International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:12, No:12, 2018

Simulation Data Summarization Based on Spatial Histograms

Jing Zhao, Yoshiharu Ishikawa, Chuan Xiao, Kento Sugiura

***Abstract*—**In order to analyze large-scale scientific data, research on data exploration and visualization has gained popularity. In this paper, we focus on the exploration and visualization of scientific simulation data, and define a *spatial V-Optimal histogram* for data summarization. We propose histogram construction algorithms based on a general *binary hierarchical partitioning* as well as a more specific one, the *l-grid partitioning*. For effective data summarization and efficient data visualization in scientific data analysis, we propose an optimal algorithm as well as a heuristic algorithm for histogram construction. To verify the effectiveness and efficiency of the proposed methods, we conduct experiments on the massive evacuation simulation data.

***Keywords*—**Simulation data, data summarization, spatial histograms, exploration and visualization.

I. Introduction

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 [5]. 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 [12]. Such kind of disaster simulations generate 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 [6].

In this paper, we focus on the *spatio-temporal simulation data summarization* for data exploration and visualization. Consider a motivating example as follows: an earthquake analyst intends to discover appropriate shelter locations and gives a query like "return the distribution of evacuees during the first day after the earthquake occurs". In general, this kind of query is used to understand the distribution of evacuees on a target spatial area during a given period of time for effective humanitarian relief and disaster management. Fig. 1 shows the distribution of evacuees during a period of time in Kantou area of Japan. The distribution is shown by heat maps, where different colors represent different densities of evacuees. In Fig. 1a, it is difficult to get an insight of the overall distribution of evacuees due to the large resolution (4096×4096). Typical exploration operators such as *drill-down* (zoom-in) and *roll-up* (zoom-out) of the data can not always solve the problem as well. As shown in Fig. 1b, even we zoom-out to a resolution of 256×256, it is still difficult for users to determine which part of the area should be focused on. This is because the target spatial area is quite large and data is loosely spreading over the region. In this case, a succinct summary of the overall region is considered to be useful.

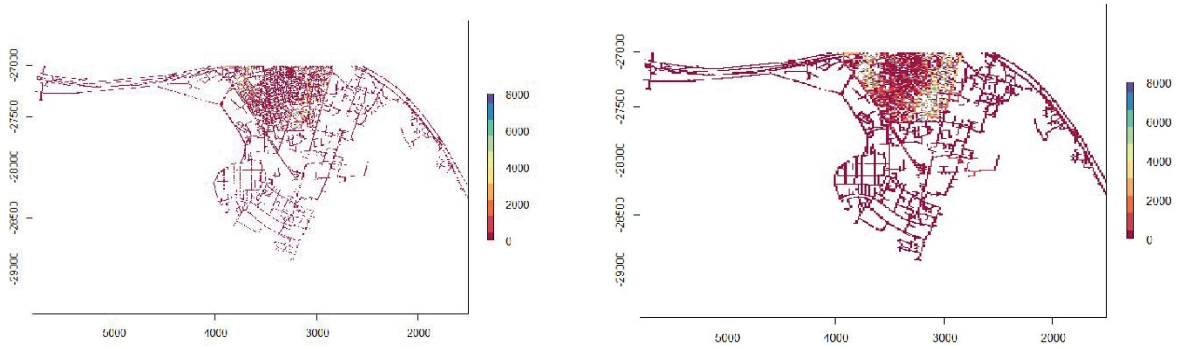
In order to catch an insight of spatio-temporal simulation data, we use the notion of *histograms* from the studies on selectivity estimation and query optimization in the database area [1], [3], [8], [11]. Histograms consist of buckets that covering the whole data domain, and each bucket should has close-to-uniform distribution. Such histograms provide a concise summary of the target data, and the efficiency and accuracy varies by the chosen partitioning types. For a two-dimensional array data, there are many types of partitioning. Here, we show the three commonly used ones in Fig. 2 [9]. Arbitrary partitioning is the one with no restrictions on the sub-regions as shown in Fig. 2a. It is obvious that the computation cost of arbitrary partitioning is the most expensive one, which is proved to be NP-hard [9]. The hierarchical one studied in this paper is shown in Fig. 2b. 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. 2c. Note that, it is a special case of the hierarchical one if the sibling nodes of the hierarchical tree are the same along one dimension.

