall solution at c_i itself, we need to concatenate the solutions from all linear descendant pairs. Thus, let $S(i) = s_i^*$ be the minimum number of occupied LH nodes among c_i and its descendants required to satisfy the given $\mathcal{L}_\infty^w$-error bound ϵ, i.e., the minimum value of $S(i,v), v \in \mathbb{R}$. Then $S(i) = \min_k \{s_i^k\}$. Consequently, our algorithm needs to search over all values of k, i.e. over all pairs $c_{i^k_L}, c_{i^k_R}$ of complementary linear descendants of c_i, and identify the set $\mathcal{L}^*$ of those pair-values that achieve this optimal-space solution, $\mathcal{L}^* = \text{argmin}_k \{s_i^k\}$. For each such $k \in \mathcal{L}^*$, $\mathbf{I}_i^k$ is established, and the union of these sets, $\mathbf{I}_i = \bigcup_{k \in \mathcal{L}^*} \mathbf{I}_i^k$ is calculated. This calculation requires k linear merge operations on the stored ordered series of intervals. The optimal solution $S(i)$ is achieved for $v \in \mathbf{I}_i$. For $v \notin \mathbf{I}_i$, a solution of $S(i)+1$ space can be achieved. Thus, the function $S(i,v)$ is succinctly expressed as follows.

$$S(i,v) = \begin{cases} S(i), & v \in \mathbf{I}_i \\ S(i)+1, & v \notin \mathbf{I}_i \end{cases} \quad (6)$$

In effect, the fact that $\mathbf{I}_i$ is, in the general case, a *union of intervals* has been established by induction. We have namely shown that it is so in the bottom-level and next-to-bottom-level cases. Then, we have also shown that, if $\mathbf{I}_j$ is expressed as a union of intervals for all linear descendants c_j of a node c_i, then $\mathbf{I}_i$ is so expressed for c_i as well.

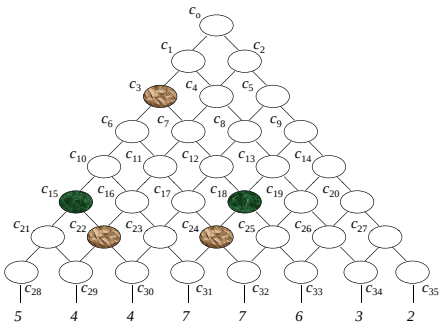

Figure 4: Selection of descendants pair

The example data set and LH in Figure 4 illustrates how the investigation of all complementary pairs of linear descendants guarantees the detection of the optimal-space solution $S(i)$ and the correct representation of the value range of $S(i,v)$. Assume we wish to calculate the value range of $S(i,v)$ at node c_3 under the maximum absolute error bound $\epsilon = 1$. Then, the examination of all complementary pairs of linear descendants results in the outcome $S(3) = s_3^* = 1$. This optimal space is achieved with an incoming value $v \in [4,5] \cup [6,7]$ at c_3. With such an incoming value, the problem at c_3 can be divided among the pair of complementary linear descendants c_{15} and c_{18}. Thus, the data in $\mathcal{R}_3$ can be approximated using only one occupied node beneath c_3, namely either $c_{15} = 4.5$, or $c_{18} = 6.5$ (colored in the figure), depending on whether the actual incoming value belongs to the subinterval $[4,5]$ or $[6,7]$. Otherwise, with an incoming value $v \notin [4,5] \cup [6,7]$ at c_3, both of the nodes c_{15} and c_{18} must be used, covering the whole of $\mathcal{R}_3$. Thus, the assignment of a value at c_3 itself is unnecessary. Naturally, when $v \notin [4,5] \cup [6,7]$, one of the values assigned at c_{15} and c_{18} can be equivalently assigned at c_3 itself; still, for the sake of clarity, we use the assignment at the lowest level. An assignment at c_3 itself would be strictly required if the intersection of the intervals defined for an optimal-space pair of complementary linear descendants had been non-empty, and the incoming value did not belong to this intersection. In such a case, (at least) the two values at the edges of $\mathcal{R}_3$, namely 5 and 6 under c_{28} and c_{33}, would be directly approximated by the value assigned at c_3. Incidentally, it is indeed possible to approximate both c_{28} and c_{33}, for example by assigning the value $= 5.5$ at c_3. However, this option would require occupying two more nodes, namely $c_{22} = 4$ and $c_{24} = 7$ (the three nodes composing this solution are colored in the figure). In other words, such a solution would be suboptimal. Thus, the investigation of complementary linear descendant pairs correctly avoids this option. Had such a solution, using an assignment at c_3, been indeed optimal, then it would be detected by the algorithm.

We have now elaborated on the computation of the function of space and its associated value ranges; we proceed to discuss the particular way these partial solutions are stored and secondary error optimization is achieved.

3.2 Storage and Error Optimization

As we have discussed, in the general case, $\mathbf{I}_i$ is stored as an ordered series of subintervals, mathematically denoted as $\mathbf{I}_i = \cup_j I_j$. Each of these subintervals $I_j = [m, M]$ arises, in the general case, from a single, or the intersection of two

or more, *tolerance intervals* of the form $[d_i - \frac{\epsilon}{w_i}, d_i + \frac{\epsilon}{w_i}]$. In order to facilitate our solution of the *strong* version of the problem, we need to store each subinterval $[m, M]$ along with some accompanying track-keeping information. This information includes:

1. In the case of maximum absolute error, the data items d_m, d_M that have defined the subinterval's limits m, M; that is, the minimum and maximum value among the data approximated by c_i, whose tolerance intervals intersection has produced $[m, M]$. In the general case of a *weighted* maximum-error metric this information should include the full set of data items that define the *error function* in the subinterval, as the analysis in [16, 15] specifies. These are the items such that their approximation error for some representative value(s) $v \in [m, M]$ is not exceeded by that of any other item, hence they define the maximum error for those v (see the thorough analysis in [15] for details).

2. The optimal value $v^* \in [m, M]$ that minimizes the target error metric. In the general case, for a *weighted* maximum-error metric (and the special case of maximum *relative* error), v^* is calculated according to the analysis in [16, 15]. In the case of the maximum absolute error, $v^* = \frac{d_m + d_M}{2}$.

