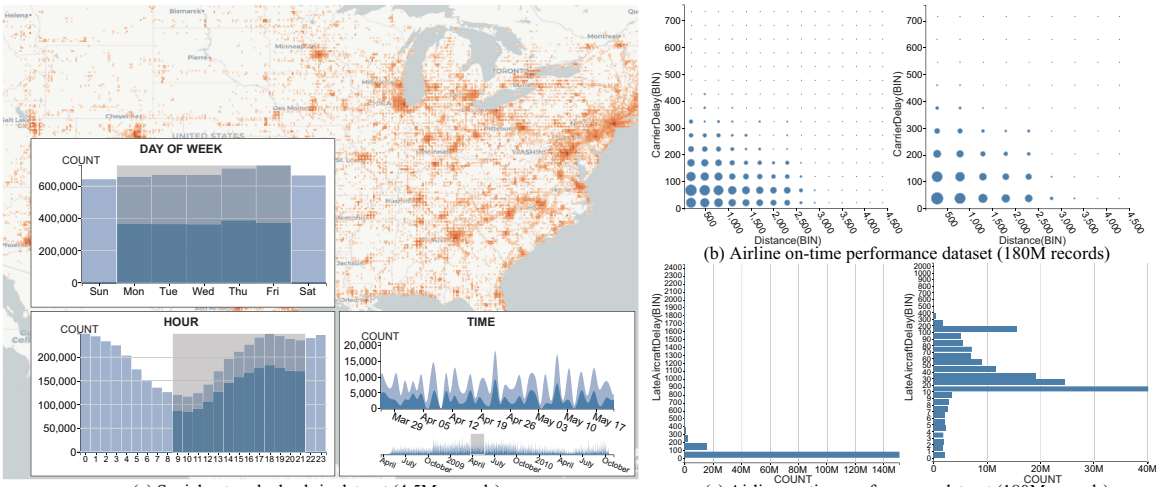
RSATree: Distribution-Aware Data Representation of Large-Scale Tabular Datasets for Flexible Visual Query

Honghui Mei, Wei Chen, Yating Wei, Yuanzhe Hu, Shuyue Zhou, Bingru Lin, Ying Zhao, Jiazhi Xia



(a) Social network check-in dataset (4.5M records)

(b) Airline on-time performance dataset (180M records)

(c) Airline on-time performance dataset (180M records)

Figure 1. RSATree facilitates fast answering of aggregate queries in large-scale tabular datasets while allowing flexible binning strategies. (a) A case built on a social network check-in dataset with 4.5 million records. A brushing and linking operation is performed by brushing workdays (Monday – Friday) and 13 hours of each day (9am – 9pm) for filtering. (b) Binned scatterplot created from an airline on-time performance dataset with 180 million records. The bin width can be freely modified and instant previews are shown. (c) Capability of RSATree to generate a histogram with varied bin widths by using the same dataset as (b). As shown on the left side of (c), the distribution of "LateAircraftDelay" is relatively unbalanced. Application of log-scale binning produces a recognizable histogram (right side). Conventional approaches cannot simultaneously achieve low response time and flexible binning strategy in the three cases.

Abstract— Analysts commonly investigate the data distributions derived from statistical aggregations of data that are represented by charts, such as histograms and binned scatterplots, to visualize and analyze a large-scale dataset. *Aggregate queries* are implicitly executed through such a process. Datasets are constantly extremely large; thus, the response time should be accelerated by calculating predefined data cubes. However, the queries are limited to the predefined binning schema of preprocessed data cubes. Such limitation hinders analysts' flexible adjustment of visual specifications to investigate the implicit patterns in the data effectively. Particularly, RSATree enables arbitrary queries and flexible binning strategies by leveraging three schemes, namely, an R-tree-based space partitioning scheme to catch the data distribution, a locality-sensitive hashing technique to achieve locality-preserving random access to data items, and a summed area table scheme to support interactive query of aggregated values with a linear computational complexity. This study presents and implements a web-based visual query system that supports visual specification, query, and exploration of large-scale tabular data with user-adjustable granularities. We demonstrate the efficiency and utility of our approach by performing various experiments on real-world datasets and analyzing time and space complexity.

Index Terms—Aggregate query, visual query, large-scale data visualization, R-tree, summed area table, hashing

1 Introduction

Exploratory data analysis (EDA) constantly involves huge size of datasets. Data items must be filtered and aggregated before encoding when visualizing a large and high-dimensional dataset due to limited number of screen pixels [28]. Particularly, data are visualized by charts, such as histograms, binned scatterplots, and heatmaps, which display the aggregations (e.g., count and average) performed in a sequence of axis-aligned subspaces (called bins) divided from the entire data space.

The parameters of these *aggregate queries*, such as filter conditions and bin width, should be frequently modified during EDA [59]. This condition poses two main challenges to the underlying query implementation. The first challenge is the capability to answer aggregate queries with low response time because high latency (more than 500 ms) may discourage user activity and decrease dataset coverage [43]. The second challenge is to support *arbitrary queries*, that is, aggregation for any specific range with flexible binning strategy. Analysts should use appropriate range filters and binning strategy (e.g., equi-width and equi-data) based on their requirements to observe patterns well. However, the two requirements are rarely supported simultaneously.

Many studies have focused on reducing the response time for aggregate queries. Data cubes are a commonly used approach, which precompute answers to possible aggregate queries with fast response [25, 42, 44, 49]. However, the queries are still limited to the predefined binning schema. This limits the flexibility of visual designs and the breadth of analysts' exploration. Support for high flexibility requires considerable preprocessing, which results in large storage consumption.

To reduce unnecessary storage, thereby avoiding huge storage cots, preprocessing should be systematically designed on the basis of usage

- *H. Mei, Y. Wei, Y. Hu, S. Zhou, B. Lin, and W. Chen are with The State Key Lab of CAD & CG, Zhejiang University. E-mail: {meihonghui, weiyating, cadhyz, zhoushuyue, linbingru}@zju.edu.cn, chenwei@cad.zju.edu.cn.*
- *Y. Zhao and J. Xia are with Central South University. E-mail: {zhaoying, xiajiazhi}@csu.edu.cn.*
- *Wei Chen is the corresponding author.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

scenarios [35, 42, 44, 49, 63]. Thus, obtaining accurate results is not constantly required when conducting exploratory analysis. In some scenarios, the answers may be slightly inaccurate in exchange for a quick response. For example, an approximate preview of the resulting chart during user interaction (e.g., dragging a slider) is shown, and accurate values are computed when the interaction stops. Such an idea is extended by the concept of *approximate query answering* [1, 13, 41]. Thus, statistical summaries of the data can be precomputed to provide answers to aggregate queries. Poosala et al. [51] used histograms to summarize the data distribution, in which requested aggregation of corresponding subcubes could be estimated. Response time and storage consumption can be remarkably reduced by allowing certain bounded errors, which support arbitrary queries.