In this work, we define a *spatial V-optimal histogram* to summarize the spatio-temporal simulation data, as an extension of one-dimensional *V-optimal histogram* [8]. We propose an optimal algorithm for the general *binary hierarchical partitioning*, and a heuristic algorithm using two greedy criteria for *l-grid partitioning*, which is a specific hierarchical partitioning to speed up the histogram construction further. We also perform experimental evaluation on evacuation simulation 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. (Section II).
- We propose an optimal algorithm based on binary partitioning as well as a heuristic algorithm to construct the *spatial V-optimal histogram*. (Section III).

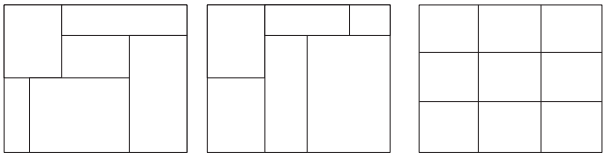
J. Zhao is with Graduate School of Information Science, Nagoya University, Furo-cho, Chikusa-ku, Nagoya, Japan (e-mail: zhao@db.ss.is.nagoya-u.ac.jp).
Y. Ishikawa and K. Sugiura are with Graduate School of Informatics, Nagoya University, Furo-cho, Chikusa-ku, Nagoya, Japan (e-mail: ishikawa@i.nagoya-u.jp, sugiura@db.ss.is.nagoya-u.ac.jp).
C. Xiao is with Institute for Advanced Research, Nagoya University, Furo-cho, Chikusa-ku, Nagoya, Japan (e-mail: chuanx@nagoya-u.jp).



(a) Resolution of 4096×4096

(b) Resolution of 256×256

Fig. 1 Evacuation distribution during a period of time



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

Fig. 2 Types of partitioning

- We conduct extensive experiments on massive simulation data to verify the effectiveness and the efficiency of the proposed methods (Section IV).

The rest of the paper is organized as follows: Section II introduces preliminaries and the problem definitions in this paper. Section III presents our method for constructing spatial histograms for simulation data exploration. Experimental results and analyses are reported in Section IV. Section V introduces the related work on histograms and data visualization. Section VI concludes the paper.

II. Preliminaries

We formulate the problem of constructing histograms for data summarization based on the *space-bounded V-optimal histogram* [8], 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 [8] proposed algorithms with a quality guarantee for *unidimensional V-optimal* histograms, the problem becomes totally different even in the two-dimensional case.

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. Related work on visualization such as MuVE [4] defines the usability using the number of views, as one of the objectives for view recommendation. Also, [2] studies the techniques of aggregation, sampling and filtering to reduce the size of the result for interactive visualization. One of the intentions of [2] is that, visualization with too many objects to draw on the screen is too dense to be useful to the user.

A. Problem Statement

Problem 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.

Problem Definition 2 (Spatial V-optimal Histogram)**.** *Given a spatial-temporal array* $\mathcal{A}$*, the type of partitioning* p*, 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 the partitioning type* p *that splits the whole spatial region into* B *non-overlapping buckets. Each bucket* b_l $(1 \leq l \leq B)$ *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* $b_l.area$*.*

The symbols used to formulate the *spatial V-optimal histogram* are shown in Table I. 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.

TABLE I
SYMBOLS AND THEIR MEANING

Symbol	Meaning
$\mathcal{A}$	A spatio-temporal array
R	The whole region of array $\mathcal{A}$
B	The number of buckets
$E()$	Error metric
p	Given partitioning type
C	The candidate result of possible subregions
r	A possible subregion
$maxB$	The maximum possible number of buckets
$curB$	The number of buckets of current result
H	The result histogram

Problem 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 $E(r) \leq E(R)$ for two buckets r and R, where $r \in R$. We say an error metric is *superadditive* if $E(r) + E(R) \leq E(r \cup R)$, where r and R are two disjoint buckets.