3. The optimal error value ε^* that can be achieved by using a value $v \in [m, M]$. This optimal error is *not always* simply the error achieved with v^*; it is so only in the base case at the bottom level. Thereafter, when the subinterval $[m, M]$ participates in a union $\mathbf{L}_i^k \cup \mathbf{R}_i^k$ for a pair of complementary linear descendants, then the best achievable error at the *other* complementary node needs to be taken into consideration at the computation of ε^*. Thus, when we have to construct $\mathbf{I}_i^k$ as a union, $\mathbf{I}_i^k = \mathbf{L}_i^k \cup \mathbf{R}_i^k$, if the previously computed optimal error ε^* for a subinterval $[m, M] = L_j^k \in \mathbf{L}_i^k$ is *lower* than the *best* achievable error $\epsilon^*_{i_R^k}$ among all subintervals in $\mathbf{R}_i^k$, then we attribute this $\epsilon^*_{i_R^k}$ as ε^* value for $[m, M]$ as a member of $\mathbf{I}_i^k$; otherwise, we maintain the previously computed value of ε^*. The same point also holds vice-versa, reversing the roles of $\mathbf{L}_i^k$ and $\mathbf{R}_i^k$ in the preceding statement. We have to do so, in order to ensure that the left member of the pair will take into consideration what error is incurred at the right pair, and vice-versa. This necessity arises from the fact that the two nodes in the pair forming a union are independent of each other as far as data approximation is concerned. On the other hand, such an issue does not arise in the case we need to construct $\mathbf{I}_i^k$ as an intersection, $\mathbf{I}_i^k = \mathbf{L}_i^k \cap \mathbf{R}_i^k \neq \emptyset$. However, a related matter appears in this case also: The *inherited*, previously computed, optimal error ε^* for each subinterval in $\mathbf{L}_i^k$ and $\mathbf{R}_i^k$ needs to be taken into consideration. Such an *inherited* error may be derived from the computation of $\mathbf{L}_i^k$ or $\mathbf{R}_i^k$ as a *union* itself. Thus, in this case, ε^* is computed as the *maximum* among three errors: the errors *inherited* from the two intersecting subintervals that form $[m, M]$, and the error achieved with v^* in $[m, M]$ itself. The latter is calculated according to the thorough analysis in [16, 15]. In the case of maximum absolute error, it is $\left|\frac{d_m - d_M}{2}\right|$.

4. The value k denoting the k^{th} pair of complementary linear descendants of c_i from which the subinterval $[m, M]$ in $\mathbf{I}_i = \bigcup_{k \in \mathcal{L}^*} \mathbf{I}_i^k$ itself arises. This value is necessary for backtracking the choices in order to construct the derived optimal Lattice Histrogram at the end of the computation.

We now elaborate on the computation of $\mathbf{I}_i$ by merging k series of subintervals, $\mathbf{I}_i = \bigcup_{k \in \mathcal{L}^*} \mathbf{I}_i^k$. Several subintervals encountered in this merge operation may overlap, as each of them is arising from a different pair of complementary linear descendants. Still, their error properties may differ. Thus, in such a case, the subinterval of *lower* optimal achievable error *subsumes* its peer subinterval in their domain where they overlap. Still, the higher-error subinterval remains valid in the non-overlapping region. In effect, $\mathbf{I}_i$ is eventually expressed in a more elaborate fashion than a simple series of disjoint intervals. It may namely contain *contiguous* intervals, each of them holding different ε^* and k attributes. We emphasize that the data items d_m, d_M, defining the *limits* m, M of an interval truncated due to such overlap, and these limits themselves, remain *unchanged* for the purposes of calculating v^* and ε^*.This observation is crucial for ensuring correct error and value calculations in subsequent steps. We re-iterate that this discussion pertains to the solution for the *strong* version of the problem only. A simpler version of the problem, in which we would not require that the error achieved within the optimal space be minimized itself, would not necessitate such an elaborate computation.

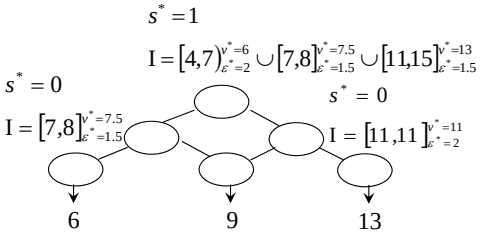

Figure 5: Calculation Example

Figure 5 provides an illustration on how partial information is kept with the computed subintervals, how these subintervals are merged along several pairs of complementary linear descendants, and how the computed information is *inherited* at higher levels of the lattice. We wish to compute an error-bounded LH for the top node in the figure under the maximum absolute error bound $\epsilon = 2$. For the sake of clarity, the computations for nodes at the bottom level are not shown in the figure. At this level, the three approximated data values, 6, 9, 13, generate the tolerance intervals $[4, 8]$, $[7, 11]$, and $[11, 15]$ respectively, each with optimal achievable error $\varepsilon^* = 0$ and preferred incoming value v^* the data value itself. At subsequent levels, each subinterval in a series is depicted along with its associated optimal value v^* as superscript and achievable error ε^* as subscript; the optimal achieved space s^* is also shown for each node. Thus, the left node in level 2 accepts incoming values within the non-empty intersection $[7, 8] = [4, 8] \cap [7, 11]$ of the tolerance intervals in its two children and achieves minimal space $s^* = 0$. The optimal value in $[7, 8]$ is $v^* = \frac{9+6}{2} = 7.5$, while the achieved error is $\varepsilon^* = \frac{9-6}{2} = 1.5$. Similarly, the right node in level 2 accepts the single incoming value 11, the single element in the intersection $[7, 11] \cap [11, 13]$ of the

two relevant tolerance intervals, and achieves $s^* = 0$ with $v^* = \frac{13+9}{2} = 11$ and $\varepsilon^* = \frac{13-9}{2} = 2$.

The top node c_0 in the figure needs to compute the solution for each of two pairs of complementary linear descendants, and merge them. For the first pair ($k = 1$), we need to check whether $[7, 8]$ and $[11, 15]$ intersect. Since they do not, we generate the union $[7, 8] \cup [11, 15]$ and compute the optimal space $s_0^1 = 1$ for this pair. In this union, the subinterval $[11, 15]$ is assigned the new optimal error value $\varepsilon^* = 1.5$, i.e., the best possible error effected by its *complementary* node in the union. This is due to the fact that, even if we assume a convenient incoming value $v \in [11, 15]$ at the top, we still have to consider the big picture on both sides. Similarly, for the second pair of complementary linear descendants ($k = 2$), we establish that $[4, 8]$ and $[11, 11]$ do not intersect either. Thus, we build the union $[4, 8] \cup [11, 11]$ and compute $s_0^2 = 1$. Again, in this union, the subinterval $[4, 8]$ assumes the new optimal error value $\varepsilon^* = 2$, since this is the best value at its complementary node in the union.