The concept of approximate answering has inspired us to construct and leverage RSATree for a distribution-aware data representation of a large-scale tabular dataset with fast approximate query answering. In the design of RSATree, we mainly consider the support of arbitrary range queries and flexible binning strategy with low response time. First, the preprocessed data representation should be able to answer the aggregate query with approximately constant response time regardless of the specified range. Second, RSATree is designed to enable flexible binning strategy rather than support of only equi-width binning. Third, approximate answers are tolerated, but the error rate should be maintained at an overall low rate. We design a distribution-aware data representation to reduce storage consumption and improve accuracy. Data distribution allows preprocessing with adaptive granularity and can remarkably reduce storage costs and maintain an overall low error.

We support arbitrary queries through auxiliary data representation based on *integral histograms* (IH) [52], which are a useful data structure that supports efficient approximate range query. IHs extend *summed area table* (SAT) [20] and can calculate an approximate data distribution under constant time in any region of discretized Cartesian data space. We adopt R-tree to divide the data space into subspaces and preprocess them with different granularities to achieve a balance between the accuracy of the queries and the storage cost of the precomputed structures. In addition, we employ *locality-sensitive hashing* (LSH) [21] functions, which support approximate KNN search in a high-dimensional space, to perform point-in-range search and range overlapping test. LSH functions enable random access to data with a reduced time complexity by reformulating the data organization.

We design and develop a web-based visual query system for high-dimensional tabular datasets using RSATree to determine its usability and efficiency. Instant visual feedback to frequent and continuous user interactions, such as specifying a filter range and changing bin settings, is enabled by precomputing and leveraging an RSATree structure in the entire process. Moreover, we propose a novel interaction scheme called *scale alignment*, which can remarkably improve the accuracy of the results. We evaluate its performance and utility on the basis of various experiments performed on real-world datasets.

In summary, the main contributions of this study are as follows: First, a novel data representation called RSATree, which supports flexible approximate query of large-scale tabular datasets, is designed and developed. Second, a web-based interactive interface, which leverages RSATree to meet the requirements of different scenarios, is proposed.

2 Related Work

2.1 Visual Query

Visual query plays a central role in visual analysis. It answers the domain of interests in a dataset through visual representations. In comparison with standard data queries that use relational languages to perform a search in relational databases, visual query languages are used to construct external representations that can be easily perceived by analysts [12, 19]. Visual query enables efficient access to valuable information in a database, thereby allowing analysts to explore datasets and focus on valuable items. This process requires queries to be efficiently performed and frequently iterate via dynamic query [57].

A direct means to perform a visual query is to allow analysts to specify a visualization form in an available selection list, as conducted by many existing visual exploration tools [46], such as VQE [22], Visage [54], and Tableau (formerly Polaris [58]). Query results are displayed by the selected visualization form. Other visualization tools, such as Spotfire [3], allow analysts to make a visual query by interacting on predefined visualizations, such as brushing and zooming on a map. In this manner, visual queries are intuitively performed.

The execution of a visual query frequently produces implicit *aggregate queries* in databases. They are performed on tabular datasets with multiple *dimensions*. For example, analysts should initially select particular dimensions and define an aggregate *function* to create a visualization of a car dataset. In the case of creating a bar chart, analysts should display the average (*function*) horsepower (*dimension*) of cars grouped by different cylinders (*dimension*). During visual analysis, visual queries frequently iterate among different parameters; this process requires an instant answer for the corresponding *aggregate query* performed in the database.

However, an underlying database management system requires massive operations over terabytes of data stored on hard disks [16], which results in a long execution time before queries are completed and a precise answer is returned. Previous research shows that most analysts prefer fast approximate answers rather than time-consuming precise answers when instant feedback is not available [69]. This requirement is met by applying approximate query answering, which has been employed by some online analytical processing (OLAP) systems [13, 51]. Approximate query effectively reduces the response time required for complex queries by utilizing different types of strategies, such as sampling- [1], histogram- [14, 45], and wavelet-based [13, 47] techniques.

Sampling is a widely used data abstraction technique to support visual abstraction [10, 17, 68]. Sampling maintains the characteristics of the data with few samples. Uniform sampling provides a simple solution but cannot constantly handle datasets with a skewed distribution [15]. To address this problem, different non-uniform sampling methods, such as visualization-aware sampling [50] and sampling with ordering guarantees [36], are used to retain the database structure at different levels of visual abstraction (e.g., during zooming). Real-time queries can be achieved through progressive data processing and presentation, such as VisReduce [53], DICE [34], sampleAction [24], and SeeDB [60], by leveraging an incremental sampling technique [32, 33].

Meanwhile, approximate queries performed on data cubes [5] have been conducted to obtain a remarkably reduced response time with degraded accuracy in exchange. Statistical summaries of the data are precomputed to provide answers to aggregate queries on subdata cubes. Poosala et al. [51] used histograms to summarize data distribution, in which the requested aggregation of corresponding subcubes can be estimated. IHs [52] are a useful data structure that supports efficient approximate query. They extend SAT [20] and enable the computation of histograms of all possible regions in Cartesian data space to be executed under constant time. However, the result of precomputation may be relatively memory-consuming, especially for high-dimensional datasets. Loading data on demand [6] is a possible solution. In designing such methods, space partitioning trees [4] are used to create an index of data space, which preserves the spatial distribution (e.g., quadtree [55] or k-d tree [9]), value distribution (e.g. R-tree [26]), or both (e.g. MRA-tree [38]). The histogram-based approximation strategy has inspired us to propose a novel data representation named RSATree for supporting efficient visual query in large-scale tabular datasets.

2.2 Interactive Visualization of Large Datasets

Various interactive visualization systems have been implemented to support efficient visual exploration of large datasets by performing data and visual abstraction techniques.

Studies have extended the concept of data cube that precomputes hierarchical binning and aggregation for multiscale visualization. Profiler [35] recommends binned views for anomaly detection. The preprocessed data cube is loaded into memory to support scalable brushing and linking. ImMens [44] utilizes the parallel computing capability of GPU to improve the performance of handling precomputed tiles of data cubes that are stored as textures. Nanocubes [42] use a well-designed

indexing scheme to reduce the size of the data cube. Hashedcubes [49] extends Nanocubes with a more compact representation and a considerably simpler implementation. On the basis of Nanocubes, Gaussian Cubes [63] further support interactive modeling, such as linear least squares and principal component analysis, by storing multivariate Gaussian rather than simple aggregation (e.g., count). BigIN4 decompose high-dimensional queries into low dimensional ones and gives approximate answers to reduce storage consumption of cubes [41]. These concepts have an outstanding performance on real-time visual exploration but still possess several limitations. Their precomputation schema is fixed, and their capabilities to answer visual queries are limited. The flexibility of users for visual exploration is diminished in exchange for high performance and instant interactive response.