III. HISTOGRAM CONSTRUCTION ALGORITHMS

In this section, first we propose an optimal histogram construction algorithm based on dynamic programming for binary hierarchical partitioning. Since the computational cost of dynamic solution is extremely expensive, we also propose a heuristic algorithm with two greedy criteria, based on *l-grid partitioning* for the efficiency.

A. Optimal Algorithm for Binary Hierarchical Partitioning

The Optimal algorithm is based on dynamic programming using (2) as described below. 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 optimal algorithm is based on the following equation.

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

That is, the minimum error of region R is the minimum error among the possible partitions along x and y dimension with possible number of buckets, based on the monotonicity and superadditiveness of the error function. 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., $1, 2, ..., B-1$. 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. 3a, the whole region of $\mathcal{A}$ is represented as $R([1 \cdots 4], [1 \cdots 4])$. There are 3 possible partitions along each dimension, and each partition splits the whole region into two subregions, for each of which, further partitions also split it into smaller subregions.

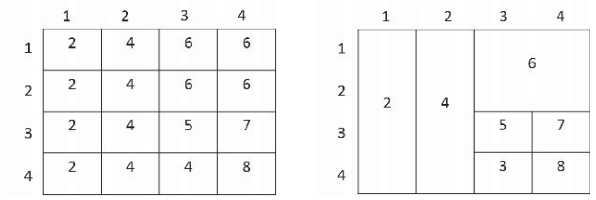
Algorithm 1: OptimalDP($\mathcal{A}, B, C, R$)

Input: $\mathcal{A}$: input array, B: limit number of buckets, C: the candidate list of possible subregions, $R = (x_{min}, x_{max}, y_{min}, y_{max})$: the whole region of $\mathcal{A}$
Output: Result histogram

```
1  foreach xl ∈ [1, x_max − x_min + 1] do
2      foreach yl ∈ [1, y_max − y_min + 1] do
3          foreach x_start ∈ [x_min, x_max] do
4              foreach y_start ∈ [y_min, y_max] do
5                  x_end ← x_start + xl − 1;
6                  y_end ← y_start + yl − 1;
7                  r ← (x_start, x_end, y_start, y_end);
8                  b ← Bucket(A, r);        // Generate a bucket with region r
9                  thisOpt ← {b};
10                 curB ← 1;        // Computed bucket number
11                 maxB ← min{B, r.size()};
12                 while curB < maxB do
13                     curB++;
14                     curOpt ← ComputeOpt (r, C, curB);
                       // Compute the optimal solution of r
15                     thisOpt ← thisOpt ∪ curOpt;
16                 C(r) ← thisOpt;
17 return C_B(R);
```

The pseudo-code of the algorithm is shown in Algorithm 1. We incrementally compute the subregions of the input array $\mathcal{A}$ by increasing the side lengths of each dimension (Line 1-7). 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. For a given subregion r, first we merge the whole region of r into one bucket (b) and compute the error metric value of b (Line 8). Then we compute the optimal solutions of r by increasing the bucket number until it equals to $maxB$ (Line 12-15). $maxB$ is computed by the maximal number of B and the element number contained in r (Line 11). For instance, given $B = 7$, region $r([1,2],[1,2]])$ shown in Fig. 3a contains 4 elements in total, so the maximum possible number of buckets is 4. The optimal solution of a region with a given bucket number is computed by ComputeOpt based on (2). The pseudo-code of ComputeOpt is shown in Appendix VI-A. Finally, the optimal result with bucket number B with the minimum error is returned. Fig. 3b shows an optimal histogram of A when the number of buckets $B = 7$, the error of which is 0.

