IEICE TRANS. INF. & SYST., VOL.E102–D, NO.4 APRIL 2019

PAPER Special Section on Data Engineering and Information Management

Building Hierarchical Spatial Histograms for Exploratory Analysis in Array DBMS

Jing ZHAO[†a)], ***Nonmember***, **Yoshiharu ISHIKAWA**[††b)], ***Member***, **Lei CHEN**[†††c)], **Chuan XIAO**[††††d)], ***and*** **Kento SUGIURA**[††e)], ***Nonmembers***

SUMMARY As big data attracts attention in a variety of fields, research on data exploration for analyzing large-scale scientific data has gained popularity. To support exploratory analysis of scientific data, effective summarization and visualization of the target data as well as seamless cooperation with modern data management systems are in demand. In this paper, we focus on the exploration-based analysis of scientific array data, and define a *spatial V-Optimal histogram* to summarize it based on the notion of histograms in the database research area. We propose histogram construction approaches based on a general *hierarchical partitioning* as well as a more specific one, the *l-grid partitioning*, for effective and efficient data visualization in scientific data analysis. In addition, we implement the proposed algorithms on the state-of-the-art array DBMS, which is appropriate to process and manage scientific data. Experiments are conducted using massive evacuation simulation data in tsunami disasters, real taxi data as well as synthetic data, to verify the effectiveness and efficiency of our methods.

key words: *spatial histograms, exploratory analysis, array DBMSs*

1. Introduction

User-driven analysis is designed for traditional database systems, which assume that users are aware of exactly what they are looking for and have good knowledge of the contents of the database. However, as big data has made great success in a variety of fields, it leads to another type of analysis, *data-driven analysis*, in which it is not necessary for users to have prior knowledge about the target dataset or be aware of the type of queries they want to pose. Users often explore massive data by navigation or visualization tools to find interesting patterns. As a result, research on *data exploration* for sophisticated analytic processing of large amount of data has gained popularity [13].

As one of the common data types of big data, *spatio-temporal data* has been widely applied in various domains such as mobile applications and scientific research [8]. For instance, in scientific fields, simulations are conducted for the purpose of predictions, decision making, etc. As one of the typical simulations, disaster simulations like human evacuation simulation are conducted for effective humanitarian relief and disaster management [25]. Such kind of disaster simulations generates large scale spatio-temporal data, which contains spatial and temporal information of evacuees on the target area during a period of time. Analysis of disaster simulation data can achieve various objectives, such as discovery of interesting patterns and shelter location suggestion [12]. Consequently, researches on enabling exploratory analysis of spatio-temporal data has been increasing in demand. Moreover, since a large amount of simulation data is generated by simulations with different conditions and parameters (e.g., time when earthquake occurs), *data warehousing* techniques that enabling massive data storage and exploration are also important [30].

In this paper, we focus on the exploration-based analysis of spatio-temporal array data, which is based on the *array* representation for large-scale scientific data computation. While the traditional relational model cannot handle such array data, we exploit the functionality of array database management systems (array DBMSs) [2], to support scientific computing and exploratory analysis. A motivating example is as follows. Consider an earthquake analyst intends to explore the evacuation simulation data after an earthquake. A query like "return the spatial distribution of evacuees during the first hour after the earthquake occurs" is used to understand the movement of evacuees during the given period of time. There are two main challenges of exploration-based analysis of spatio-temporal array data: 1) Spatio-temporal data has characteristics on spatial and temporal scales because of the semantic meanings of different granularities, such as one day, one hour and one minute on the time dimension, as well as country, city and district on the spatial dimension. However, instead of exhausting exploration by drill down and roll up operators, more advanced operators such as effective summarization and visualization that navigates users to find interesting knowledge are required. 2) Since the target spatial area of simulation data is usually quite large, it is difficult for users to determine which part of the area should be focused on. In this case, a succinct summary of the overall region is considered to be useful.

In order to summarize the spatio-temporal array data

Manuscript received June 27, 2018.
Manuscript revised November 5, 2018.
Manuscript publicized January 18, 2019.

[†]The author is with Graduate School of Information Science, Nagoya University, Nagoya-shi, 464–8601 Japan.
[††]The authors are with Graduate School of Informatics, Nagoya University, Nagoya-shi, 464–8601 Japan.
[†††]The author is with Hong Kong University of Science and Technology, Hong Kong.
[††††]The author is with Institute for Advanced Research, Nagoya University, Nagoya-shi, 464–8601 Japan.
a) E-mail: zhao@db.is.i.nagoya-u.ac.jp
b) E-mail: ishikawa@i.nagoya-u.jp
c) E-mail: leichen@cse.ust.hk
d) E-mail: chuanx@nagoya-u.jp
e) E-mail: sugiura@db.is.i.nagoya-u.ac.jp
DOI: 10.1587/transinf.2018DAP0020

to show the overall data distribution, we use the notion of *histogram* [14], which is widely used in database systems to summarize relations for query selectivity estimation and approximate query answering [1], [4], [15], [22]. Such histograms provide a concise summary of the target data, and the accuracy depends on the chosen partition rules. Other techniques for data summarization, such as *wavelets* [17], which is important for image compression, has also been used extensively for approximate query answering in the database area, while it is not the focus of this paper.

Existing work on histograms generally focus on the accuracy of histograms or approximation of query results (such as selectively estimation), while our work considers the quality of histograms as well as the efficiency of histogram construction, which is important for visualization and data exploration. In this paper, we define a *spatial V-optimal histogram* to represent the array data on the two-dimensional spatial area, as an extension of one-dimensional *V-optimal histogram* [15]. However, building histograms is often very time consuming, especially for spatial V-optimal histogram, which is proved to be NP-hard [18]. Unlike the error-bounded V-optimal histogram construction problem studied in [18], we focus on the space-bounded histogram construction, and propose exact and approximate algorithms based on dynamic programming for *hierarchical* histogram construction. We prove that the exact and approximate algorithms have approximation bounds with respect to the arbitrary partitioning method. In order to enable interactive analysis of large-scale array data, we also propose a heuristic approach based on *l-grid partitioning*, which is a specific hierarchical partitioning, to speed up the histogram construction further. In addition, we implement our proposed histogram construction methods on SciDB [26], [27], an open source array-oriented DBMS for the efficient storage and manipulation of large array data. We perform experimental evaluation on evacuation simulation data as well as real taxi trajectory data to show the effectiveness and the efficiency of the proposed methods.

Our contributions can be summarized as follows:

- We study the problem of constructing histograms that summarize the data distribution of a specific spatial area during a time interval (Sect. 2).
- We propose both exact and approximate algorithms to construct the *spatial V-optimal histogram*. In addition, we also propose a heuristic algorithm based on *l*-grid partitioning for fast histogram construction (Sect. 3).
- We implement the proposed histogram construction algorithms on SciDB, the state-of-the-art array DBMS (Sect. 4).
- We conduct extensive experiments on massive simulation data, real taxi data as well as synthetic data with different distributions, to verify the effectiveness and the efficiency of the proposed methods (Sect. 5).

The rest of the paper is organized as follows: Sect. 2 introduces preliminaries and the problem definitions in this paper. Section 3 presents our method for constructing hierarchical spatial histograms for array data analytics. Section 4 discusses the implementation of our method on an array DBMS, SciDB. Experimental results and analyses are reported in Sect. 5. Section 6 introduces the related work on histograms and data visualization. Section 7 concludes the paper.

2. Preliminaries

In this section, first we introduce the basic definition of spatio-temporal arrays. Then, we formally define the histogram construction problem that we are going to investigate in this work.

Definition 1 (Spatio-temporal Array): A *spatio-temporal array* $\mathcal{A}(Dim, Attr)$ is a three-dimensional array with numeric attribute values. Dim consists of dimensions x, y and t, while $Attr$ contains a list of attributes $(a_1, \cdots, a_{|Attr|})$. Dimensions x and y correspond to a two-dimensional spatial grid structure, and dimension t is represented by a series of time stamps with equal time interval τ. Each element $e_{x,y,t}$ in the spatio-temporal array contains a tuple of numeric attribute values corresponding to the position (x, y, t), that belongs to the domains of dimensions, i.e., $x \in Dim_x, y \in Dim_y$ and $t \in Dim_t$.