Another approach to accelerate visual query is by providing approximate query answers. Analysts prefer a fast and approximate answer rather than an exact answer in many situations. Such an idea can be enhanced by applying *progressive visual analytics* [23, 69], which incrementally processes the queries and provides a dynamic tradeoff between the result accuracy and response time. Generally, progressive systems apply sampling-based computation with increasing sample rate to produce accurate results with time. Progressive visual analytics has been proven to provide better insight than typical visualizations that process the entire dataset before displaying the result [69]. In practice, progressive systems should estimate and depict the uncertainties, which are usually confidence intervals, in the current calculation process [24, 29]. Other efforts have been conducted to select the best data subset to be initially refined [53] or prune the possible candidates of queries by using decision-making strategies [60].

Generally, these systems aim to compute and present the most valuable data to users by allowing them to steer the exploration process and refine the query range and presentation method. Users' areas of interests in the dataset should be identified [27, 66], and proper views should be selected [8, 65]. RSATree naturally supports these methods through a progressive exploration process. Meanwhile, RSATree can support flexible specification on data and views, which is useful for exploratory navigation and analysis in large spaces (e.g., Voyager [64]).

3 Design Methodology

In this section, we present our methodology for designing the data representation that supports flexible approximate query of a large-scale tabular dataset. We discuss the design considerations and raised challenges by investigating several usage scenarios.

3.1 Scenarios

Several typical scenarios occur when working with large-scale tabular datasets through interactive analysis tools. As an example, Figure 1(a) shows a common visual analytics system for spatiotemporal data [18, 31, 42, 67, 70]. The main body of the system interface consists of a map with overlaid heatmap and statistical charts that display related attributes, such as histograms and line charts. Common interactions include panning and zooming on the map and filtering by brushing or selection in analyzing such spatiotemporal data.

Analysts assume that the map can be quickly refreshed during panning and zooming. Flexible and continuous zooming may be helpful, because the effect of heatmap representations depends on reasonable binning. Bins that are very fine may fail to capture the distribution, while bins that are very coarse will lose most details. The lack of continuous transition during zooming can also confuse analysts. Moreover, analysts may select any arbitrary range as a filter in performing filtering operations. All other charts are quickly updated when filtering, which is a common brushing and linking operation. Instant previews during such operations can provide a good exploration experience. At this point, a fuzzy result is also acceptable in exchange for fast exploration, while the error rate of the preview must be controlled within a certain range without affecting data investigation.

Besides the smooth exploration experimence, flexible visual representation methods are also required [40, 46, 64]. For example, a simple equi-width binning cannot guarantee desired charts that can exhibit a particular pattern due to skewed data distribution. At this point, analysts require considerable binning strategies, such as a log-scale binning, to produce a well-distributed histogram (Figure 1(c)).

3.2 Design Considerations

We have identified some usage scenarios to determine the requirements when working with EDA tools. We obtain the following design considerations by summarizing the requirements and combining our experience.

R1. Answer arbitrary range queries. We consider a type of query to the data cube called *range queries* to support the brushing and linking operation properly. Range queries request aggregation within a specified contiguous range in the domains of involved dimensions. The underlying data structure should be designed to answer any arbitrary range query due to the importance of data coverage during exploratory analysis.

R2. Flexible binning strategy. An important step in EDA is to find an appropriate binning strategy for aggregation. The selection of the width and number of bins is related to whether the characteristics and patterns of data can be correctly displayed. Most of the previous work only support a predefined equi-width binning strategy, which is useful but frequently insufficient. A flexible binning strategy should be supported to provide a good analysis.

R3. Low accuracy loss. We can use approximate queries that can sacrifice accuracy to reach the goals that are otherwise difficult to achieve, such as fast response speed and low storage consumption. However, such losses must be limited within a reasonable range. The degree of tolerance is determined on the basis of the usage scenario. In addition, the introduced uncertainty should be presented to the user through appropriate visual design.

R4. Low response time and low storage consumption. Fast response is the primary goal that should be achieved to provide a good exploratory analysis environment. Moreover, low storage consumption is important because it allows the entire precomputed data structure to be loaded in the main memory for high access efficiency.

3.3 Design Challenges

A clear picture of our data representation can be observed after organizing possible usage scenarios and design requirements. We construct an RSATree based on such requirements. However, we still experience many challenges based on three aspects.

The first challenge is answering arbitrary aggregate queries with low response time **(R1, R4)**. We follow the common practice and calculate a data cube that provides a multidimensional summarization of the raw data and allows fast access to aggregated results. However, conventional data cubes rely on rollup operations and traversal, which is time-consuming, because the range may cover a large number of values when answering range queries. In summary, the first challenge is to modify the data cube representation to enable aggregate queries over arbitrary ranges efficiently.

The second challenge is optimizing storage consumption while allowing a flexible binning strategy **(R2, R4)**. A data cube consumes considerable storage space, especially with the increase in the resolution and number of dimensions. Moreover, query answering is limited by the predefined binning schema due to the nature of the data cube. Therefore, high flexibility of binning strategy requires considerable preprocessing and fine granularity, which leads to considerable storage consumption problems. Approximate answering can alleviate such a problem to some extent. Nevertheless, an optimized preprocessing design that minimizes space consumption is still required, which is our second challenge.

Meanwhile, approximate answering creates a third challenge, which is reducing the effects of inaccurate answers **(R3)**. Inaccurate answers are allowed to accelerate the response and reduce storage consumption, thereby improving usability in actual usage scenarios. However, the tolerance for errors is limited. Particularly, two aspects should be mainly considered. One is the reduction of overall error level by systematically designing the precalculation and query processes. Second is the data structure adjustment to minimize the influence on results (such as generated statistical charts) when the overall error is constant.

4 RSATREE

We design and implement a novel data representation called RSATree, which adaptively approximates the aggregated values of the underlying dataset. The response time of aggregate queries is remarkably reduced by placing the precomputed RSATree structure into memory and querying for the approximation at controllable and low error rates.

4.1 Representation

RSATree is a precomputed data structure used to support efficient aggregate query for large-scale tabular data. The design of RSATree is enlightened on the basis of the observation that the data points of a multidimensional dataset are not uniformly distributed; thus, piles of data points can be packed in which their spatial similarity is preserved and summarized at a controllable information loss.