We observe that both pairs of complementary linear descendants have achieved equal space, hence $s_0^* = 1$. In effect, we now proceed to properly merge the series of intervals for these two pairs. This operation encounters a double overlap among them. Subinterval $[7, 8]$ for the first pair overlaps with $[4, 8]$ for the second, while $[11, 15]$ overlaps with $[11, 11]$ too. Thus, the subinterval of best error should prevail in each case. Since the achievable error for $[7, 8]$ is 1.5, which is less than the $\varepsilon^* = 2$ for subinterval $[4, 8]$, the latter is truncated to $[4, 7)$. Still, it maintains the same value $v^* = 6$, as well as the value $k = 2$, denoting the pair it has arisen from. This is followed by subinterval $[7, 8]$. Then, in the overlap between $[11, 15]$ and $[11, 11]$, the subinterval $[11, 15]$ with $\varepsilon = 1.5$ prevails and completely subsumes its peer. In conclusion, we obtain the series of intervals depicted at the top of Figure 5. The calculation proceeds in a similar fashion at subsequent nodes and levels.

3.3 Construction

Eventually, the minimum-space result is calculated as the value of $S(0, 0)$. The minimum error that can be achieved with a Lattice Histogram of that space is also calculated thanks to the performed track-keeping outlined in Section 3.2; hence the secondary error optimization is achieved. However, apart from deriving the optimal space and error result, we need to actually construct the Lattice Histogram that achieves these optimal results. In order to retrieve this optimal LH, we have to trace through the choices made at each node after the solution at the top is established. The construction commences with the root node and assumes incoming value $v = 0$. Thereafter, whenever the incoming value we end up with for a node c_i is space-suboptimal, we have to assign a value v_i^*, that achieves a minimal error ϵ_i^* compared to all options at c_i. This value v_i^* is selected as the optimal value v^* for a subinterval in $\mathbf{I}_i$ that achieves the optimal error ϵ_i^*. If $\mathbf{I}_i$ at the examined node c_i is expressed as a single interval, then we can deduce that this has been derived from an *intersection* of subsets among a pair of complementary linear descendants. In this case, if $v \notin \mathbf{I}_i$, then we definitely have to assign v_i^* at c_i itself. In effect, we achieve the solution of $S(i) + 1$ space at c_i. Otherwise, if $v \in \mathbf{I}_i$, then we can simply proceed to the selected pair of complementary linear descendants for the interval to which v belongs. On the other hand, if $\mathbf{I}_i$ is expressed as a series of more than one subintervals $\bigcup_j I_j$, then we can deduce that this has been derived from a union of disjoint subintervals. Then, for $v \notin \mathbf{I}_i$, we can achieve the $S(i) + 1$ solution by simply proceeding to the selected pair of complementary linear descendants with which the optimal space is achieved. Appropriate values are then assigned to the descendants themselves, as required. In all cases, we proceed the LH construction with the appropriate pair of linear descendants of c_i.

3.4 Complexity Analysis

The space required to store the set (ordered series of intervals) $\mathbf{I}_i$ representing the optimal-space domain of $S(i, v)$ for a node c_i grows with the LH level in which c_i resides. In a next-to-bottom-level node c_i, two distinct value intervals need to be stored in the worst case, one for each approximated data value. In general, in the worst case, a node c_i at level k, counting from the bottom, achieves optimal space in a union of as many intervals as the data items under its scope, i.e. k intervals. Therefore, $O(k)$ space is required in order to store $\mathbf{I}_i$ in the worst case. However, in practice this space is significantly pruned, in a data-driven fashion. Whenever an intersection of the series of intervals at a two complementary nodes is achieved, only those subintervals that participate in this intersection survive for computations at subsequent levels. Problems with a large number of intersecting intervals are more interesting, since it is exactly such intersections that exploit the hierarchical potential of the LH structure. Thus, the more interesting a data summarization problem is, the more pruning it also achieves in the LH computation.

Nevertheless, the intersection of $\mathbf{L}_i^\ell$ and $\mathbf{R}_i^\ell$ has to be checked, and their union or intersection computed, for each of the $k - 1$ pairs of complementary linear descendants of c_i. The series of intervals are always maintained in sorted order, hence these union/intersect operations are performed in $O(k)$ time. Thus, $O(k^2)$ processing time is required for each node in level k. The are $n-k+1$ nodes in level k, hence the total time complexity is $O\left(\sum_{k=1}^{n} k^2(n-k)\right) = O(n^4)$. Similarly, the space is $O\left(\sum_{k=1}^{n} k(n-k)\right) = O(n^3)$. We expect the runtime to be closer to cubic in practice, thanks to the pruning achieved whenever series of intervals intersect.

4. GREEDY LATTICE HISTOGRAM CONSTRUCTION

The algorithm of Section 3 achieves optimal quality for any given maximum-error metric. Still, its time- and space-complexity is impracticable for very large data sets. Therefore, in this section, we develop a scalable, greedy algorithm, that constructs an *error-bounded* LH, given any weighted maximum-error metric and an associated error bound ϵ.

Our algorithm only needs to perform a linear scan of the data, in a *streaming* fashion. During this scan, it tries to build an LH that satisfies the given error bound with as few LH nodes as possible. In order to achieve this purpose, in includes a *retrospective* as well as a *forward-looking* operation. In the forward-looking step, the algorithm extends the right boundary of the running k^{th} LH bucket b_k as long as the error bound is satisfiable in b_k, i.e. there exists a representative value v with which an $\mathcal{L}_\infty^w$ error $\leq \epsilon$ can be achieved for b_k. In order to determine this satisfiability we maintain the running set of such values. An encountered

data item d_i with associated weight w_i defines, as in Section 3.1, a *tolerance interval* $I_i = [d_i - \frac{\epsilon}{w_i}, d_i + \frac{\epsilon}{w_i}]$; values in this interval can satisfy ϵ for d_i. The algorithm maintains the intersection $\mathcal{I}_k$ of such intervals for b_k along the scan. When the tolerance interval I_i of the last read data item d_i has null intersection with the current $\mathcal{I}_k$ for the running bucket b_k, then a new bucket boundary is introduced before item d_i.