Note that the time interval τ can be one minute, one hour, etc., and depends on the dataset and the purpose of analysis.

In order to catch an insight of spatio-temporal arrays, we use the notion of *histograms* from the studies on selectivity estimation and query optimization in the database area [1], [4], [15], [22]. In this paper, we formulate the problem of constructing histograms for array data exploration based on the *space-bounded V-optimal histogram* [15], which is defined as follows. For a given number of buckets, a V-optimal histogram is the one with the number of buckets bounded by the specified threshold, but having the least variance, where variance is the sum of squared differences between the actual and approximate frequencies. While [15] proposed algorithms with a quality guarantee for *unidimensional V-optimal* histograms, the problem becomes totally different even in the two-dimensional case.

In our work, we focus on the two-dimensional space-bounded V-optimal histogram because of its bound on the number of buckets, which is able to enhance the usability of visualization. More precisely, the number of buckets in the result histogram affects the quality of the visualization, since visualizing the objective data by a histogram with too many buckets will confuse an analyst and increase the inefficiency of exploratory analysis [6]. Therefore, a space bound (i.e., the number of buckets) is considered to be important for the usability of visualization.

Definition 2 (Spatial V-optimal Histogram): Given a spatial-temporal array $\mathcal{A}$, an attribute a, an integer B defining the limit number of buckets, and an error metric $E()$, the *spatial V-optimal histogram* H of $\mathcal{A}$ consists of a set of buckets $\{b_1, b_2, \cdots, b_B\}$ with the minimum error. The histogram

is generated by partitioning the whole spatial region into B non-overlapping buckets. Each bucket b_l ($1 \le l \le B$) has a corresponding rectangle area $b_l.area$ and an aggregated value $b_l.val$. The value of $b_l.val$ is calculated by averaging the attribute values of elements in $\mathcal{A}$, the area of which is covered by $b_l.area$.

Next, we define the error function. It is based on the notion of *sum of squared error* (SSE), which is a common error metric for measuring difference between two data distributions.

Definition 3 (Error Function): The *error function* E of a spatial V-optimal histogram H is defined as

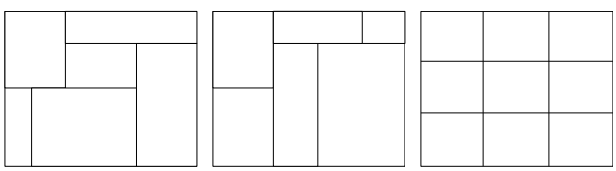
$$E_B(H) = \sum_{l=1}^{B} \sum_{e \in \mathcal{A} \wedge e.(x,y) \in b_l.area} (e.a - b_l.val)^2. \quad (1)$$

The defined error function has two properties, *monotonicity* and *superadditiveness*. An error function is *monotonic* if $E(r) \le E(R)$ for two buckets r and R, where $r \in R$. We say an error metric is *superadditive* if $E(r) + E(R) \le E(r \cup R)$, where r and R are two disjoint buckets.

[18] proved the problem of minimizing the heft (i.e., error metric value) of the partitioning with p tiles on two dimensional array is NP-hard for some measures, where the metric of sum of squared error (SSE) is one of those metrics. The problem of constructing a spatial V-optimal histogram defined in this paper is intrinsically the same as the proved NP-hard problem, and the proof is based on a reduction from the *Planar 3-SAT problem* (shown to be NP-complete in [16]).

In general, the histogram construction problem changes with different partitioning strategies. For a two-dimensional array, there are many types of partitioning. Here, we show the three commonly used ones in Fig. 1 [18]. *Arbitrary partitioning* is the one with no restrictions on the sub-rectangles shown in Fig. 1 (a). It is obvious that the computation cost of arbitrary partitioning is the most expensive one, which is proved to be NP-hard as described above. In this paper, we consider the *hierarchical partitioning* shown in Fig. 1 (b). A general hierarchical partitioning can be represented by a binary tree in which each node represents a subarray, and the root represents the whole region of the computed array. The $p \times p$ *partitioning* conducts p times of partitions on each dimension, as shown in Fig. 1 (c). It is a special case of the hierarchical one if the sibling nodes of the hierarchical tree are the same along one dimension.

We propose both exact and approximate algorithms for hierarchical partitioning, and prove that compared to the arbitrary partitioning method, the errors introduced by the exact or the approximate algorithm are bounded. In order to enable interactive analysis of large-scale array data, we also propose heuristic approaches based on the l-grid partitioning, which is a special case of hierarchical partitioning.



(a) Arbitrary (b) Hierarchical (c) $p \times p$

Fig. 1 Types of partitioning

3. Hierarchical Histograms

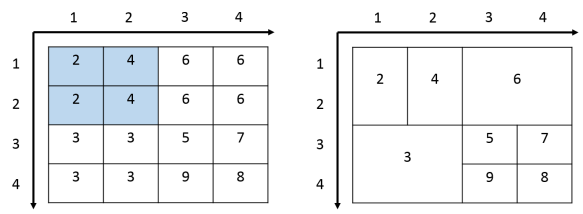
In this section, first we propose an exact histogram construction algorithm based on dynamic programming for general hierarchical partitioning. Since the computational cost of dynamic solution is expensive, we propose an approximate solution with an approximation bound. Both of the two algorithms are proved to have approximation bounds with respect to the arbitrary partitioning, in which no restrictions on the arrangement of buckets. In addition, we also propose a heuristic algorithm for the efficiency.

3.1 Exact Algorithm

Recall that the problem is to find a histogram which has at most B buckets with minimum error. We define $E^*_B(R)$ as the minimum error of region $R([i \cdots j], [k \cdots l])$ with at most B buckets. The computation strategy of the exact algorithm is based on the following equation.

$$\begin{aligned} E^*_B(R) = \min_{i \le x < j, k \le y < l, 1 \le b < B} \{ & E^*_b(i \cdots x, k \cdots l) \\ & + E^*_{B-b}(x+1 \cdots j, k \cdots l), E^*_b(i \cdots j, k \cdots y) \\ & + E^*_{B-b}(i \cdots j, y+1 \cdots l) \} \end{aligned} \quad (2)$$

That is, the minimum error of region R is the minimum error among the possible partitions along x (horizontal) and y (vertical) dimension with possible number of buckets. As shown in Eq. (2), b represents the possible number of buckets assigned to each subregion by a partition. For each partition, there are $B-1$ possible values of b, i.e., $1, 2, \ldots, B-1$. Furthermore, there are $j - i$ possible partitions along x dimension, and $l - k$ possible partitions for y dimension. For instance, given a 4×4 array $\mathcal{A}$ shown in Fig. 2 (a), the whole region of $\mathcal{A}$ is represented as $R([1 \cdots 4], [1 \cdots 4])$. There are 3 possible partitions along y dimension (lines in blue), each of which splits the whole region into two subregions, as shown in Fig. 3. Note that, for each subregion, there are also several possible partitions to split it into further smaller subregions. The same holds for x dimension.



(a) An example 2-D array $\mathcal{A}$ (b) The optimal histogram ($B = 8$)

Fig. 2 An example array with its optimal histogram

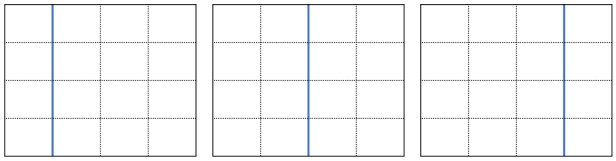


Fig. 3 Possible partitions along y dimension

The exact algorithm is based on dynamic programming using Eq. (2) as described above. Consider the whole region of the input array is represented as R, in order to compute the optimal result of R, we incrementally compute the possible subregions of R by increasing the side length of each dimension. E.g., for array $\mathcal{A}$ with region $R([1,4],[1,4])$, we incrementally compute the subregions with side lengths of $1, 2, 3$ and 4, for each dimension. As a result, there are 10^2 possible subregions of the whole region for two dimensions. Figure 2 (b) shows an optimal histogram of A when the number of buckets $B = 8$, the error of which is 0.