Consider a high-dimensional tabular data cube $\mathcal{V}$ with n dimensions $\mathcal{D} = \{d_1, \cdots, d_n\}$. An RSATree reformulates $\mathcal{V}$ into p small non-uniform data cubes with different levels of details based on the distribution of $\mathcal{V}$, which is denoted as $\mathcal{V}' = \{v'_1, \cdots, v'_p\}$. As shown in Figure 2, an RSATree is basically a nested three-level representation that flattens the input dataset. The top level is the index of partitioned spaces by LSH, the next level is the IH, and the innermost layer contains the *feature descriptor* that represents the underlying data points. Each item in the upper level is constructed by the items in the next lower level, thereby forming a nested structure

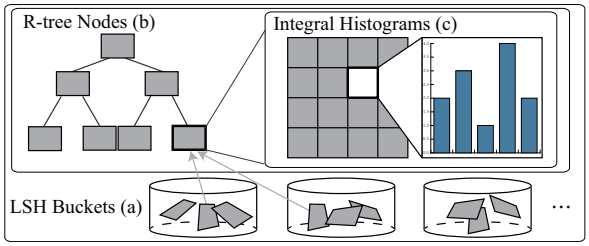


Figure 2. Nested RSATree representation: (a) LSH buckets used to store similar leaf nodes of an R-tree (b). Each node stores a set of IHs (c).

In the first level, each $v'_i \in \mathcal{V}'$ is grouped into an LSH bucket. The group result is determined on the basis of the applied LSH functions. With the arrangement of coherent subspaces, buckets of hash tables ensure an efficient spatial locality by grouping similar (v'_i, v'_j) pairs together. Thus, operations imposed on spatial neighborhood are empowered, such as range query.

In the second level, a set of IHs H is calculated to construct all the subspaces $\{v'_i | v'_i \in \mathcal{V}'\}$. Each $h_i \in H$ is a cube, where each cell contains a feature descriptor of all data points in the cell (contents of the third layer). After preprocessing, IH can quickly respond to range queries and return a feature descriptor of all data within a certain range.

The last level is the feature descriptor, which takes many forms. The simplest form is by directly recording the aggregated values of the underlying data. This form works for the measures that are distributive (e.g. *count* and *sum*) or algebraic (e.g. *mean* $=$ *sum*$/$*count*). For other measures (e.g. *median*), RSATree can estimate the answer by recording the data distribution on the dimensions with statistical histograms. This form is used in the original IH; however, we have made some adjustments to suit our algorithm.

The nested RSATree representation is a reformulation, abstraction, and simplification of the input data. The elements of an RSATree are placed in a hybrid linear tree structure, which supports random access and maximizes the query performance. Particularly, the following functionalities are provided: First, aggregate query efficiency is improved by searching the approximation of data distribution, where time complexity is independent of the number of data points. Second, efficient online visual query is achieved by reducing the storage consumption, which is independent of the number of data points. Third, random access to data is enabled when querying on large-scale datasets.

4.2 Construction

Figure 3 illustrates the construction of an RSATree. For a given tabular dataset, we initially partition its data space into multiple subspaces with different granularities based on data distribution. The approximation of each subspace is computed and stored to support efficient aggregate query, which can be used to estimate the distribution of input data. Approximation sets are then re-organized into a compact storage. All these computations are preprocessed, and a progressive construction scheme is applied to reduce its time and space complexity.

4.2.1 Space Partitioning

IHs enable range query to be performed under an approximately constant time at a huge storage cost ($O(N_1 \times \cdots \times N_d \times K)$, where $N_1, \cdots, N_d$ are the number of bins of each dimension, and K is the number of bins in the histograms). We partition the entire data space into subspaces with different granularities (Figure 3(a)) and compute IHs for each subspace (Figure 3(b)) instead of compressing the produced histograms, as proposed by previous studies [39].

We use R-tree to partition the space. The core idea of R-tree is to place nearby nodes under the same parent, which is represented as the minimum bounding rectangle of all the nodes it contains. The rectangle of each leaf node represents an object (in RSATree, each object is a data point represented as a rectangle with side lengths of zero). All non-leaf nodes are the aggregation of data points, and the increasing number of points are aggregated at high levels. Each level can be assumed as an approximation of the dataset. The leaf level is the finest-grained approximation (completely accurate), and the coarser the approximation is, the higher the level will be. We adopt the R*-tree [7] strategy among multiple R-tree variants. This strategy outperforms other existing R-tree variants with a minimization of coverage and overlap of the partitioned result, which fits our requirements.

As the core of RSATree, the distribution-based partition of the data space determines the approximation quality of the input data. Our goal is to make the data points as evenly distributed as possible within each subspace. Thus, the proper granularity can be chosen for each subspace, making the underlying data points be represented with less storage space and minimal loss. Therefore, we modify the R*-tree strategy for inserting points by taking into account the density changes within each node. To determine which branch the newly inserted node should fall into, the original algorithm selects the branch with the smallest change in area, that is, $\arg\min_{node}(area_{new} - area)$. We also consider the density, which changes from $\frac{n}{area}$ to $\frac{n+1}{area_{new}}$. We multiply the ratio of the density change by the area change as the final new optimization target, that is, $\arg\min_{node} \frac{n}{n+1} \times \frac{area_{new}}{area} \times (area_{new} - area)$. It can be simplified as $\arg\min_{node} \frac{area_{new}}{area} \times (area_{new} - area)$ because n is always very large.

In this manner, the divided subspaces have the following features. First, most of the empty spaces are eliminated. Second, few overlaps exist with one another. Third, each subspace contains a similar number of data points, which causes data-intensive regions to have fine granularities and vice versa. Fourth, the data points contained in each subspace are distributed as evenly as possible. This feature allows the partition to capture the distribution characteristics of the original data points well. Then, we begin to build an approximation of underlying data points in each partitioned subspace.

4.2.2 Building Integral Histograms

Integral histograms [52] is an extension to summed area table [20]. As shown in Figure 4(a), an SAT can generate the sum of values in an arbitrary rectangular area of a data grid in constant time. The value in a grid at position (x, y) of an SAT is the sum of all grids in the rectangle bounded by $(0, 0)$ and (x, y). IHs summarize the distribution of data points falling in each grid rather than storing a single scalar in each grid. This process efficiently returns the answer of an aggregate query by computing the histograms of all data points within the query range in a manner similar to computing the sum of a rectangular area via SAT [14, 45].

Formally, for a d-dimensional dataset binned by $N_1 \times \cdots \times N_d$ grids and summarized by histograms with B bins, the IH $H(x_1, \cdots, x_D, b)$, where $x_1, \cdots, x_D$ are indices of bins on different dimensions, and b is