1) Complexity Analysis: 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, which is computed in $O(1)$ by efficient methods such as holding the statistic information of subregions for further computation. Therefore,



(a) An example 2-D array A (b) A result histogram ($B = 7$)

Fig. 3 An example array with its optimal histogram

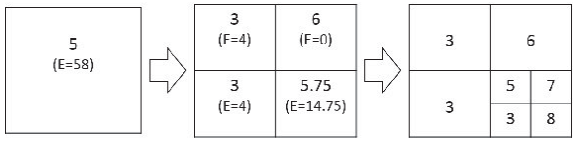


Fig. 4 An example l-grid partitioning ($l = 2$)

the total computation time is $O(N^{2.5}B^2)$, where N is the size of the computed array (i.e., $N = n^2$).

B. Greedy Approach based on l-Grid Partitioning

Since the proposed optimal algorithm for binary hierarchical partitioning is quite computationally expensive, we also propose a heuristic approach based on a more specific hierarchical partitioning, the *l-grid partitioning*, 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. Fig. 4 shows an example histogram construction by l-grid partitioning when $l = 2$.

As shown in Algorithm 2, the heuristic approach follow a *top-down greedy* strategy to conduct the l-grid partitioning, until the number of buckets is not less than B. We define two greedy criteria as *Max-Error* (Line 7-8) and *Max-Red* (Line 9-11). *Max-Error* conducts partitioning on the bucket with the maximum error, and *Max-Red* chooses the partition with the maximum error reduction.

For instance, Fig. 4 shows an example partitioning when $l = 2$ (i.e., *quadtree partitioning*). 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 (or 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 $B = 7$.

1) Complexity Analysis: 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

Algorithm 2: LGridGreedy($\mathcal{A}, B, R, G$)

Input: $\mathcal{A}$: input array, B: limit number of buckets, R : the whole region of $\mathcal{A}$, G: the greedy criterion
Output: Result histogram H

1 $H \leftarrow$ Bucket($\mathcal{A}, R$); // Initialize the result histogram with the root node
2 $curB \leftarrow 1$;
3 $optE \leftarrow 0$;
4 **while** $curB < B$ **do**
5 **foreach** $b \in H$ **do**
6 $thisE \leftarrow 0$;
7 **if** $G =$ Max-Error **then**
8 $thisE = b.error$;
9 **if** $G =$ Max-Red **then**
10 $C \leftarrow$ children(b); // Set of the children cells
11 $thisE = b.error - \sum_{c \in C} c.error$;
12 **if** $thisE > optE$ **then**
13 $nextPb \leftarrow b$;
14 $newBkts \leftarrow$ children($nextPb$);
15 $H \leftarrow H \setminus \{nextPb\} \cup newBkts$;
16 **return** H;

$l^2 - 1$ 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.

IV. Experiments

A. Experimental Dataset

The experimental dataset is the evacuation simulation data in the event of large-scale earthquakes in Kantou area of Japan. The simulation dataset contains about 194 millions records of evacuees' mobility data during 24 hours after an earthquake occurs.

The records are 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 dataset by dividing the spatial regions with a maximum grain size of $4,096 \times 4,096$, while the number of objects in each cell is aggregated by every minute and ten minutes, respectively. The preprocessed data consists of 3 dimensions as x, y, t, as well as an attribute num, which is the sum of objects of each grid cell.

B. Experimental Settings

We perform experiments to evaluate the efficiency and quality of the proposed methods, the symbols of which are shown in Table II. The efficiency is evaluated by the execution time of the algorithms, and the quality is measured by the errors of result histograms. We use the sum of squared error (SSE), which is defined in (1) to measure the quality. We check the effect of n, the length of each dimension, and B, the number of buckets, on the performance of each method. We implement the proposed algorithms in *C++* language, and

all experiments are conducted on an experimental server with Intel Xeon(R) CPU E5620 @ 2.40GHz $\times$ 2, RAM 32GB, OS Ubuntu 14.04 LTS 64bit.