At this point the *retrospective* step of the algorithm takes over. The algorithm looks back to previously created LH buckets b_j, $j < k$, that are not hierarchically contained by other already created buckets. To facilitate this operation, a *lattice skyline*, i.e., an ordered list of such hierarchically non-contained LH buckets, is maintained. The *nearest* bucket b_ℓ such that I_i has a non-empty intersection with $\mathcal{I}_\ell$, if such exists, is assigned data item d_i. If no such bucket b_ℓ is found, then d_i forms the first item in a new bucket b_{k+1}. Otherwise, bucket b_ℓ is *shifted* to a higher hierarchical level in the lattice, and is henceforward extended further in the forward-looking mode. The choice of the *nearest* bucket b_ℓ that can accommodate d_i is sound, since it greedily minimizes the number of interloping buckets, between b_ℓ and d_i, that are becoming hierarchically contained, hence can not be used in subsequent steps.

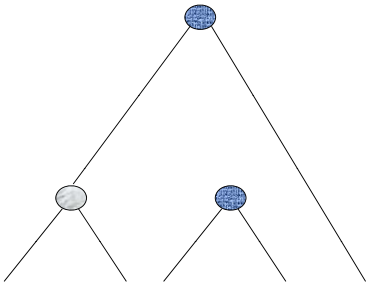

Figure 6: Greedy *shift* operation

Figure 6 illustrates the *shift* operation performed by the greedy algorithm. Assume we are working with a maximum absolute error bound $\epsilon = 2$. Then, the three first items $4, 3, 5$ in the figure can be included in the same bucket b_1, as the intersection of their tolerance intervals is $\mathcal{I}_1 = [2, 6] \cap [1, 5] \cap [3, 7] = [3, 5]$. Still, the fourth item, $d_4 = 9$, defines the tolerance interval $I_4 = [7, 11]$ that has null intersection with $\mathcal{I}_1 = [3, 5]$. Hence, the extension of the first bucket ends there. There are no previously created buckets to be searched in the retrospective step, hence the forward-looking operation resumes with the second bucket b_2. This can include the next three items $9, 10, 11$ that collectively define the tolerance interval $\mathcal{I}_2 = [9, 11]$. Still, b_2 comes to an end when the seventh item, $d_7 = 6$, is encountered, as its tolerance interval $I_7 = [4, 8]$ does not intersect with $[9, 11]$. Still, now the retrospective step succeeds, as $[4, 8]$ intersects with $\mathcal{I}_1 = [3, 5]$. Thus, b_1 is now *shifted* to a higher hierarchical level, assumes the position of a *containing* bucket of b_2, and incorporates d_7. Thus, subsequent retrospective steps will only see b_1 and miss b_2. For the maximum absolute error, the optimal value to be assigned at a bucket b_k is $v_k = \frac{M_k + m_k}{2}$, and the error achieved with it is $\varepsilon_k = \frac{M_k - m_k}{2}$, where m_k (M_k) is the minimum (maximum) value in b_k. In the general case of a *weighted* maximum-error metric, v_k and ε_k are calculated according to the analysis in [16, 15].

In the worst case, this GreedyLH algorithm needs to look back at all previously created buckets for each newly created bucket b_k. Thus, if B is the eventual number of buckets created, or a known upper bound for this number, then GreedyLH requires $O(n + B^2)$ time and $O(n)$ space. Figure 7 illustrates the operation of GreedyLH.

Algorithm GreedyLH(ϵ, B)
Input: error bound ϵ, n-data vector $[d_0, \ldots, d_{n-1}]$
Output: **LH** of no more than B buckets that satisfies ϵ
1. $f = 0$; $i = 0$; $j = 1$; $c = j$; $\mathcal{I}_j = \mathbb{R}$;
2. **while** $(i < n) \wedge (j \leq B)$
3. read d_i;
4. compute I_i;
5. **if** $(\mathcal{I}_c = I_i \cap \mathcal{I}_c \neq \emptyset)$
8. include d_i in bucket b_c;
5. **else**
7. **if** $\exists \ell : (I_i \cap \mathcal{I}_\ell \neq \emptyset)$ // b_ℓ is nearest fit noncontained bucket
8. include d_i in bucket b_ℓ;
7. $c = \ell$; // start extending b_ℓ
5. **else**
7. $j = j + 1$; $c = j$; // resume counter
7. start new bucket b_j with d_i; $\mathcal{I}_j = I_i$
9. $i := i + 1$;
11. **if** $(i < n)$ **return** failed;
12. **else return** created **LH**;

Figure 7: Greedy LH construction algorithm

5. SOLVING THE SPACE-BOUNDED LH CONSTRUCTION PROBLEM

We have now introduced two algorithms for the error-bounded LH construction problem. Still, we are most interested in solving the reverse, space-bounded problem.

PROBLEM 2. *Given a data vector* $\mathbf{D}$ *and a space bound* B*, construct an LH* $\mathbf{L}$ *using no more than* B *nodes, producing an approximation* $\hat{\mathbf{D}}$ *of* $\mathbf{D}$*, so that the (weighted) maximum-error metric* $\mathcal{L}_\infty^w \left(\left\| \mathbf{D} - \hat{\mathbf{D}} \right\| \right)$ *is as low as possible.*

5.1 Testing Error-Optimality

Our approach to this problem is to apply our algorithms for its error-bounded counterpart in a binary search mode. Still, we need some elaboration in order to guarantee the soundness and convergence of our method. Thus, we introduce the following lemma, which defines an *error-optimality* test for the LHs returned by both our error-bounded LH construction algorithms. That is, it defines the conditions under which the *actual* $\mathcal{L}_\infty^w$-error $\bar{\epsilon}$, achieved by the LH returned by the employed algorithm with error bound ϵ, is the *lowest* error that the employed algorithm could satisfy within the same amount of space B^* (or even other, higher values). Hence, for the optimal error-bounded LH algorithm, this test provides a guarantee that the secondary error-optimization problem has been effectively solved.