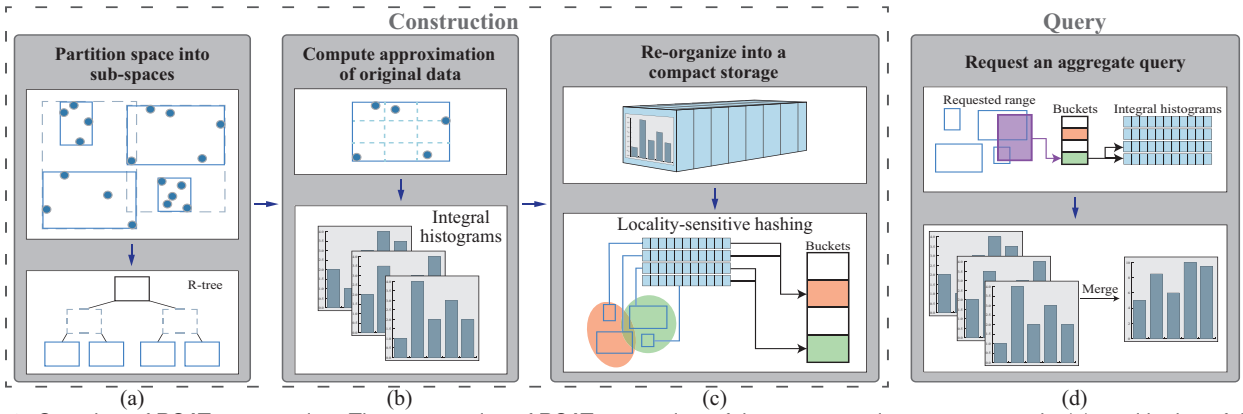


Figure 3. Overview of RSATree operation. The construction of RSATree consists of three consecutive stages, namely, (a) partitioning of the data space into subspaces on the basis of data distribution using R-tree; (b) computation of IHs as the approximation of the original data for each subspace; (c) and storing and indexing of IHs by LSH, thereby preserving the spatial coherence of subspaces. (d) Possible subspaces that intersect with the specified range are fetched from LSH buckets when RSATree is used to execute an aggregate query. After validation, involved IHs are merged and used to estimate the actual distribution of the requested values.

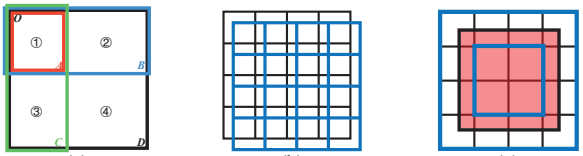
the index of the histogram bin, is defined as

$$H(x_1, \cdots, x_D, b) = \sum_{x'_1=1}^{x_1} \cdots \sum_{x'_D=1}^{x_D} \sum_{b'=1}^{b} h(x'_1, \cdots, x'_D, b') \quad (1)$$

where $h(x_1, \cdots, x_D, \cdot)$ calculates the values of all histogram bins b and represents the histogram of values in each binned grid. The IHs of any rectangular area in the data space can be calculated as

$$\sum_{p \in \{0,1\}^d} (-1)^{d-\|p\|_1} H(x^p, \cdot) \quad (2)$$

where x^p are the corners of the rectangular area with $p \in \{0, 1\}^d$.



(a) (b) (c)

Figure 4. (a) By using an SAT, the sum of values inside area can be calculated as $D+A-B-C = (①+②+③+④)+①-(①+②)-(①+③) = ④$. (b) Misaligned computational grids and SAT cells. (c) Errors occured due to the mismatching of computational grids (blue rectangles) and data cells (the red region).

Moreover, the feature descriptor in each cell in RSATree is not necessarily a histogram. The feature can also be one or more aggregated values (such as sum or $count$) or their combination (e.g., store sum and $count$ to calculate the average). However, the feature descriptor can be regarded as an array of length B regardless of its type and be treated similar to a histogram when calculating .

Local histogram binning. Binning IHs on the basis of the global range of all data points results in skewed distributions of histograms because each histogram is only built upon a small portion of data points in the same subspace with similar values on all dimensions. Thus, we calculate the local range of data points in each subspace to enable dynamic binning, thereby providing high approximation accuracy.

4.2.3 Generating LSH Buckets

We adopt the LSH strategy to enable efficient random access of IHs that possibly have intersections with query ranges. LSH can be used as an approximate nearest neighbor search, which is a point-to-point search, for multidimensional points. To support point-in-range searches and range overlapping tests, we extend LSH by applying a uniform sampling on the edges of the range (a hyper-rectangle). Thus, the extended LSH can be used to accelerate the RSATree construction and range query processes by hashing generated IHs into locality-preserving buckets (Figure 3 (c)). More details can be found in Section 1 in the supplementary material. The LSH functions we use satisfy the p-stable distribution of points in the Euclidean space [21].

4.2.4 Progressive Construction

The three aforementioned steps denote a standard process of constructing an RSATree. Its performance is inversely proportional to the data size because a standard R-tree keeps all data points in memory. Our solution for that is a progressive construction scheme.

We initially uniformly sample the input data and construct an R-tree based on the sampled data points. We preserve the nodes of the R-tree and obtain a staged partition. When inserting the remaining points, the expansion and splitting of the nodes are no longer processed, and only the node containing the point is selected. If the node containing the point does not exist, then the nearest node is searched and its rectangle is expanded to include the new point. IHs are computed for each node by re-inserting all points inside it. Feature descriptors are stored and updated when inserting points rather than keeping the original points. Progressive construction not only reduces the storage cost by loading parts of data into the memory but also improves the construction efficiency by reducing the calculation of branch selection and splitting, thereby making it relatively suitable for large-scale datasets. However, new coming points that cannot fit in any existing subspace of the staged partition may exist during the formulation of a complete partition. The resulting expansion of the existing subspaces may cause the original partition to be distorted to some extent. These IHs cover a large space and are slightly inaccurate due to the decreasing granularity.

4.3 Usage

We denote a high dimensional dataset as $\mathcal{V}$ with n dimensions $\mathcal{D} = \{D_1, \cdots, D_n\}$. Assume that the domain of dimensions is $\{[a_1, b_1], \cdots, [a_n, b_n]\}$.

Aggregate query. We can define an aggregate query with range $R = \{[x_1^1, x_1^2], \cdots, [x_n^1, x_n^2]\}$ and an aggregation function A, denoted as $Q(R, A)$, which applies aggregation to the set of all data points located within range R ($\{\mathbf{v}|\mathbf{v} \in \mathcal{V}$ and $\forall d, \mathbf{v}_d \in [x_d^1, x_d^2]\}$).

Computational grids. We consider a subset $\mathcal{S} = \{D_{i_1}, \cdots, D_{i_k}\}$ of $\mathcal{D}$ and an aggregation function A of interests when drawing a k-dimensional statistical chart (e.g., a histogram when $k = 1$, or a binned scatterplot when $k = 2$). With given aggregate dimensions $D_m \in \mathcal{D}$ and a range filter $R_f = \{[y_1^1, y_1^2], \cdots, [y_n^1, y_n^2]\}$, the result is denoted as a binned k-dimensional cube $\mathcal{G}$ on $\mathcal{S}$. Each dimension $D_d \in \mathcal{S}$ is splitted into N_d bins $[h_d^0, h_d^1), \cdots, [h_d^{N_d-1}, h_d^{N_d}]$, where $h_d^0 = y_d^1$ and $h_d^{N_d} = y_d^2$. Thus, $\mathcal{G}$ contains a total of $N_{i_1} \times \cdots \times N_{i_k}$ computational *grids*. Each grid in $\mathcal{G}$ can be identified by a k-tuple $T = \langle t_{i_1}, \cdots, t_{i_k} \rangle$ and contains an aggregated value. Each value can be fetched by performing an aggregate query $Q(R_T, A)$, where R_T is the intersection of the range defined by filter R_f and the bin, which is denoted as

$$R_T = \left\{ r_j \middle| r_j = \begin{cases} [h_j^{t_j-1}, h_j^{t_j}], & j \in \{i_1, \cdots, i_k\} \\ [y_j^1, y_j^2], & \text{otherwise} \end{cases} \right\}$$

In addition, the computational grids do not have to be equidistant.