3.1.1 Complexity Analysis and the Approximation Bound

The total computation time is computed by $O(n^4(nB^2 + t_E))$, where $O(n^4)$ represents the possible number of subregions, and $O(nB^2 + t_E)$ represents the cost of computing candidate histograms for each subregions. t_E is an upper bound on the time taken to calculate the error metric value of any given bucket. For each subregion, we compute $O(B)$ optimal result histograms with bucket number $b \in [1, B]$. Moreover, for the result histogram with one bucket, one calculation of the error of the target region is necessary, which is t_E, while computing the result histograms with bucket number b ($1 < b \leq B$), $O(nB)$ computation cost is needed based on Eq. (2). Since the error of a given bucket can be computed in $O(1)$ by efficient methods such as holding the statistic information of subregions for further computation, t_E can be reduced to $O(1)$. Therefore, the total computation time is $O(N^{2.5}B^2)$, where N is the size of the computed array (i.e., $N = n^2$).

In order to compute the optimal histogram of a target region with a given bucket number B, the optimal results (i.e., a list of optimal solutions with the possible number of buckets) of its subregions are temporally held. Since there are totally $O(N^2)$ possible subregions for the whole region R, the space complexity of the algorithm is $O(N^2B)$.

In what follows, we argue that the proposed exact algorithm for hierarchical histogram construction can approximately solve the problem of arbitrary partitioning with the approximation bound.

Lemma 1: Consider any arbitrary partitioning of an array $\mathcal{A}$ with superadditive error metric at most δ and B buckets. There exists a hierarchical histogram of $\mathcal{A}$ with error metric at most δ and at most $4B$ buckets.

Proof. It is proved that any rectangular partition can be converted into a hierarchical one by splitting each rectangle to at most four disjoint rectangles [5]. Consider a given histogram which has B buckets with error δ, it can be converted into a hierarchical one with at most $4B$ buckets. Moreover, since the error metric is superadditive, the metric value of this resulting hierarchical histogram is at most δ. □

Based on the above observation with the construction algorithm of hierarchical histograms, we can obtain the following result.

Theorem 1: Let δ_0 be the minimum error of the arbitrary partitioning problem with at most B_0 buckets under a superadditive error metric. Then, in $O(N^{2.5}B^2)$ time, we can compute a histogram consists of at most B buckets with error $\delta \leq \delta_0$ and $B \leq 4B_0$.

3.2 Approximate Algorithm

Since the proposed exact algorithm for hierarchical partitioning is quite computationally expensive, next we propose an approximation algorithm to improve the running time of dynamic programming algorithm by limiting the permissible side intervals of buckets. Moreover, the approximation algorithm can control the approximation bound in a structured manner as described below.

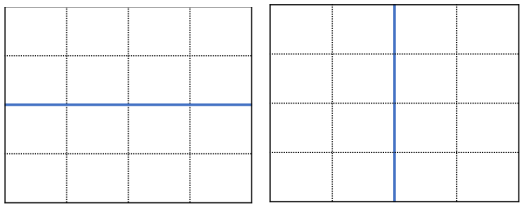
Given a set of positive integers $L_0, L_1, L_2, \ldots, L_k$, which is a geometric progression of values, such that $L_0 = n$ and L_{i+1} divides L_i, for $0 < i < k$. For each L_i, there is a set of permissible intervals S_i that the length of each interval in S_i is a multiple of L_i. In other words, the permissible intervals of L_i are intervals $[jL_i+1, lL_i+1]$, for some positive integers $j < l$, such that $lL_i - jL_i + 1 < L_{i-1}$. For instance, if $L_0 = 3L_1$, the permissible intervals of L_1 for each dimension comprise the following intervals with side lengths L_1 and $2L_1$: $[1, L_1], [L_1 + 1, 2L_1], [2L_1 + 1, 3L_1], [1, 2L_1], [L_1 + 1, 3L_1]$. For L_k, since we aim to compute the optimal result for the intervals with length L_k, there's no constraints on the interval length. Therefore, the permissible intervals of S_k contain intervals in S_i for $i = k$ as defined above, as well as all their subintervals.

In the same example of a two dimensional array $\mathcal{A}$ in Fig. 2 (a), each dimension corresponds to the target spatial region of interval $[1, 4]$. Each element of the array contains a numeric attribute value at the corresponding position. Given $L_0 = 4$ and $L_k = L_1 = 2$, the permissible intervals of each dimension (x or y) for L_0, represented as S_0 is $[1, 4]$. For L_1, which is also L_k, the permissible intervals S_1 consists of $[1, 2]$ and $[3, 4]$ with side length 2, as well as all the subintervals of them, i.e., $[1, 1], [2, 2], [3, 3], [4, 4]$.

We define a *k-hierarchical histogram* based on the following *partition rules* for each dimension.

1. For intervals with side lengths in S_i ($0 \leq i < k$), the permissible partitions split the interval into subintervals in S_{i+1}, the lengths of which are multiples of L_{i+1}.
2. For intervals with side length of L_k, all the subintervals are permissible by partitions with no constraint.

For instance, Fig. 4 shows the possible partitions (lines in blue) for intervals with length $L_0 = 4$. Based on the partition rule (1), there's only one possible partition along x



(a) partition on x dimension (b) partition on y dimension

Fig. 4 Possible partitions for side length of $L_0 = 4$

Algorithm 1: AprDP($\mathcal{A}, B, S$)

Input: $\mathcal{A}$: input array, B: maximal number of buckets, $S = \{L_k, L_{k-1}, \cdots, L_0\}$: permissible side lengths
Output: Result histogram $H_B(\mathcal{A})$

1 $C \leftarrow \emptyset, C' \leftarrow \emptyset$; `// Initialize the candidates`
2 **foreach** $xl \in S$ **do**
3 $C \leftarrow C', C' \leftarrow \emptyset$; `// Update C with the results of the previous loop, clear C'`
4 **foreach** $yl \in S$ **do**
5 Divide the whole region of $\mathcal{A}$ evenly into subregions R with size (xl, yl);
6 **foreach** $r \in R$ **do**
7 **foreach** $b \in [1, B]$ **do**
8 **if** $xl = y_l = L_k$ **then**
9 Compute $H_b(r)$ by the exact algorithm;
10 **else if** $xl = L_k \vee yl = L_k$ **then**
11 Compute $H_b(r)$ by Eq. (2) based on partition rules (1) and (2);
12 **else**
13 Compute $H_b(r)$ by Eq. (2) based on partition rule (1);
14 $C' \leftarrow C' \cup H_b(r)$; `// Update candidates`
15 **return** $H_B(\mathcal{A})$;

and y dimension, respectively, since the subintervals should equal to $L_1 = 2$. For $L_k = 2$ (rule (2)), consider a subregion $r([1, 2], [1, 2])$, the shaded area shown in Fig. 2 (a), the possible partitions along x and y dimension are also two (one for each dimension).

Based on the partition rules described above, we propose an approximation algorithm to efficiently construct histograms, which is again based on dynamic programming. The basic idea is to incrementally compute the candidate results of possible subregions by increasing side lengths. The pseudo-code is illustrated in Algorithm 1.

We incrementally compute the result histograms of subregions with side lengths in S_i for each dimension, where $i \in \{k, k-1, \cdots, 0\}$ (Line 2–14). For each subregion r, we compute the result histogram $H_b(r)$ with bucket number $b \in [1, B]$ based on Eq. (2) (Line 6–14). As different partition rules are conducted according to different side lengths of target regions, we consider three cases as follows. First, we call the exact algorithm for the subregions with both of the two side lengths equal to L_k (Line 8–9). The second case is that, only one of the side lengths equals to L_k (Line 10–11). In this case, we partition on the dimension with side length larger than L_k by partition rule (1), while for the dimension with side length of L_k, we use partition rule (2). The final case is that both of the side lengths are larger than L_k, in which case we partition the target region based on partition rule (1). After we compute the result histograms of the whole region of $\mathcal{A}$, the result $H_B(\mathcal{A})$ is returned.