LEMMA 1. *Let* $\mathbf{L}$ *be the* $\bar{B}$*-bucket Lattice Histogram of* $\mathbf{D}$ *for the* $\mathcal{L}_\infty^w$*-error bound* ϵ *returned by an algorithm for the error-bounded problem, and* $\bar{\epsilon} \leq \epsilon$ *be the actual* $\mathcal{L}_\infty^w$*-error of* $\mathbf{L}$*. Let* $\tilde{\mathbf{L}}$ *be the* $\tilde{B}$*-bucket Lattice Histogram of* $\mathbf{D}$ *returned by the same algorithm running in a different mode, under the constraint* $\mathcal{L}_{\infty,r}^w < \bar{\epsilon}$*. That is, for both our algorithms, all computed intervals are* open *instead of* closed*; thus, error values* less than *but not* equal to *the new bound* $\bar{\epsilon}$ *are allowed. Then,* $\bar{\epsilon}$ *is the minimum* $\mathcal{L}_\infty^w$*-error that the given algorithm can satisfy for* $\mathbf{D}$ *using any number of LH buckets in and only in* $\{\bar{B}, \ldots, \tilde{B} - 1\}$.

PROOF. By definition, the algorithm needs *at least* $\tilde{B}$ space in order to achieve any error bound *less than* $\bar{\epsilon}$. Thus, if $\tilde{B} > \bar{B}$ then any LH of $\mathbf{D}$ with $\mathcal{L}_\infty^w$-error less than $\bar{\epsilon}$ requires more than $\bar{B}$ buckets. Thus, $\bar{\epsilon}$ is the best error the algorithm can satisfy using $\bar{B}$ LH buckets, and indeed any number of LH buckets in the non-empty set $\{\bar{B}, \ldots, \tilde{B}-1\}$. Besides, $\epsilon \geq \bar{\epsilon}$ requires *at least* $\bar{B}$ LH buckets; thus, space less than $\bar{B}$ can only satisfy errors higher than $\bar{\epsilon}$. Likewise, space more than $\tilde{B}-1$ can satisfy errors lower then $\bar{\epsilon}$. ∎

5.2 Main Methodology

We now describe our methodology for the space-bounded LH construction problem under any weighted maximum-error metric, based on Lemma 1.

The starting point of our scheme is the observation that the required space is monotonically non-increasing as the $\mathcal{L}_\infty^w$ -error bound grows. Thus, a binary search process over the domain of error bound ϵ leads to the lowest error an error-bounded LH construction algorithm can satisfy for a given space budget B. This monotonicity holds with the optimal LH construction algorithm; it is also generally true with the greedy algorithm, except for pathological cases, in which a larger error bound may allow for greedy shift operations that spoil subsequent steps. Still, the binary search can still apply in such cases, and result in a local minimum of error in the worst case.

Algorithm IndirectLH(B)
Input: space bound B, n-data vector $[d_0, \ldots, d_{n-1}]$
Output: $\mathcal{L}_\infty^w$-error optimal histogram partitioning $\mathbf{H}$
1. $\epsilon_u = \mathcal{L}_\infty^w$-error of equi-width B-histogram;
2. $e_l = 0$; $e_h = \epsilon_u$;
3. **while** not finished
4. $e_m = (e_h + e_l)/2$;
5. $\bar{B}$ = ErrorboundedLH(e_m);
6. $\bar{\epsilon}$ = actual $\mathcal{L}_\infty^w$-error of returned **LH**; // $\bar{\epsilon} \leq \epsilon$
7. **if** ($\bar{B} \leq B$)
8. $\tilde{B}$ = ErrorboundedLH($< \bar{\epsilon}$); // optimality test
9. **if** ($\tilde{B} > B$)
10. finished := 1; // found best error
11. **else** $e_h = \bar{\epsilon}$;
12. **else if** ($\bar{B} > B$) $e_l = e_m$
13. **return LH**;

Figure 8: Indirect space-bounded LH construction

The operation of the general IndirectLH procedure is shown in Figure 8. The required seed value of the error bound ϵ is obtained by measuring the $\mathcal{L}_\infty^w$-error of an equiwidth B-bucket histogram of $\mathbf{D}$; this error constitutes an upper bound for the $\mathcal{L}_\infty^w$-error achieved using B LH nodes with both our error-bounded LH algorithms. Besides, IndirectLH performs an *optimality test* whenever the tested error bound value ϵ requires exactly B or less LH nodes to be satisfied. When such an optimality test indicates that any error bound ϵ *less* than the already achieved $\bar{\epsilon}$ would require *more* than B LH nodes, the search can safely terminate. Thus, the convergence of the search is reassured. In fact, as OptimalLH achieves secondary optimization of error, the optimality test is redundant with it when OptimalLH requires *exactly* B LH nodes in order to satisfy the tested error bound - in this case the optimal error in B units has already been achieved.

The complexity effect of this binary search operation is an $O(\log \epsilon^*)$ factor in time. The gist of this analysis has appeared in [35, 29, 27]. In effect, the time complexity of our algorithm for space-bounded LH construction is $O(n^4 \log \epsilon^*)$ with OptimalLH, and $O\left((n+B^2)\log \epsilon^*\right)$ with GreedyLH.

Coupled with our OptimalLH algorithm, IndirectLH results into an LH of the minimum $\mathcal{L}_\infty^w$-error ϵ^* that can be achieved in the space budget B, hence achieves the *optimal* solution to the space-bounded Lattice Histogram construction problem for any maximum-error metric. In conclusion, this result overcomes the difficulty of this problem observed in [28].

Having an algorithm for the space-bounded problem under any maximum-error-based metric, we can customize it to work for general error metrics as a heuristic. We first let the algorithm build an LH for a native maximum-error-based metric most closely related to the general error metric at hand. Then, after the B occupied nodes are established, we alter their assigned values so as to render them optimal for the error metric at hand in the data subset they affect. In Section 6 we include a comparison of this general-error extension of our algorithms against conventional techniques.

5.3 Ascertaining Convergence

We emphasize that IndirectLH shall always converge to the optimal error result, even if that error is a recurring number. Namely, even if it has to calculate that error by means of the binary-search calculations, it still approximates that result with as much decimal precision as the machine allows. The same limitation holds for any *exact* algorithm.