Querying using IH. The result of an aggregate query can be calculated from the preprocessed IH by using Equation 2. However, making an aggregate query of an arbitrary query range on the constructed IH can result in an imperfect fit between the corners of the queried rectangular area and the boundary of binned grids (Figure 4(b)). In practice, the coordinates of 2^d corners of the rectangular area are rounded before taking into Equation 2. We also implement an algorithm that can reduce the number of required additions or subtractions when batching the aggregated results that correspond to the computational grids $\mathcal{G}$ [30]. See more details in the supplementary material (Section 1.2).

Querying using RSATree. Figure 3(d) illustrates the operation of an aggregate query in an RSATree. When an aggregate query $Q(R, A)$ comes with a dimension subset $\mathcal{S}$ and an aggregate dimension $D_m \in \mathcal{D}$, an RSATree initially leverages LSH functions to narrow down the searching space $\mathcal{G}$ into a candidate range, in which IHs that are possibly overlapping with R are stored. The RSATree collects a set of histogram tables overlapping with R by comparing the bounding boxes of all candidate IHs on $\mathcal{S}$ to R. This set is then used to calculate the histogram from which the approximation of aggregation result $A(D_m)$ can be estimated.

5 Visual Query with RSATree

In this section, we explain the manner in which an RSATree can be used in visual analytics. We initially introduce the general flowchart of an RSATree-based visual query. Then, we present the visual interface and related optimization. Moreover, the influence of approximate query is discussed.

5.1 Flowchart

Visual query with RSATree forms a seamless composition of multiple offline and online steps. First, an RSATree representation is precomputed offline on the basis of the selected dimensions in a dataset. RSATree representation adaptively partitions the entire data space on the user-defined binning dimensions and uses IHs to depict the distribution of each data subspace. Each histogram or other descriptors are computed to summarize the data distribution of the user-defined aggregate dimension. Second, the binning strategy is determined on the basis of the predefined schema or user-defined parameters to generate visual charts that display the data attributes. On the basis of the binning strategy, computational grids are generated before performing aggregate queries. These aggregate queries are then answered by searching the corresponding IH in the RSATree to estimate the distribution of the binned area. Subsequently, the requested visualization can be created. The construction of the visualization and aggregate queries are performed online because the RSATree is stored in memory. Moreover, analysts are allowed to adjust interactively the visualization parameters, such as filters, bin widths, and dimensions to query. These interactions update the computational grids and trigger a new series of aggregate queries, which are efficiently answered by the RSATree. Therefore, instant visual feedback is provided in the visualization.

By supporting arbitrary range queries and flexible binning strategies, RSATree enables serveral useful operations for visual query systems. Table 1 compares RSATree to existing approaches for visual query.

Table 1. Comparison of different data cube approaches for visual query. Hashedcubes is not shown as it share similar properties with Nanocubes.

	Square Crossfilter	imMens	Nanocubes	RSATree
Architecture	Client	Client (Server)	Client-Server	Client (Server)
Demonstrated data size	10^5	10^{12}	10^{12}	10^{12}
2D binning	No	Yes	Yes	Yes
Binning strategies	Predefined grouping	Equi-width	Equi-width	Flexible strategies
Multiple brushes	Yes	No	Yes	Yes
Zooming	No	Predefine levels	Quad-tree levels	Continuous zooming
Supported measures	Algebraic	Count only	Algebraic	Not limited[1]
Approximation	No	No	No	Yes

[1] Non-algebraic measures are estimated from data distributions.

5.2 Interface

To show how an RSATree is used to support visual query, we implement a visual interface that can be accessed through a web browser. The visual interface has two forms, corresponding to the two scenarios discussed in Section 3.1: brushing and linking for spatial-temporal data (Figure 1(a)), as well as customized specification of charts (Figure 1(b)(c)).

The brushing and linking operations are achieved by regenerating the computational grids and querying during brushing. When creating custom charts, users can modify parameters, such as query ranges and bin widths, by dragging the sliders. Although a slider is designed to allow the selection of any value within a certain range of the real number field, only a certain number of discrete values can be actually taken due to the limitation of screen pixels. This conclusion also applies to the generation of computational grids. On this basis, we can further optimize the usage efficiency of RSATree without affecting user interactions.

Scale alignment can reduce the error of the results through better integration of the interface and precomputation process. We can generate queries that are suitable for our preprocessed data structure without affecting the user's exploration by reasonably defining and limiting the design space for user interaction.

To support a flexible binning strategy, RSATree applies selection of adaptive granularities, which makes a fine overall granularity (with storage space remaining the same). However, such an approach causes some problems. The computational grids cannot be perfectly matched to the cube cells similar to using traditional data cubes due to the different sizes of cube cells. This process requires interpolation to derive the answer, which is the main source of the error. In principle, the error should be limited in a reasonable range as long as the granularity is fine. However, in practice, a slight shift will cause the fine-grained precomputation results to completely fail to provide accurate answers, thereby resulting in an actual error that is higher than expected (Figure 4(b)). To solve such issue, we consider scale alignment, which is a good integration of the interface and precomputation.

Scale alignment is based on an observation. Although the user specifies the query range and parameters in floating point numbers (by box selection or sliders), the actual execution still occurs in discrete spaces due to the limitation of screen pixels. As such, we can align the design space of user interaction, such as the slider scale and the "scales" of cube cells. This condition makes the computational grids and the cube cells to have a large probability of full coincidence when performing range queries, particularly when the granularity is fine (Figure 8(e-f)). In practice, the domains of each dimension are divided into scales, and the size of IH cells and computational grids are represented by the number of scales. These numbers are calculated on the basis of their original size and automatically adopt a close number containing only 2, 3, and 5 as prime factors (e.g., 3600). This condition makes the size of computational grids likely to be a multiple of related IH cells, thereby resulting in accurate queries. The size of the scales can be automatically determined on the basis of the screen resolution (e.g., a pixel on the slider) or manually specified for a dimension with special meaning (e.g., seconds for time dimension).

5.3 Performance-Accuracy Tradeoff

Errors are produced because the answer returned by RSATree is an approximate. We provide an option that allows users to toggle between displaying aggregated values and errors.