3.2.1 Complexity Analysis and Approximation Bounds

There are $O(\frac{n^2}{L_k^2})$ possible subregions, for each of which, we compute $O(B)$ result histograms with the possible bucket number $b \in [1, B]$. The computation cost of each $H_b(r)$ is $O(L_k^5 B)$, calculated by the cost of exact algorithm as mentioned in the last section, which is larger than the cost of other cases (i.e., Line 10–13 in Algorithm 1). As a result, the computation cost for all of the generated subregions is $O(n^2 L_k^3 B^2)$. We assume that L_k is bounded by n^ϵ, where ϵ is a small positive constant factor, then the total computation cost is represented as $O(N^{1+1.5\epsilon} B^2)$.

For space complexity, as shown in Algorithm 1, the candidates of subregions with region size (xl, yl), for each xl and all $yl \in S$ are temporally held in C'. There are the $O(\frac{n^2}{L_k^2})$ possible subregions, for each of which, there are $O(B)$ possible candidate results with different bucket numbers, as well as $O(L_k)$ candidate results of subregions with one side length equals to 1, are needed for further computation. Therefore, the total space complexity is $O(\frac{n^2}{L_k} B)$, represented as $O(N^{1-\frac{\epsilon}{2}} B)$.

In the following part, we also argue that the proposed approximation algorithm can approximately solve the hierarchical partitioning problem as well as arbitrary partitioning problem with the approximation bounds.

Lemma 2: Consider an optimal hierarchical histogram of an array $\mathcal{A}$ with superadditive error metric at most δ and B buckets. There exists a k-hierarchical histogram of $\mathcal{A}$ with error metric at most δ and at most $(2k + 1)^2 B$ buckets.

Proof. Consider the optimal histogram of array $\mathcal{A}$ with B buckets is known as H^* with error δ, and the optimal k-hierarchical histogram of $\mathcal{A}$ is represented as H. For any arbitrary bucket b^* in H^*, there are at most $(2k + 1)^2$ buckets generated in H with the same error reduction as b^* in H^*. The reason is basically based on the fact of possible partitions of the k-hierarchical histogram are constrained by the given k integers, i.e., $L_1, L_2, \cdots, L_k$. In detail, consider the side interval of b^* along one dimension is $[a, b]$ with length larger than L_1, where n is a multiply of L_1, as shown in Fig. 5. For the endpoint a, there are at most k partitions that are necessary if we use the approximate algorithm based on the binary partitioning, since each partition is conducted only on the positions specified by L_i as mentioned. As shown in Fig. 5, there are at most $P_1, P_2, \ldots, P_k$ partitions for a, and the same holds for endpoint b. Therefore, there are totally $2k + 1$ intervals generated in H for interval

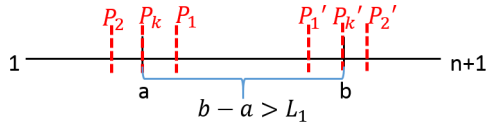


Fig. 5 Number of partitions for $[a, b]$

$[a, b]$, so that there are at most $(2k + 1)^2$ buckets generated in H for each bucket in H^*. Moreover, since the error metric is monotonic and superadditive, the error of H is not bigger than $E(H^*)$. Based on the above mentioned observation, there are at most $(2k + 1)^2 B$ buckets in H, with an error of at most δ. □

Based on the analysis described above, we can conclude as follows.

Theorem 2: For any superadditive error metric, in $O(N^{1+1.5\epsilon}B^2)$ time, we can compute a histogram consists of at most B buckets with error $\delta \leq \delta_0$ and $B \leq O((1 - \epsilon)^2(\log n)^2 B_0)$, where δ_0 is the minimum error of any arbitrary histogram with at most B_0 buckets.

Proof. As we know, ϵ determines the time complexity of the approximate algorithm, as well as the length of L_k, which determines the possible permissible side lengths by $k = \log n - \log L_k$. Since $L_k = n^\epsilon$, then $k = (1 - \epsilon) \log n$. Based on Lemma 2, we can know that, for an optimal hierarchical histogram of array A that has at most B_0 buckets with error δ_0, in $O(N^{1+1.5\epsilon}B^2)$ time, we can find a k-hierarchical histogram that has at most $O((1-\epsilon)^2(\log n)^2 B_0)$ buckets with error δ_0. Moreover, based on the above results and Theorem 1, we can get the approximation bound for arbitrary histograms as $O(4(1 - \epsilon)^2(\log n)^2 B_0)$, which is represented as $O((1 - \epsilon)^2(\log n)^2 B_0)$ as well. □

By setting ϵ appropriately, the algorithm can be executed in near linear running time and has a quality guarantee as well. The smaller ϵ indicates less computation cost but more unnecessary buckets, which means the worse result quality, vice versa.

3.3 Heuristic Algorithm Based on l-grid Partitioning

As described above, the computational cost of the approximate approach is still high, when the array size N and the number of buckets B is large. Considering the usability of visualization, which demands for immediate response of queries, we also propose a heuristic algorithm based on a more specific hierarchical partitioning, the *l-grid* hierarchical partitioning, which is introduced as follows.

The overall spatial region corresponds to the root of the l-grid tree structure at depth 0, every partition evenly divides the interval of target region into l sub-intervals for each dimension. For instance, the spatial V-optimal histogram is considered as a 2-dimensional case, each partition of the target region generates $l \times l$ sub-regions with the same dimension length. Figure 6 shows an example histogram construction by l-grid partitioning when $l = 2$.

The heuristic algorithm follows a a *top-down greedy* strategy to conduct the l-grid partitioning, until the number

Fig. 6 An example l-grid partitioning ($l = 2$)

of buckets is not less than B. We define the greedy criteria as *Max-Red*, which chooses a partition with the maximum error reduction at each time. As the example shown in Fig. 6 when $l = 2$, the first partition conducts with only one choice, which is to partition the whole region into four subregions. Then, the second partition conducts by comparing the four possible partitions of current buckets. Since the error reduction of the right-down bucket is the largest one, we further split this bucket into four subregions to construct the result histogram when $B = 7$.

The number of additional buckets Δb generated by each partition, is computed as $l^2 - 1$, if no empty cell exists, e.g., 3 additional buckets are generated when $l = 2$. Then, the total partition time is computed by $\frac{B-1}{l^2-1}$, since the maximum number of buckets is B. As the whole region is considered as 1 bucket, the first partition divide the whole region into l^2 subregions, further partition generates $l^2 - 1$ additional buckets as mentioned above. Therefore, the list of bucket numbers generated by each partition is shown as $(l^2 - 1) + 1, 2(l^2 - 1) + 1, \cdots, B$. For each partition, the greedy strategy is conducted by comparing all the buckets that can be split. As a result, the total computation cost is $O(B^2 l^{-2} t_E)$, where t_E is an upper bound cost of computing the error metric value of a given bucket.

4. Implementation on Array DBMS

Since scientific data typically have spatio-temporal structure, indexed arrays are often used to represent scientific data, while the traditional relational model cannot handle such array data [23]. Some array database management systems are developed to support scientific computing and online analytical processing (OLAP), such as RasDaMan [3], ArrayDB [2], SciDB [24], [26], [27], etc. In this paper, we implement the proposed histogram construction methods on SciDB, one of the state-of-the-art array DBMSs.

4.1 Overview of SciDB and Our Configuration

SciDB is a parallel database system based on the shared nothing architecture, and designed specifically for scientific data management and analysis. In SciDB, array data is organized on disks as *chunks*, which also work as the I/O units. Chunking ensures that logically co-located cell values are clustered into the same physical block, and enables efficient query processing. Chunks are distributed among all instances of a SciDB cluster. SciDB uses a variety of functions for chunk mapping, and the default is the hash-based distribution. During query execution, each instance node executes query locally with the partial array data, and transfers

intermediate results across nodes when global computing is necessary.