TABLE II
SYMBOLS OF COMPARED METHODS

Proposed methods	Symbols
Optimal algorithm for binary partitioning	Binary-Opt
Greedy approach based on *Max-Red*	LG-MaxRed
Greedy approach based on *Max-Error*	LG-MaxE

C. Effect of n

We compare the execution time of *Binary-Opt* with heuristic approaches when increasing the array size, with constant B as well as increased B, in Figs. 5a and 5b, respectively. The result shows that, *Binary-Opt* is the most time consuming one comparing to *LG-MaxRed* and *LG-MaxE*. Moreover, the execution time of *Binary-Opt* increases rapidly as the increasing of n, especially when B increases. Note that, *Binary-Opt* becomes inexecutable when array size is 128×128, due to its large time complexity introduced in Section III-A1. In addition, both *LG-MaxRed* and *LG-MaxE* perform efficiently even when n increases, and *LG-MaxE* is slightly faster than *LG-MaxRed*.

On the other hand, we also compare the corresponding result errors (i.e., SSE) as shown in Figs. 6a and 6b. The errors increase as n increases, which is intuitive. Note that, when $n = 16$, the errors of the proposed methods are all 0 in Fig. 6a, since the array is small. The quality of *Binary-Opt* is better than *LG-MaxRed* and *LG-MaxE*, and the distinction is not affected by the change of array size. *LG-MaxRed* and *LG-MaxE* perform similar with each other when B is constant, while *LG-MaxRed* performs better than *LG-MaxE* with increased B. Based on this observation, it can be inferred that increasing B improves the quality of *LG-MaxRed*, while it does not affect *LG-MaxE*.

D. Effect of B

In what follows, we evaluate the effect of B on the efficiency and quality of the proposed methods. Figs. 7a and 7b demonstrate the result of varying B when array size is small ($n = 32$). As shown in Fig. 7a, the execution time of the three methods is not influenced by the increasing of B. Even though *Binary-Opt* performs slowly than the two heuristic approaches, its quality is better than them, and the larger the B is, the bigger the distinction between them, as shown in Fig. 7b. Moreover, *LG-MaxE* performs better than *LG-MaxRed* in quality with similar execution time, since the array size is small.

Considering the unavailability of *Binary-Opt*, we conduct experiments with large array size ($n = 128$) to compare *LG-MaxRed* and *LG-MaxE*. Figs. 8a and 8b show that, as the increasing of B the quality of *LG-MaxRed* is better than *LG-MaxE* with similar execution time. The reason of the bad performance of *LG-MaxE* is that, the larger the array size is, the greedy criterion performs worse, because the larger error does not mean the better choice of partition.

E. Discussion

The experimental results evaluate the efficiency and quality of the proposed methods, which can be concluded as follows. *Binary-Opt* is the most time consuming one, while performs best in quality, and the larger the B is, the distinction is bigger. Both of the two greedy criteria perform efficiently even when increasing the array size and the number of bucket. The quality of *LG-MaxE* and *LG-MaxRed* varies by the array size. The larger the array is, the better the quality of *LG-MaxRed* than *LG-MaxE*, and the distinction becomes larger when bucket number (B) increases.

V. RELATED WORK

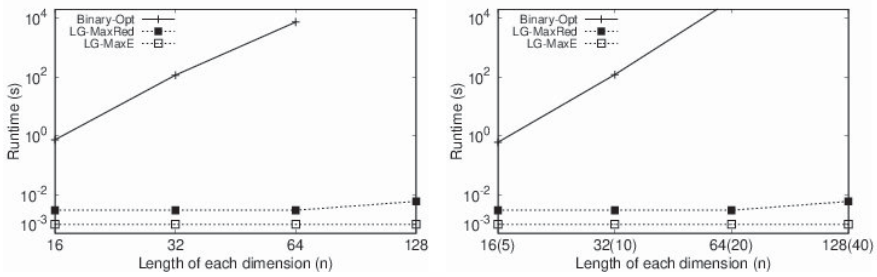
As data analytics has recently attracted increasing attention, data visualization is found effective in supporting interactive analyses and many studies on the subject are now underway. From the database perspective, technologies that can instantly visualize large-scale data or select data to be visualized are important. Since histograms can present the overview of data distribution in a summarized way, visualization is also a popular usage of histograms [4], [10]. For example, MuVE [4] 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 [10].