5.3.1 Estimating Error

When using a distributive measure, the margin of error due to the mismatching of computational grids and data cells can be quickly determined. As shown in Figure 4(c), the aggregated value of the red region lies in the range of two values of the two blue rectangles. Therefore, we can determine the error of the result by two additional

queries. Calculating the error in this manner does not require access to raw data, although it increases the time complexity to 3 times of the original query. We define the error to be $(V_{max} - V_{min})/V_{returned}$. A switch is provided in the interface to open the error display rather than the original results, which are encoded by color (heatmap and binned scatterplot), error bars (bar chart and histogram), or y-axis (line chart).

However, estimating errors for non-distributive measures is non-trivial. Algebraic measures can be calculated based on its definition (e.g. $mean_{max} = sum_{max}/count_{min}$). Other measures can only be estimated using the distributions recorded by IHs, which may produce huge uncertainties.

5.3.2 LSH

Generally, LSH improves the performance in exchange for reduced accuracy. In many situations, analysts may have different requirements on the accuracy of the answer; hence, analysts may be allowed to modulate the performance–accuracy tradeoff when making visual queries. Therefore, LSH can be reasonably applied to accelerate the query and guarantee a reduced but still acceptable accuracy. When visualization is used as a data preview (e.g., analysts navigate a map to identify interesting regions), RSATree with the LSH scheme is preferred to accelerate online data exploration.

5.3.3 R-Tree Hierarchy

In addition to the resolution of IHs, the fineness of R-tree partition and the tree height directly affect the response time and accuracy of the queries. Figure 8 shows the query performance with different R-tree heights. Moreover, we can construct IHs at different levels of the R-tree to obtain a multiresolution data representation. This condition provides many options to the scenario-based performance–accuracy tradeoff.

We can design a progress exploration scheme based on such hierarchy. The progressive exploration scheme allows analysts to view the result returned from a coarse level and apply filters to focus on a small region of the input data based on observations received from the approximate preview. Therefore, a small number of IH tables are involved with such coarse grain. On the basis of this preliminary option, the R-tree can be pruned to eliminate irrelevant subtrees. Analysts then receive an accurate observation for the small region. This process continuously iterates during progressive exploration.

5.4 Implementation

We implement a prototype system to demonstrate the visual query with RSATree using a client–server architecture. The server runs Node.js with JavaScript code that can flexibly handle various types of data. The client requests data from the server through a defined API. The interface is written by means of HTML5, JavaScript, SVG, Canvas, and D3.js [11]. WebGL is not used.

6 Experiments

In this section, we present the experiments to evaluate the capabilities of RSATree. The evaluation was performed on a synthetic dataset and seven real-world datasets to test the construction of RSATree and its performance for three interactive tasks. All experiments are performed on a 3.40 GHz Intel(R) Xeon(TM) E3-1245 CPU with 32 GB main memory. The web-based interface is viewed in Chrome 71.0.3578.98.

In the experiments, we use an average relative error (ARE) metric to evaluate an approximate query. Suppose that a query computes an aggregation A over n bins of data $X_1, \cdots, X_n$ and returns n representative values $v_1, \cdots, v_n$. The ARE of the query is defined as

$$ARE = \frac{1}{n}\sum_{i=1}^{n}\frac{|v_i - A(X_i)|}{max(v_i, A(X_i))}$$

where $A(X_i)$ is the exact aggregate over the ith bin.

6.1 Datasets

The datasets we collected range up to over 1 billion elements. In addition, a synthetic dataset is included, which is widely used for evaluation in previous studies. Table 2 summarizes the relevant information of the experimental datasets, including the ScatterPlot Matrix (SPLOM) [35] and six real-world datasets. The number of data records in the dataset, storage consumption, and construction time of RSATree are reported in the first three columns. Column "Schema" indicates the dimensions of each dataset that are used to build RSATree. The height of constructed R-tree, number of partitioned subspaces, and bin number of each created SAT are shown in the last three columns. Deatails of the datasets can be found in the supplementary material (Section 2).

6.2 RSATree Construction

We show the effect on construction time, storage consumption, and response time through an example by using the SPLOM dataset [44]. Figure 5 shows the results. We evaluate 10 different datasets. Figure 5(a)–(c) use datasets with five dimensions, and the number of bins of each dimension set from 10 to 50. The number of bins of datasets used in Figure 5(d)–(f) is 30, and the number of dimensions changes from 1 to 5. We adjust the parameters to achieve accurate answers to queries, excluding the influence of accuracy. The response time is the average time to generate a heatmap that consists of two dimensions (the two datasets with only one dimension use a one-dimensional heatmap). Table 2 summarizes the relevant information in constructing an RSATree for the datasets.

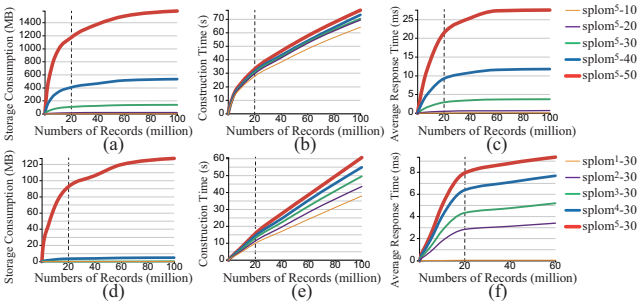


Figure 5. Number of records and (a and d) storage consumption, (b and e) construction time, and (c and f) response time when creating an RSATree using different SPLOM datasets. The dashed lines show the number of points sampled for the progressive construction.

The growth of storage consumption and response time gradually decrease after the progressive construction starts, and remain unchanged when the number is larger than a certain number. This condition remarkably improves the efficiency of RSATree built on large-scale datasets. The construction time nearly linearly increases with the number of records. However, a slight slow down can be observed after entering the progressive construction. The number of dimensions and bins considerably affect the storage consumption but not the construction time. The response time is dependent on the number of bins because it directly affects the number of aggregated results. By contrast, the number of dimensions has little influence on response time.

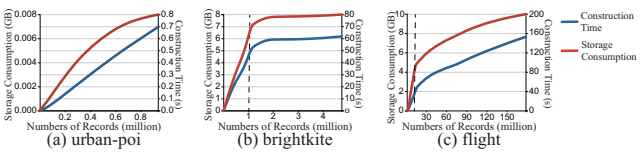


Figure 6. The growth of storage consumption and construction time when inserting records into RSATree for (a) Urban-POI, (b) Brightkite, and (c) Flight datasets.

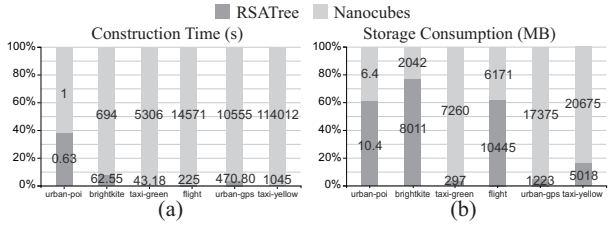


Figure 7. Comparisons of (a) construction time and (b) storage consumption to Nanocubes. The x-axis is ordered by the record number of each dataset.