SciDB supports a variety of operators for array data, such as data extraction (e.g., *slice*, *between*) and aggregation (e.g., *aggregate*, *regrid*). For aggregate operators, a lot of aggregate functions (e.g., sum, min, and max) are supported. In addition to built-in operators and aggregate functions, users are also allowed to implement user-defined operators (UDOs) and user-defined aggregates (UDAs) using C++ language based on the plugin mechanism. Once a plugin operator is loaded into the SciDB library system, it can perform in the similar way with the built-in operators of SciDB for data processing. SciDB also support a python interface, SciDB-Py, that leverages data-parallel computing of scientific data on Python, while SciDB acts as a back-end database server. The details of implementation will be explained in the following section.

Our experiments are performed on a SciDB cluster, which consists of a coordinator server (CPU: Intel Xeon E5-2637 v4 @ 3.50GHz × 2, RAM: 128GB, OS: Ubuntu 14.04 LTS 64bit) and three worker servers (CPU: Intel Xeon E5620 @ 2.40GHz × 2, RAM: 32GB, OS: Ubuntu 14.04 LTS 64bit), each with three instances. There are totally 10 SciDB instances that work in parallel. The used version of SciDB is 16.9.

4.2 Implementation Details

The proposed methods, i.e., exact algorithm, approximate algorithm and greedy algorithm, are implemented on SciDB as UDAs, represented as *DPMerge*, *AprMerge* and *LGreedy*, respectively. The UDAs are executed by employing the *aggregate* operator that conducts aggregation on the input array. An example of the aggregate query is shown as follows:

```
aggregate(array_name,UDA(atr_name))
```

where *array_name* is the name of input array, and *atr_name* is the name of the aggregated attribute.

Using the UDAs described above, users are able to conduct the proposed histogram construction methods on any target array by setting it as the input array. Moreover, parallel computation is available by the *regrid* operator, which conducts aggregation on the sub-regions with specified block sizes at the same time. An example of the *regrid* operator is as follows:

```
regrid(array_name,blk_x,blk_y,UDA(atr_name))
```

where *blk_x* and *blk_y* represent the specified block sizes for each dimension. In addition, using the SciDB-Py interface, visualization and interactive analysis of histograms are also available.

5. Experiments

5.1 Objective datasets

The objective datasets consist of an evacuation simulation data in the event of large-scale earthquakes in Kantou area of Japan, a real dataset of taxis' trajectory data in Beijing [28], [29], and synthetic datasets with three different distributions. The simulation data, represented as Dataset-1, contains abount 194 millions records of evacuees' mobility data during 24 hours after an earthquake occurs. The real trajectory data, represented as Dataset-2, contains about 15 million data points of 10,357 taxis during the period of Feb. 2 to Feb. 8, 2008.

The records of Dataset-1 and Dataset-2 are both in the format of (id, time, x, y): id denotes user ID; time, a time stamp; x, y, the location of a user. We preprocess the two datasets by dividing the spatial regions with a maximum grain size of 4,096 × 4,096, while the number of objects in each cell is aggregated by every minute and ten minutes, respectively. The aggregated data resulting from the preprocessing are loaded to SciDB and are subjected to the query processing. The schemas of the generated spatio-temporal array stored in SciDB are represented as follows.

```
Dataset-1<Num:int64>
[X=1:4096;Y=1:4096;T=1:1440]

Dataset-2<Num:int64>
[X=1:4096;Y=1:4096;T=1:900]
```

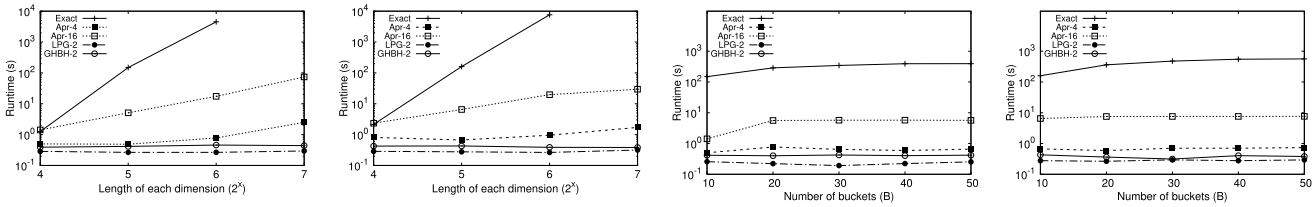
Both of the two above arrays consist of 3 dimensions as X, Y, T, and the attribute *Num* is the sum of objects of each grid cell.

In addition, we generate synthetic data that contains an array with the uniform distribution of interval $(0, 1)$, as well as others with the Gaussian distribution. We generate two Gaussian distribution array data with the same mean (0) and different variances (0.25 and 1), which is represented as Gaussian-1 and Gaussian-2, respectively.

5.2 Experimental Setup

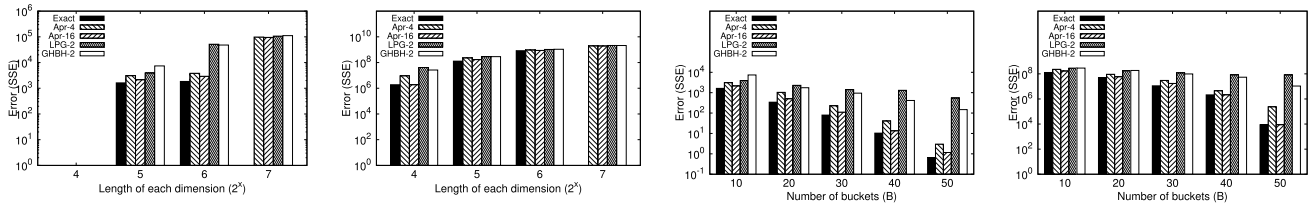
We perform extensive experiments to evaluate the quality and efficiency of the proposed methods on a SciDB cluster, the details of which is introduced in Sect. 4.1. First, we compare the proposed approximate and heuristic algorithms with exact algorithm, as well as *GHBH* (grid hierarchical binary histogram), which is proposed in [10]. Since the computation cost of the exact algorithm is extremely expensive, we generate array data with lengths from 16 to 128 for each dimension, and set the number of buckets varying from 10 to 50. Then, we conduct experiments on large datasets, each dimension length of which varies from 64 to 1024, to evaluate the scalability of the proposed algorithms and *GHBH* on efficiency and quality. In addition, we conduct experiments on synthetic data to measure the robustness of the proposed algorithms under different distributions.

The compared methods are represented in short as in Table 1. To evaluate the result quality of *Apr*, we set the parameter L_k as 4 and 16 to compare the errors of result histograms with ground truth, which is computed by the exact algorithm. For the l-grid heuristic algorithm, we set l as



(a) Dataset-1: varying n (B = 10) (b) Dataset-2: varying n (B = 10) (c) Dataset-1: varying B (n = 32) (d) Dataset-2: varying B (n = 32)

Fig. 7 Efficiency (Execution time)



(a) Dataset-1: varying n (B = 10) (b) Dataset-2: varying n (B = 10) (c) Dataset-1: varying B (n = 32) (d) Dataset-2: varying B (n = 32)

Fig. 8 Quality (Error)

Table 1 Symbols of compared methods

Proposed methods	Symbol
Exact algorithm	Exact
Approximate algorithm	Apr
Heuristic algorithm based on l-grid partitioning	LPG
Grid hierarchical binary histogram	GHBH

Table 2 Computation cost of proposed methods

Proposed methods	Time complexity	Space complexity
Exact	$O(N^{2.5}B^2)$	$O(N^2B)$
Apr	$O(N^{1+1.5\epsilon}B^2)$	$O(N^{1-\frac{\epsilon}{2}}B)$
LPG	$O(B^2l^{-2}t_E)$	$O(B)$

2, which is a general grid-based partitioning, i.e., quadtree-based partitioning. Also, we show the results of *GHBH* when parameter k is set as 2, which performs best. We use a hyphen to concatenate algorithm names and parameter settings, e.g., *Apr-4* means the approximate algorithm with an L_k of 4.