Furthermore, the convergence of IndirectLH coupled with OptimalLH is not brought about by means of the binary search computations per se. It is achieved even more rigorously, by means of the secondary error optimization. Thus, the binary search terminates when it reaches an error bound ϵ that necessitates the same space budget as the optimal error within the given space budget B, i.e., either B itself, or a budget $\bar{B} < B$ that achieves the same optimal error as B. In effect, any error bound within an appropriate *interval* is sufficient for termination. Let $f : \mathbb{R}_+ \rightarrow \mathbb{N}$ be the non-increasing function that returns the least space $B = f(\epsilon) \in \mathbb{N}$ required to satisfy the error bound $\epsilon \in \mathbb{R}_+$; f defines the concept of the *error interval* of B, i.e., the domain of error values that can be satisfied in B but no less space: $[\epsilon_{min}^B, \epsilon_{min}^{<B})$, where ϵ_{min}^B is the minimum error that can be achieved using B LH nodes and $\epsilon_{min}^{<B}$ is the minimum error achievable in *less* than B LH nodes. Thus, B is the minimum budget of LH nodes required to achieve error bound ϵ *if and only if* $\epsilon \in [\epsilon_{min}^B, \epsilon_{min}^{<B})$. A B value has a *null* error interval if the optimal error achieved with B can also be achieved in less than B space. We now prove the convergence of IndirectLH coupled with OptimalLH more robustly, *without* reference to the precision of the machine it runs on.

THEOREM 2. IndirectLH *converges to the optimal error in* $O\left(\log \frac{\mathcal{E}}{r_{B^*}}\right)$ *iterations, where* $\mathcal{E}$ *is the seed error bound of the binary search,* $B^* \leq B$ *be the minimum space in which the same minimum error* ϵ^* *as in the given space budget* B *can be achieved in a given summarization problem, and* $r_{B^*} = \epsilon_{min}^{<B^*} - \epsilon_{min}^{B^*}$ *is the size of the error interval of* B^*.

PROOF. As soon as the binary search reaches a value of the error bound ϵ in the error interval of B^*, the actual minimum error ϵ^* in space B^* is calculated, and the optimality test is positive. In effect, the binary search does not need proceed to higher precision; thus, IndirectLH converges to the optimal error result in $O\left(\log \frac{\mathcal{E}}{r_{B^*}}\right)$ iterations. ∎

Conclusively, IndirectLH would still converge to the *optimal* error result even on an *ideal* machine that allowed for *infinite* decimal-point (or binary-point) precision.

6. EXPERIMENTAL EVALUATION

We are primarily interested to assess our greedy algorithm in terms of accuracy. We possess an ideal accuracy benchmark, provided by the OptimalLattice algorithm. Furthermore, we included other popular data summarization techniques in our comparison, as follows:

- **Plain Histogram** The optimal histogram algorithms of [23, 16]. These provide an upper bound to the quality of approximate histograms [21, 38, 37, 11, 14, 40].
- **CHH** The winning greedy heuristic for a Compact Hierarchical Histogram [39]. This technique initially computes an *overlapping* partitioning in the fixed binary hierarchy defined by a CHH. In such an overlapping partitioning, the value assigned on a CHH node is the optimal value for the complete data interval under that node's scope, regardless of the data set the node eventually approximates. Then, it modifies the values assigned to the selected nodes so as to better approximate the data they actually affect. The CHH is a special case of a Lattice Histogram.
- **Haar$^+$** The synopsis construction model based on the Haar$^+$ tree [26]. This model supersedes previous wavelet-based techniques [9, 13]. Besides, the CHH can also be seen as a special case of a Haar$^+$ tree.
- **Optimal Lattice** Our algorithm for Optimal LH construction described in Sections 3 and 5.
- **Greedy Lattice** Our Greedy algorithm for LH construction, as in Sections 4 and 5.

In order to extend the scope of our comparison, we have applied our algorithms not only to maximum-error-based metrics for which they are designed, but also for other, general metrics, as described in Section 5.2. All algorithms were implemented using the g++ 4.1.2 compiler. The experiments were run on a 2 CPU Xeon 2.8GHz machine with 2.5GB of main memory running Centos 5.2.

Description of Data For our quality assessment, we have used two real-world data sets characterized by hard to approximate bursts and discontinuities, and another real data set with more pronounced continuity features. In order to enable binary-interval-based techniques such as Haar$^+$ and CHH to perform smoothly, we have used binary data sizes. Our first data set[1] (FR) is discussed in [33]; it contains a sequence of the mean monthly flows of the Fraser River at Hope, B.C. The flows present periodic autoregression features, while they average at 2709 with standard deviation 2123 and feature discontinuities (min value: 482, max value: 10800). We have used a 512-value prefix of this data set. Our second data set[2] (FC) is extracted from a relation of 581,012 tuples describing the forest cover type for 30 x 30 meter cells, obtained from US Forest Service. FC contains the frequencies of the distinct values of attribute `Aspect` in the relation. The frequencies average at 1613 (standard deviation: 730) and feature spikes of large values (min value: 499, max value: 6308). We have used a 256-value prefix of FC. The third data set (DJIA) hails from the Dow-Jones Industrial Average (DJIA) data set[3]; it contains closing values of the Dow-Jones Industrial Average index from 1900 to 1993. Negative values were removed. We used a 512-value subset of closing values from April 14th, 1948 to February 8th, 1950. The closing values average at 182 (standard deviation: 8.73) and exhibit both continuities and hierarchical patterns (min value: 161.6, max value: 205.03). Even though the GreedyLattice algorithm can run on very large data sets, we focus on data sizes for which the OptimalLattice algorithm can run smoothly, facilitating our comparison.

[1]Available at http://lib.stat.cmu.edu/datasets/fraser-river
[2]Available at http://kdd.ics.uci.edu/

6.1 Synopsis Accuracy on Bursty Data

We first examine how several techniques perform on data containing spikes and discontinuities.

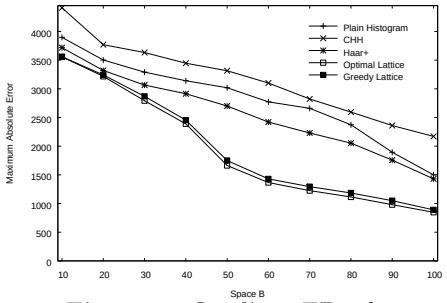

Figure 9: Quality: FR, $\mathcal{L}_\infty$

In our first experiment we evaluate the accuracy achieved with the $\mathcal{L}_\infty$ metric on the the FR data set. Figure 9 shows the results we obtained. The Haar$^+$ technique summarizes data using a resolution parameter δ. This parameter has been set at $\delta = 50$. Smaller values did not confer any significant quality benefit. The OptimalLattice achieves observably the highest quality. Moreover, remarkably, the more efficient GreedyLattice technique *follows* the accuracy performance of OptimalLattice from a very close distance, and also outperforms all other contenders. Haar$^+$ achieves the best quality among these other techniques.