A *histogram* is one of the popular approaches in summarizing large datasets and often used in database systems [7]. One of the applications of histograms is query cost estimation based on cardinality estimation of a query result [1], [3], [8], [11]. 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 [9], existing techniques use heuristics to partition the data space into buckets, while they do not provide any guarantees on the quality of histograms.

In this paper, we consider a general hierarchical histogram based on binary partitioning, as well as a more specific one, the l-grid partitioning. Like the one proposed in [9], the optimal algorithm is also based on dynamic programming. However, [9] focuses on the error-bounded V-optimal histogram construction and ours works for space-bounded histogram construction. In order to enable interactive analysis of large-scale array data, we also propose a heuristic algorithm with two greedy criteria, based on l-grid partitioning.

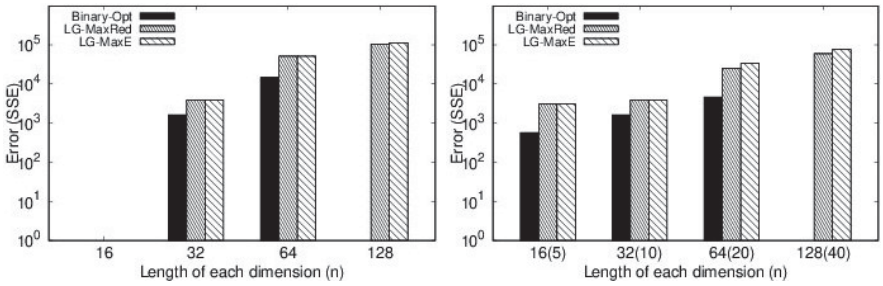
VI. CONCLUSION

In this paper, aiming to enable the advanced analysis functionality of spatio-temporal simulation data, we defined a spatial V-Optimal histogram and propose histogram construction algorithms for data summarization. We proposed an optimal algorithm for binary hierarchical histograms, as well as heuristic approaches based on two greedy criteria, to further speed up the histogram construction. In order to verify the effectiveness and efficiency of our methods, we



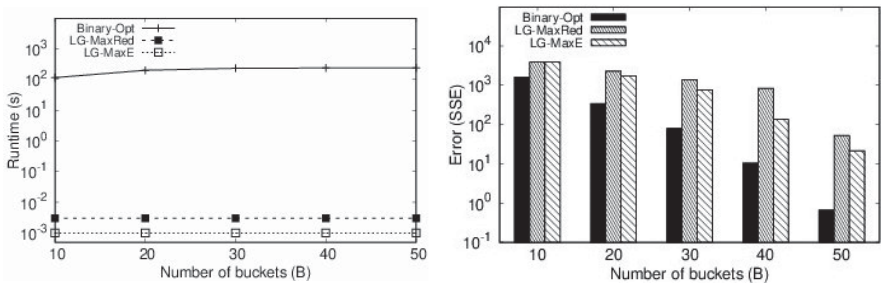
(a) Execution time ($B = 10$) (b) Execution time (increasing B)

Fig. 5 Varying n (execution time)



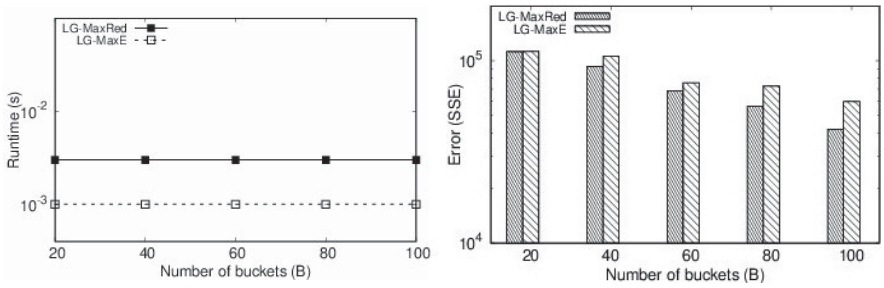
(a) Error ($B = 10$) (b) Error (increasing B)

Fig. 6 Varying n (error)



(a) Execution time (b) Error

Fig. 7 Varying B ($n = 32$)



(a) Execution time (b) Error

Fig. 8 Varying B ($n = 128$)

conducted experiments on the massive evacuation simulation data. Experimental results demonstrate that the quality of the optimal histogram for binary hierarchical histograms is the best one, while it is time consuming. The efficiency of the proposed heuristic algorithm is also verified by the experimental result.