The quality of the algorithms is measured by the errors of result histograms in terms of the input array. We use the sum of squared error (SSE), which is defined in Eq. (1) to measure the quality. The efficiency is evaluated by the execution time of the algorithms. We evaluate the efficiency and quality of the proposed methods by varying the dimension length n of the computed array and the number of buckets B.

5.3 Comparing with Exact Algorithm

5.3.1 Evaluation of Efficiency

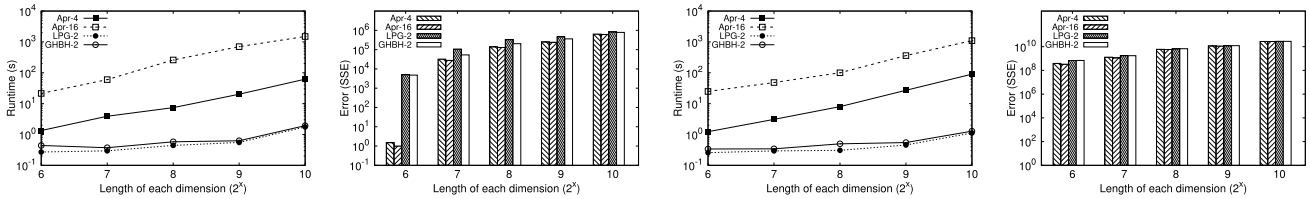
Figure 7 shows the execution time of the compared methods by varying the dimension length n (i.e., array size of $n \times n$), as well as the number of buckets B. The result shows that *Exact* is the most time consuming one, and it becomes inexecutable when n is larger than 128, as shown in Fig. 7 (a) and Fig. 7 (b). The reason of its unavailability is that, the generated candidate results by the dynamic programming algorithm exceeds the main memory. As shown in Table 2, the space complexity of *Exact* is $O(N^2B)$, which is the most expensive one than other methods.

The result also demonstrates that the approximate and heuristic algorithms are more efficient than *Exact*. For the approximate algorithm, *Apr-16* is much slower than *Apr-4*, since the large parameter L_k indicates larger ϵ of Theorem 2 in Sect. 3.2, which affects the time complexity of the approximate algorithm. Also, *Apr* runs slowly than the heuristic algorithms *GHBH-2* and *LPG-2*, but the distinction is minor when data size is small and the parameter L_k is small. As shown in Fig. 7, the execution time of *Apr-4* is close to *GHBH-2* when data size is small ($n < 128$), and it is not affected by the increase of the bucket number B. Moreover, we can improve the efficiency of *Apr* by decreasing the parameter L_k at the expense of quality.

On the other hand, the proposed heuristic algorithm (*LPG-2*) is faster than *GHBH-2*, and it is the most efficient one, even when $l = 2$. As the time complexity of *LPG* shown in Table 2, the larger the l is, the more efficient of *LPG*. Moreover, the increasing of bucket number B does not affect the efficiency of the approximate and heuristic algorithms, since B is much smaller than N.

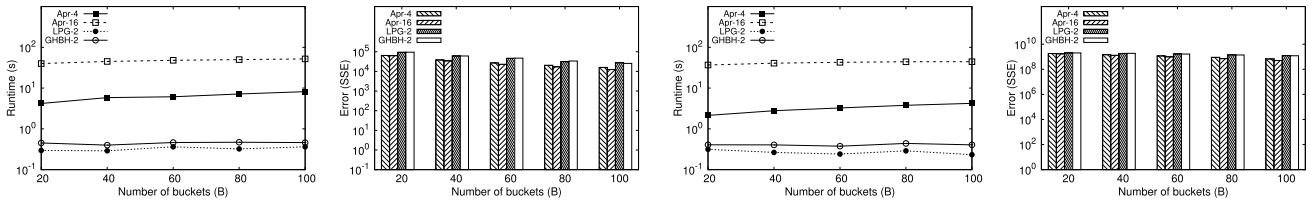
5.3.2 Evaluation of Quality

We also compare the corresponding errors of result histograms (i.e., SSE) as shown in Fig. 8. The errors increase as n increases or B decreases, which is intuitive. For Dataset-1 as illustrated in Fig. 8 (a), the errors of the compared methods are all 0 when n is 16, due to the small size of array. For the approximate algorithm, the result demonstrates that increasing the parameter L_k makes the result error decrease. It verifies Theorem 2 in Sect. 3.2, since large L_k indicates large ϵ, which means the high time complexity as well as



(a) Dataset-1: execution time (b) Dataset-1: error (c) Dataset-2: execution time (d) Dataset-2: error

Fig. 9 Varying n ($B = 50$)



(a) Dataset-1: execution time (b) Dataset-1: error (c) Dataset-2: execution time (d) Dataset-2: error

Fig. 10 Varying B ($n = 128$)

high approximation of results. Moreover, both of *Apr-4* and *Apr-16* perform better than *LPG-2* and *GHBH-2*, and the larger the parameter L_k, the closer the results of *Apr* to the ground truth.

Figure 8 (a) and Fig. 8 (b) show the influence of the array size n on the result errors. When data size is small (e.g., $n = 32$ in Dataset-1 and $n = 16$ in Dataset-2), the errors of *GHBH-2* and *LPG-2* differ slightly, which depends on the target data. As data size becomes larger, the errors of *GHBH-2* and *LPG-2* are almost the same, and their difference with *Apr* also becomes smaller. This indicates that, the larger the array size, the influence of the partitioning strategy becomes weaker on the result errors.

Figure 8 (c) and Fig. 8 (d) demonstrate that, the distinction between the approximate algorithm and heuristic algorithms becomes larger when B increases. The reason is that, for the heuristic algorithms *LPG* and *GHBH*, larger B indicates more times of partitions, which causes more imprecise on the results. In addition, when B is small ($B \leq 30$), *LPG-2* and *GHBH-2* perform similar with each other, while when B becomes larger, *GHBH-2* performs better than *LPG-2*, due to the flexible structure of *GHBH* with respect to *LPG*.

5.4 Comparing Proposed Methods and GHBH

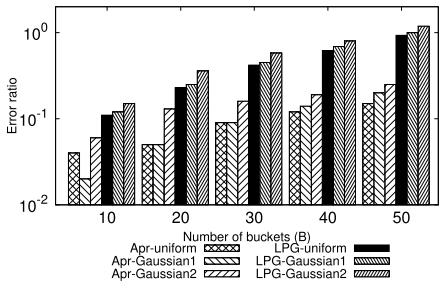
In what follows, we evaluate the efficiency and quality of the proposed approximate and heuristic algorithms as well as *GHBH*, using large size of data, where the length of each dimension varies from 64 to 1024, and the number of buckets changes from 10 to 100.

5.4.1 Varying the Array Size ($n \times n$)

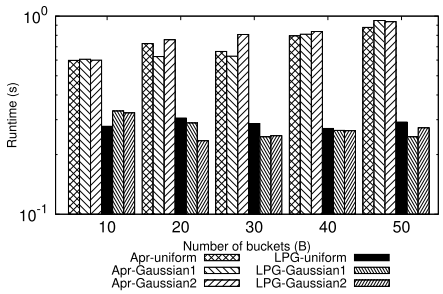
Figure 9 demonstrates the results of execution time and errors of the compared algorithms by varying n, which is the length of each dimension. In Fig. 9 (a) and Fig. 9 (c), the execution time of the approximate algorithm with different L_k increases with the increase of n. The smaller the L_k is, the faster the approximate algorithm. The reason is also the same with the one explained in the previous section, which is determined by the time complexity $O(N^{1+1.5\epsilon}B^2)$, where small L_k indicates small ϵ. The result also verifies that the efficiency of the heuristic algorithm *LPG* is the best one, and *GHBH-2* is slightly slower than *LPG-2*. On the other hand, Fig. 9 (b) and Fig. 9 (d) show that, the quality of results generated by *LPG-2* and *GHBH-2* is worse than the approximate algorithm, and the distinction between them becomes smaller when array size becomes larger. One of the reasons of the reduced distinction is that the larger the array size is, the more imprecise of the approximate algorithm is, according to Theorem 2 The other reason is that, the larger the array size, the influence of the partitioning strategy becomes weaker on the result quality.