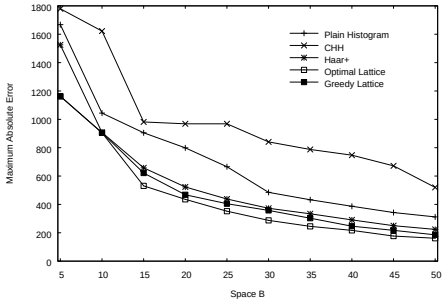

Figure 10: Quality: FC, $\mathcal{L}_\infty$

Our second experiment (Figure 10) examines the state of affairs with the FC data set. The resolution of Haar$^+$ was set at $\delta = 10$. The picture is pretty much the same, while Haar$^+$ performs slightly better in this case. Still, both our LH algorithms achieve consistently higher accuracy, while GreedyLattice is again approaching at a close distance from OptimalLattice. This observation further verifies the claim that our greedy algorithm achieves near-optimal quality.

Next, we examine the accuracy on the *contrasting* $\mathcal{L}_1$ metric with the the same data sets and resolution settings. Now both LH algorithms operate as heuristics, as described in Section 5.2. Figure 11 shows the FR results. Despite the

[3]Available at http://lib.stat.cmu.edu/datasets/djdc0093

fact that none of our LH heuristics confers any guarantee of optimality in this case, they both achieve mostly the highest accuracy in this case as well. Moreover, the Greedy-based heuristic is closely following the results of the Optimal-based one in this setting too. A weakness of performance appears only for small values of space budget B. This weakness is natural, and arises from the fact that a small space budget, while well disposed for a given maximum-error metric, may not occupy LH nodes suitable for a metric of different characteristics. Still, this effect is attenuated as the space budget grows. Among the other contenders, Haar^+ again performs best.

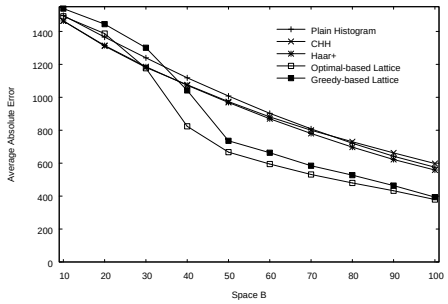

Figure 11: Quality: FR, $\mathcal{L}_1$

Figure 12 shows the results with the FC data set. In this case, the Optimal-based algorithm again outperforms other contenders, even though being a heuristic. This result is due to the inherent advantages of the Lattice Histograms. Still, even though the Greedy-based heuristic follows the same trend as the Optimal-based one, it does not perform as well compared to Haar^+ in this case. Again, this result is due to the fact that good choices for a maximum-error metric are not always as good for a different metric.

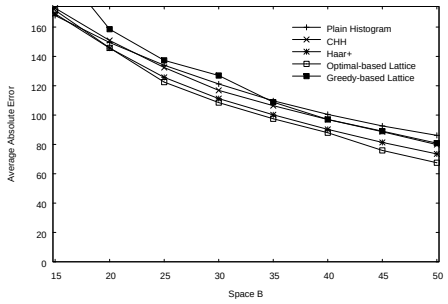

Figure 12: Quality: FC, $\mathcal{L}_1$

6.2 Synopsis Accuracy on Smooth Data

We now turn our attention to the accuracy on smoother data, which do not present as sharp discontinuous features.

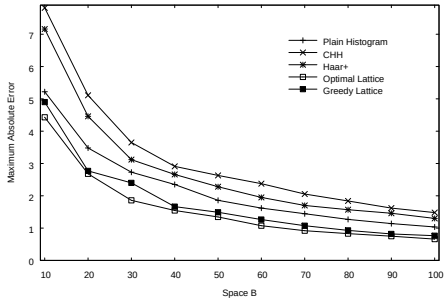

Figure 13: Quality: DJIA, $\mathcal{L}_\infty$

Figure 13 shows the results on the DJIA set with the maximum absolute error metric. The resolution value for Haar^+ is now set at $\delta = 0.5$. Again, the Lattice algorithm achieves the highest quality. More interestingly, in this case the best performer among the other techniques is not the Haar^+ any more; now the plain histogram does better. Still, despite this change of affairs among the other techniques, the Lattice algorithms remain unchallenged, while the Greedy method almost matches the optimal Lattice Histogram quality again.

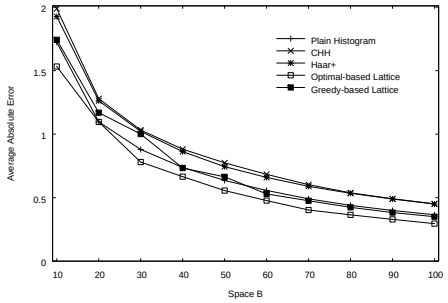

Figure 14: Quality: DJIA, $\mathcal{L}_1$

Last, the results for DJIA with $\mathcal{L}_1$ are in Figure 14. The relationship among techniques is again the same. Now the difference between the Lattice-based heuristics and the other techniques is less pronounced, but holds. Thus, even the fast greedy LH heuristic matches the $O(n^2 B \log n)$ $\mathcal{L}_1$-optimal plain histogram algorithm. The Greedy-based algorithm again follows the Optimal-based heuristic closely.

6.3 Runtime Comparison

We now extend our comparison to a runtime assessment. For this purpose we constructed data sets of various sizes by randomly drawing values from FR. We varied the synopsis budget B linearly with the data set size n, as $B = \frac{n}{8}$. Figure 15 shows the results. As expected, the runtime difference between the OptimalLattice and GreedyLattice is paramount, while other methods fall in between. Thus, GreedyLattice emerges as a genuinely practical algorithm, possessing advantages of both efficiency and quality.