APPENDIX

A. Pseudo-Code of ComputeOpt

The pseudo-code of ComputeOpt is shown in Algorithm 3. The basic idea is based on (2), i.e., the optimal result is computed by comparing the candidate results generated by partitions along x dimension (Line 2-8) and y dimension (Line 9-15), respectively. For each partition (e.g., px or py), the target region r is split into two subregions, candidate results are computed by assigning different number of buckets to each subregion. Finally, the optimal result with bucket number B with the minimum error is returned.

Algorithm 3: ComputeOpt(r,C,B)

Input: $r = (x_{min}, x_{max}, y_{min}, y_{max})$: the target region, C: candidate result of subregions of r, B: limit number of buckets
Output: Result histogram

1 $opt \leftarrow \emptyset$; // Initialize the optimal solution
2 **foreach** $px \in [x_{min}, x_{max} - 1]$ **do**
  // Partition along x dimension
3 $upR \leftarrow (x_{min}, px, y_{min}, y_{max})$;
4 $downR \leftarrow (px + 1, x_{max}, y_{min}, y_{max})$;
5 **foreach** $b \in [1, B]$ **do**
6 $thisOpt \leftarrow C_b(upR) \cup C_{B-b}(downR)$;
7 **if** $thisOpt.error \leq opt.error$ **then**
8 $opt \leftarrow thisOpt$;
9 **foreach** $py \in [y_{min}, y_{max} - 1]$ **do**
  // Partition along y dimension
10 $leftR \leftarrow (x_{min}, x_{max}, y_{min}, py)$;
11 $rightR \leftarrow (x_{min}, x_{max}, py + 1, y_{max})$;
12 **foreach** $b \in [1, B]$ **do**
13 $thisOpt \leftarrow C_b(leftR) \cup C_{B-b}(rightR)$;
14 **if** $thisOpt.error \leq opt.error$ **then**
15 $opt \leftarrow thisOpt$;
16 **return** opt;

ACKNOWLEDGMENT

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*, pages 13–24, 1999.
[2] L. Battle, M. Stonebraker, and R. Chang. Dynamic reduction of query result sets for interactive visualizaton. *2013 IEEE International Conference on Big Data*, pages 1–8, 2013.
[3] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: A multidimensional workload-aware histogram. In *SIGMOD*, pages 211–222, May 2001.
[4] H. Ehsan, M. A. Sharaf, and P. K. Chrysanthis. MuVE: Efficient multi-objective view recommendation for visual data exploration. In *ICDE*, pages 731–742, 2016.
[5] A. Eldawy and M. F. Mokbel. The era of big spatial data: A survey. *Found. Trends databases*, 6(3-4):163–273, Dec. 2016.
[6] V. Hristidis, S. C. Chen, T. Li, S. Luis, and Y. Deng. Survey of data management and analysis in disaster situations. *J. Syst. Softw.*, 83(10):1701–1714, Oct. 2010.
[7] Y. Ioannidis. The history of histograms (abridged). In *VLDB*, pages 19–30, 2003.
[8] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, pages 275–286, 1998.
[9] S. Muthukrishnan, V. Poosala, and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In *ICDT*, pages 236–256, 1999.
[10] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina. SeeDB: Visualizing database queries efficiently. *Proceedings of the VLDB Endowment*, 7(4):325–328, 2013.
[11] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, pages 486–495, 1997.
[12] 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*, pages 730–736, 2015.

Jing Zhao is a PhD 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 Information Science, 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.

Chuan Xiao is an assistant professor in Graduate School of Information Science, 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.