5.4.2 Varying the Number of Buckets (B)

Figure 10 (a) and Fig. 10 (c) show the execution time of the compared methods are not sensitive to the number of buckets B. This can be explained their time complexity results in Table 2. Since the value of B is a small constant value (here, we consider it as $[10, 100]$), which is much smaller than array size N, the execution time is not sensitive to B. The errors of the results are shown in Fig. 10 (b) and Fig. 10 (d). The approximation of the approximate algorithm is also verified by the fact that *Apr-16* performs better than *Apr-4*. However, the distinction between them is not obvious since the array size is large. Meanwhile, the accuracy of the approximate algorithm is better than *LPG-2* and *GHBH-2*, and the larger the B is, the bigger the gap between them. This can be also explained by the imprecise of results caused by the increase of partitions as mentioned before. The result of Dataset-2 is less obvious than Dataset-1, because its larger data density, which leads to the imprecise of result histograms.



(a) Quality (error ratio)



(b) Efficiency (execution time)

Fig. 11 Data distributions, varying B ($n = 16$)

5.5 Performance on Different Data Distributions

Further, we perform experiments to measure the approximate algorithm (*Apr-4*) and heuristic algorithm (*LPG-2*) on different data distributions. The quality of the methods is measured by the error ratio with the ground truth E_0, i.e., $(E - E_0)/E_0$, where E is the SSE value of the result histogram, and E_0 is computed by *Exact*. Figure 11 (a) demonstrates that the quality of the two algorithms with Uniform data and Gaussian-1 (Gaussian data with smaller variance) is better than Gaussian-2 (the one with larger variance). This is because the error (SSE) of Gaussian-2 is intrinsically large, which leads to the result with more deviation. Meanwhile, the quality of algorithms on the uniform data and Gaussian-1 is similar. Furthermore, the execution time of them with the three distributions are similar with each other, as shown in Fig. 11 (b).

5.6 Discussion

The experimental results verify the efficiency and quality of the proposed approximate and heuristic algorithms. Comparing to the exact algorithm, the execution time of the approximate and heuristic algorithms is faster, and not as space consuming as the exact algorithm. Meanwhile, the quality and efficiency of the approximate algorithm, which is proved in Theorem 2, is also verified by the fact that the larger the parameter L_k is, the more accuracy of the result and more time consuming. We also observed that, when data size is small (e.g., $n < 128$), the efficiency of *Apr-4* is close to the existing heuristic method *GHBH-2*, with significant quality improvement. Moreover, the increase of the number of buckets B makes the distinction of quality more obvious.

For the heuristic algorithm (*LPG*), it runs faster than *GHBH* even when l is 2, with similar result quality. Moreover, *LPG* has good efficiency even if we increase the array size and the number of buckets. Although the accuracy of the results are worse than the approximate algorithm, the larger the array size, the distinction is less obvious. In addition, the efficiency of the approximate and heuristic algorithms is not affected by different data distributions, while the accuracy on data with the uniform distribution and Gaussian distribution with a small variance is better than Gaussian distribution with a large variance.

6. Related Work

A *histogram* is one of the popular approaches in summarizing large datasets and often used in database systems [14]. One of the applications of histograms is query cost estimation based on cardinality estimation of a query result [1], [4], [15], [22]. Researches in the literature focus on the histogram construction methods and the estimation of attribute values or frequencies, which is not the focus of this paper. For the construction of multi-dimensional histograms, as the cost of constructing optimal histogram is prohibitively large [18], existing techniques use heuristics to partition the data space into buckets, while they do not provide any guarantees on the quality of histograms.

MHIST [22] is based on a greedy strategy *MaxDiff(v,a)*, defined in [21], to partition the buckets in the most need of partitioning. Each partition divides the two consecutive values with the largest "area gap" [21] along one dimension. *MHIST* iteratively conducts partitioning until it reaches a given number of buckets. [10] proposes a *Grid Hierarchical Binary Histogram* (*GHBH*) based on the binary partition schema, which is the most relevant one to our work. In GHBH, each partition of a block is constrained to be laid onto a grid, which divides the block into a number of equal-size sub-blocks. This number is a parameter of the partition, given by k. The paper also proposes a greedy algorithm to construct a histogram with a given storage bound. Several greedy criteria are used, and the experimental results show that combination of the *Max-Var* and *Max-Red* performs best than other state-of-the-art techniques, such as *MHIST* and *GENHIST* [11].

Histograms considering *spatial and temporal* aspects are also studied for approximate query answering [7], [9], [20]. [9] proposed a framework for building and continuously maintaining spatio-temporal histograms. The histogram has an $N \times N$ grid structure, which is progressively refined by query feedbacks, to support continuous queries on dynamic streaming data. [20] studied the approximate query answering for future location-based query, using predictions based on spatio-temporal histograms. [7] evaluated several spatial partitioning techniques by their effects on the performance of spatial queries. However, their focused problems are different from ours and their methods

are not adaptive to solve our problem.

In this paper, we consider a general hierarchical histogram based on binary partition, as well as a more specific one (l-grid-based) for summarization. For the general hierarchical structured histogram, we propose a k-hierarchical histogram, which has different constraints on the buckets with GHBH mentioned above. Moreover, we proved and verified the approximation bound of k-hierarchical histogram with respect to the optimal histogram as mentioned in Sect. 3.2. The comparison experiments of our proposed methods and GHBH are shown in Sect. 5.

Another feature of our approaches is the use of an array DBMS, such as RasDaMan [3], ArrayDB [2], SciDB [24], [26], [27]. Since array DBMSs are now becoming popular in scientific computing and online analytical processing (OLAP), their effective use is an important issue in the database research area. We have used SciDB for implementing the proposed three algorithms. Using the feature of user-defined operators, we could easily implement our operators.

In addition, as data analytics has recently attracted increasing attention, data visualization is found effective in supporting interactive analyses and many studies on the subject are now underway. From the database perspective, technologies that can instantly visualize large-scale data or select data to be visualized are important. Since histograms can present the overview of data distribution in a summarized way, visualization is also another popular usage of histograms [6], [19]. For example, MuVE [6] visualizes data by bar graphs as a result of their consideration on the viewpoints that will concentrate data into specified conditions remarkably different from the whole data. SeeDB system for the visualization of databases, though intended for category attributes, is also closely related to this study [19].

7. Conclusions and Future Work

In this paper, aiming to realize the advanced analysis functionality of large-scale spatio-temporal array data, we defined a spatial V-Optimal histogram and propose histogram construction algorithms for hierarchical partitioning. We proposed exact and approximate algorithms with approximation bounds, as well as a heuristic approach to further speed up the histogram construction for data exploration. To improve the efficiency of the proposed methods on massive data, we implemented them on SciDB, the state-of-the-art array DBMS.

In order to verify the effectiveness and efficiency of our methods, we conducted experiments on the massive evacuation simulation data and real taxi data, as well as synthetic data with different distributions. Experimental results demonstrate that the proposed approximate and heuristic algorithms perform well in efficiency than the exact algorithm, and also verify the approximation quality of the approximate algorithm. The approximate algorithm performs best on quality with similar efficiency comparing to other heuristic algorithms when data size is small. As data size increases, the proposed heuristic algorithm performs best on efficiency with similar result quality with the approximate algorithm.