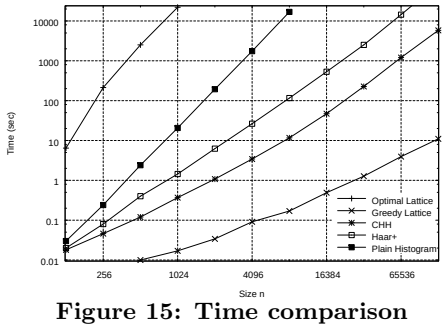

Figure 15: Time comparison

7. DISCUSSION

A remarkable finding of our results is that the relatively simple greedy algorithm can construct Lattice Histograms of near-optimal quality, outperforming other summarization techniques in both quality and efficiency. The accuracy it achieves generally follows the optimal accuracy at a close distance, with rare exceptions. However, the runtime difference between the two LH algorithms is much more significant, with the optimal-accuracy algorithm running in near-cubic time while the greedy runs in near-linear time. Moreover,

the accuracy results are consistent across data posing diverse requirements, with different second-best contenders in each case, and are maintained even when our algorithms are deployed as heuristics for a non-maximum-error metric.

8. CONCLUSIONS

A Lattice Histogram approximates data by discovering hierarchical associations without the constraint of a predefined hierarchy. This paper has provided two divergent, yet related, contributions in the domain of LH construction algorithms. First, we have addressed the question of *optimality* in LH construction, by proposing a low-polynomial-time dynamic programming scheme that calculates a minimal-space LH under any maximum-error-based error bound, while secondarily optimizing the actual error in this minimal space. Furthermore, we have shown that the same scheme robustly achieves a minimal-error solution to the space-bounded LH construction problem. Secondly, we have also treated the issue of *scalability*, by providing an efficient near-linear greedy algorithm for the same problems. Our experimental investigation has verified the effectiveness of these LH methods at achieving high-accuracy data approximations compared to conventional techniques. More significantly, we have found that the greedy algorithm consistently provides near-optimal accuracy, even though it is much less computationally demanding than the algorithm that guarantees optimality.

9. REFERENCES

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: building histograms without looking at data. In *SIGMOD*, 1999.

[2] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, 1999.

[3] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: a multidimensional workload-aware histogram. In *SIGMOD*, 2001.

[4] F. Buccafurri, G. Lax, D. Saccà, L. Pontieri, and D. Rosaci. Enhancing histograms by tree-like bucket indices. *The VLDB Journal*, 17(5):1041–1061, 2008.

[5] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The VLDB Journal*, 10(2-3):199–223, 2001.

[6] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *TODS*, 27(2):188–228, 2002.

[7] F. Furfaro, G. M. Mazzeo, D. Saccà, and C. Sirangelo. Hierarchical binary histograms for summarizing multi-dimensional data. In *ACM SAC*, 2005.

[8] M. Garofalakis and P. B. Gibbons. Probabilistic wavelet synopses. *TODS*, 29(1):43–90, 2004.

[9] M. Garofalakis and A. Kumar. Wavelet synopses for general error metrics. *TODS*, 30(4):888–928, 2005.

[10] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In *SODA*, 1999.

[11] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *TODS*, 27(3):261–298, 2002.

[12] S. Guha. On the space—time of optimal, approximate and streaming algorithms for synopsis construction problems. *The VLDB Journal*, 17(6):1509–1535, 2008.

[13] S. Guha and B. Harb. Approximation algorithms for wavelet transform coding of data streams. *IEEE Transactions on Information Theory*, 54(2):811–830, 2008.

[14] S. Guha, N. Koudas, and K. Shim. Approximation and streaming algorithms for histogram construction problems. *TODS*, 31(1):396–438, 2006.

[15] S. Guha and K. Shim. A note on linear time algorithms for maximum error histograms. *IEEE TKDE*, 19(7), 2007.

[16] S. Guha, K. Shim, and J. Woo. REHIST: Relative error histogram construction algorithms. In *VLDB*, 2004.

[17] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *The VLDB Journal*, 14(2):137–154, 2005.

[18] Y. E. Ioannidis. Universality of serial histograms. In *VLDB*, 1993.

[19] Y. E. Ioannidis. Approximations in database systems. In *ICDT*, 2003.

[20] Y. E. Ioannidis. The history of histograms (abridged). In *VLDB*, 2003.

[21] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *SIGMOD*, 1995.

[22] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *VLDB*, 1999.

[23] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, 1998.

[24] P. Karras. Multiplicative synopses for relative-error metrics. In *EDBT*, 2009.

[25] P. Karras and N. Mamoulis. One-pass wavelet synopses for maximum-error metrics. In *VLDB*, 2005.

[26] P. Karras and N. Mamoulis. The Haar$^+$ tree: a refined synopsis data structure. In *ICDE*, 2007.

[27] P. Karras and N. Mamoulis. Hierarchical synopses with optimal error guarantees. *ACM TODS*, 33(3):1–53, 2008.

[28] P. Karras and N. Mamoulis. Lattice histograms: a resilient synopsis structure. In *ICDE*, 2008.

[29] P. Karras, D. Sacharidis, and N. Mamoulis. Exploiting duality in summarization with deterministic guarantees. In *KDD*, 2007.

[30] A. C. König and G. Weikum. Combining histograms and parametric curve fitting for feedback-driven query result-size estimation. In *VLDB*, 1999.

[31] T. Li, Q. Li, S. Zhu, and M. Ogihara. A survey on wavelet applications in data mining. *SIGKDD Explorations Newsletter*, 4(2):49–68, 2002.

[32] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*, 1998.

[33] A. McLeod. Diagnostic checking of periodic autoregression models with application. *Journal of Time Series Analysis*, 15(2):221–233, 1994.

[34] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *SIGMOD*, 1988.

[35] S. Muthukrishnan. Subquadratic algorithms for workload-aware Haar wavelet synopses. In *FSTTCS*, 2005.

[36] V. Poosala, V. Ganti, and Y. E. Ioannidis. Approximate query answering using histograms. *IEEE Data Eng. Bull.*, 22(4):5–14, 1999.

[37] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, 1997.

[38] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *SIGMOD*, 1996.

[39] F. Reiss, M. Garofalakis, and J. M. Hellerstein. Compact histograms for hierarchical identifiers. In *VLDB*, 2006.

[40] E. Terzi and P. Tsaparas. Efficient algorithms for sequence segmentation. In *SIAM SDM*, 2006.

[41] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, 1999.

[42] H. Wang and K. C. Sevcik. Histograms based on the minimum description length principle. *The VLDB Journal*, 17(3):419–442, 2008.