In this work, we treat the spatio-temporal data as a series of spatial data, and introduced aggregation and summarization approaches by constructing histograms on spatial data. However, methods for appropriately setting the time interval adapting to the requests of analysis is on demand. In future work, we consider to construct a three-dimensional spatio-temporal histogram that deals with both spatial and temporal dimensions, which has not been studied in the literature as far as we know. Moreover, we also intend to solve the problem of updating histograms for continuous queries of data exploration. To enable interactive analysis of large-scale spatio-temporal array data, developing techniques on array DBMS that improving the efficiency of query processing is also in demand.

Acknowledgments

This study was partly supported by the Grants-in-aid for Scientific Research (16H01722) and CREST: “Creation of Innovative Earthquake and Tsunami Disaster Reduction Big Data Analysis Foundation by Cooperation of Large-Scale and High-Resolution Numerical Simulations and Data Assimilations”.

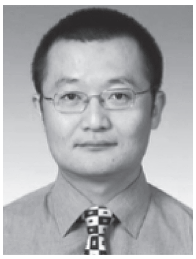
References

[1] S. Acharya, V. Poosala, and S. Ramaswamy, “Selectivity estimation in spatial databases,” In SIGMOD, vol.28, no.2, pp.13–24, 1999.
[2] A.P. Marathe and K. Salem, “Query processing techniques for arrays,” The VLDB Journal, vol.11, no.1, pp.68–91, 2002.
[3] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann, “The multidimensional database system RasDaMan,” In SIGMOD, vol.27, no.2, pp.575–577, 1998.
[4] N. Bruno, S. Chaudhuri, and L. Gravano, “STHoles: A multidimensional workload-aware histogram,” In SIGMOD, vol.30, no.2, pp.211–222, May 2001.
[5] F. d’Amore and P.G. Franciosa, “On the optimal binary plane partition for sets of isothetic rectangles,” Inf. Process. Lett., vol.44, no.5, pp.255–259, 1992.
[6] H. Ehsan, M.A. Sharaf, and P.K. Chrysanthis, “MuVE: Efficient multi-objective view recommendation for visual data exploration,” In ICDE, pp.731–742, 2016.
[7] A. Eldawy, L. Alarabi, and M.F. Mokbel, “Spatial partitioning techniques in SpatialHadoop,” Proc. VLDB Endow., vol.8, no.12, pp.1602–1605, Aug. 2015.
[8] A. Eldawy and M.F. Mokbel, “The era of big spatial data: A survey,” Found. Trends databases, vol.6, no.3-4, pp.163–273, Dec. 2016.
[9] H.G. Elmongui, M.F. Mokbel, and W.G. Aref, “Spatio-temporal histograms,” In Proc. 9th Int’l Conf. on Advances in Spatial and Temporal Databases Spatial and Temporal Databases, SSTD’05, vol.3633, pp.19–36, Springer-Verlag, Berlin, Heidelberg, 2005.
[10] F. Furfaro, G.M. Mazzeo, D. Saccà, and C. Sirangelo, “Compressed hierarchical binary histograms for summarizing multi-dimensional data,” Knowl. Inf. Syst., vol.15, no.3, pp.335–380, 2008.
[11] D. Gunopulos, G. Kollios, V.J. Tsotras, and C. Domeniconi, “Selectivity estimators for multidimensional range queries over real attributes,” The VLDB Journal, vol.14, no.2, pp.137–154, April 2005.
[12] V. Hristidis, S.-C. Chen, T. Li, S. Luis, and Y. Deng, “Survey of data management and analysis in disaster situations,” J. Syst. Softw., vol.83, no.10, pp.1701–1714, Oct. 2010.
[13] S. Idreos, O. Papaemmanouil, and S. Chaudhuri, “Overview of data

exploration techniques," In SIGMOD, pp.277–281, ACM, 2015.
[14] Y. Ioannidis, "The history of histograms (abridged)," In VLDB, pp.19–30, 2003.
[15] H.V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik, and T. Suel, "Optimal histograms with quality guarantees," In VLDB, pp.275–286, 1998.
[16] D. Lichtenstein, "Planar formulae and their uses," SIAM Journal on Computing, vol.11, no.2, pp.329–343, 1982.
[17] Y. Matias, J.S. Vitter, and M. Wang, "Wavelet-based histograms for selectivity estimation," In SIGMOD, vol.27, no.2, pp.448–459, ACM, 1998.
[18] S. Muthukrishnan, V. Poosala, and T. Suel, "On rectangular partitionings in two dimensions: Algorithms, complexity, and applications," In ICDT, vol.1540, pp.236–256, 1999.
[19] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina, "SeeDB: Visualizing database queries efficiently," Proc. VLDB Endow., vol.7, no.4, pp.325–328, 2013.
[20] H.K. Park, J.H. Son, and M.H. Kim, "Dynamic histograms for future spatiotemporal range predicates," Inf. Sci., vol.172, no.1-2, pp.195–214, 2005.
[21] V. Poosala, P.J. Haas, Y.E. Ioannidis, and E.J. Shekita, "Improved histograms for selectivity estimation of range predicates," In SIGMOD, SIGMOD '96, vol.25, no.2, pp.294–305, ACM, 1996.
[22] V. Poosala and Y.E. Ioannidis, "Selectivity estimation without the attribute value independence assumption," In VLDB, pp.486–495, 1997.
[23] F. Rusu and Y. Cheng, "A survey on array storage, query languages, and systems," CoRR, abs/1302.0103, 2013.
[24] Paradigm4: Creators of SciDB a computational DBMS, http://www.paradigm4.com/.
[25] X. Song, Q. Zhang, Y. Sekimoto, R. Shibasaki, N.J. Yuan, and X. Xie, "A simulator of human emergency mobility following disasters: Knowledge transfer from big disaster data," In AAAI, pp.730–736, 2015.
[26] M. Stonebraker, P. Brown, A. Poliakov, and S. Raman, "The architecture of SciDB," In SSDBM, LNCS, vol.6809, pp.1–16, 2011.
[27] M. Stonebraker, P. Brown, D. Zhang, and J. Becla, "SciDB: A database management system for applications with complex analytics," IEEE Computational Science & Engineering, vol.15, no.3, pp.54–62, 2013.
[28] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," In SIGKDD, KDD '11, pp.316–324, 2011.
[29] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-drive: Driving directions based on taxi trajectories," In SIGSPATIAL, GIS '10, pp.99–108, 2010.
[30] J. Zhao, Y. Ishikawa, Y. Wakita, and K. Sugiura, "Difference operators in simulation data warehouses," J. Disaster Res., vol.12, no.2, pp.347–354, 2017.

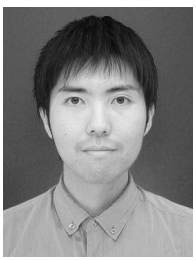
Jing Zhao is a Ph.D. candidate at Graduate School of Information Science, Nagoya University. She received her M.S. degree in Information Science from Nagoya University in 2015, and B.E. degree in Computer Science and Technology from Tianjin University of Science and Technology, China in 2012. Her research interests include spatio-temporal databases, scientific databases, and data warehousing. She is a member of DBSJ.

Yoshiharu Ishikawa is a professor in Graduate School of Informatics, Nagoya University. His research interests include spatio-temporal databases, mobile databases, scientific databases, data mining, and Web information systems. He is a member of the Database Society of Japan, IPSJ, IEICE, JSAI, ACM, and IEEE Computer Society.



Lei Chen is a professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include crowdsourcing on social networks, uncertain and probabilistic databases, Web data management, multimedia and time series databases, and privacy. He is a senior member of the IEEE.

Chuan Xiao is an assistant professor in Institute for Advanced Research, Nagoya University. He received B.E. degree from Northeastern University, China in 2005, and Ph.D. degree from The University of New South Wales in 2010. His research interests include data cleaning, data integration, textual databases, and graph databases. He is a member of DBSJ.



Kento Sugiura is a research associate in Graduate School of Informatics, Nagoya University. He received the B.S., M.S., and Ph.D. degrees from Nagoya University in 2013, 2015, and 2018, respectively. His research interests include data stream processing, uncertain data management, and spatio-temporal data processing. He is a member of DBSJ, IPSJ, and ACM.