Table 2. Information of experimental datasets and their associated RSATree representations. The numbers in parentheses on the "#Records" column denote the sampling rates when using the progressive construction.

Name	#Records	Storage	Time	Schema	Tree height	#Sub-spaces	#Bins
splom-10	1.0B (0.02)	1.3M	9:11m	d1, d2, d3, d4, d5	2	100	10
splom-50	1.0B (0.02)	1.1G	10:55m	d1, d2, d3, d4, d5	2	90	50
brightkite	4.5M (0.2)	7.8GB	63s	lon, lat, weekday, hour, time	3	854	60
flight	180 (0.05)	10.2G	3:48m	Distance, LateAircraftDelay, CarrierDelay, year, month, weekday	5	934	60
taxi-yellow	1.5B (0.01)	4.9G	17:24m	lon, lat, weekday, hour, time	6	1974	60
taxi-green	69M (0.02)	296.5M	54s	lon, lat, weekday, hour, time	6	1945	60
urban-poi	0.9M	10.4M	<1s	lon, lat	5	1346	60
urban-gps	375M (0.01)	2.5G	5:10m	lon, lat, time, speed, direction	6	1992	60
didi	1.5B (0.01)	1.4G	12:18m	lon, lat, month, day, time	4	171	60

Figure 6 presents the curves for construction time and storage consumption of three real-world datasets (urban-POI, brightkite, and flight). Urban-POI is a small dataset that the progressive construction is not involved, while the curves for the other two datasets are similar to those of SPLOM datasets. Figure 7(a) shows that the construction times of RSATree are shorter than Nanocubes, especially on big datasets. Storage consumptions of RSATree, as shown in Figure 7(b), are larger than Nanocubes on small datasets, but take up relatively less space on big datasets. The exception is the Flight dataset, which may because of a specific distribution [42]. We discuss this issue in Section 7.1.

6.3 The Response Time

In summary, RSATree fulfilled the three design requirements in the user study. The underlying data structure of RSATree can answer arbitrary range queries. RSATree can achieve fast response speed and low storage consumption during the query by using approximate queries and control the loss of accuracy to a reasonable range. The supported flexible binning strategies can provide a more efficient exploration than only using equi-width binning strategy.

We have asked 8 participants (all computer science students with knowledge of visual analysis) to explore three datasets using our prototype system and recorded the response time of queries. Three datasets represent three typical scenarios, namely, an individual heatmap, spatiotemporal analysis, and exploring by specifying custom charts. The participants are assigned specific tasks. The results of user study in supplementary materials show that the response time can meet the requirements for exploratory analysis, as shown in Table 3. Moreover, the result of an objective questionnaire shows that supported flexible binning strategies can provide a more efficient exploration than only using equi-width binning strategy.

We also compare the performances with Nanocubes [42]. Because Nanocubes is limited to spatial-temporal datasets, only the Brightkite dataset is tested in Nanocubes. Results show that the response time using RSATree is weaker than Nanocubes, as a trade-off for flexible binning. Another reason might be that our backend algorithms are run on a NodeJS server, which leads to lower performance than C++. Moreover, there are opportunities for both inter- and intra-IH parallelization [30].

Table 3. Statistics of response times over three datasets in microseconds

statistic/dataset	urban-poi	brightkite	brightkite_nanocubes	flight
count	426	1864	2024	6383
median	266.86	222.39	31.27	28.21
mean	267.14	223.25	38.28	28.75
stdev	8.72	5.94	27.58	1.53
max	281.66	234.72	173.34	31.79
90-percentile	273.21	230.37	69.88	30.82
mean ARE	1.08%	8.05%	–	9.01%
mean JSON size	882.0KB	732.3KB	506.8KB	68.2KB

6.4 Performance-Accuracy Tradeoff

As discussed in Section 5.3, several factors, such as response time and accuracy, may influence the query performance of RSATree. We analyze the influences of R-tree height, LSH, and scale alignment with RSATree built on the Urban-POI dataset. The Urban-POI dataset is part of the urban data collected from January $10-31$, 2014 in a city [61,62,71]. It contains information on approximately 1 million POI locations. We use their longitudes and latitudes to build the RSATree. To show clearly how data distribution can be captured and affect the performance of RSATree, we evaluate the response time and ARE rate of three regions with different densities, that is, a global region with 933230 data points, Region 1 with 133118 data points, and Region 2 with 8333 data points (Figure 8(a)). For each region, experiments are performed by using LSH or R-tree search, which is returned at different heights, as well as the baselines. We choose two different raw SAT structures as baselines, including one with very fine granularity (S1) and the other with a similar size to RSATree whose tree height is 5 (S2). The scale alignment scheme is used in these experiments. The accuracy improvement caused by the scale alignment scheme is then evaluated (Figure 8(e-f)).

Storage Consumption. Figure 8(b) shows the relations between storage consumption and tree height. Moreover, it can be seen that the storage consumption of S1 is much larger than RSATree and S2 has a similar storage consumption to RSATree.

Response Time. As shown in Figure 8(c), querying on Region 1 requires more time than querying on the global region and Region 2. This condition is probably because Region 1 has a higher average density and is partitioned into more subspaces to reach a higher query accuracy in comparison with the other two regions. Interestingly, raw SAT does not have the fastest response time for all the cases. The reason may be that the excessive storage space makes the system cache less efficient.

Error Rate. Figure 8(d) shows that the error rates of different regions are maintained at the same level. Thus, distribution-aware adaptive granularity can balance the overall accuracy well. Queries using LSH have the lowest response times but highest ARE rates. The response time increases and ARE changes in the opposite direction with the increase of the R-tree height. This condition satisfies expectations, in which the response time and accuracy can be balanced by selecting the appropriate query parameters based on usage scenarios. S1 has the lowest ARE (less than 0.1%) at the expense of storage consumption. On the other hand, S2 has relative high AREs, especially with the highest ARE in Region 1. In contrast, RSATree can better balance the granularities of different regions.

Scale Alignment. As shown in Figure 8(e), the scale alignment scheme reduces more than 50% of ARE rates in all regions. Particularly, the ARE rate of Region 1 is mostly reduced in comparison with the other regions, thereby verifying the effectiveness of the scale alignment scheme on querying in dense regions. Scale alignment is effective because it can make many computational grids that completely coincide with IH cells, thereby reducing the rounding operation during calculation (described in Section 4.3). Figure 8(f) shows the rates of computational grids that coincide with IH cells and does not require to be rounded. Scale alignment can eliminate many of the misalignments, especially in the region with high density.

7 Discussions

7.1 Technological Choices

Space Partitioning Algorithms. In addition to R-tree, many space partitioning algorithms, such as quadtree, k-d tree, and their variants (e.g., k-d-b tree) are used. R-tree fits our requirements because it can compactly partition the space into subspaces and discard empty subspaces. We test k-d tree using the "median of the most spread

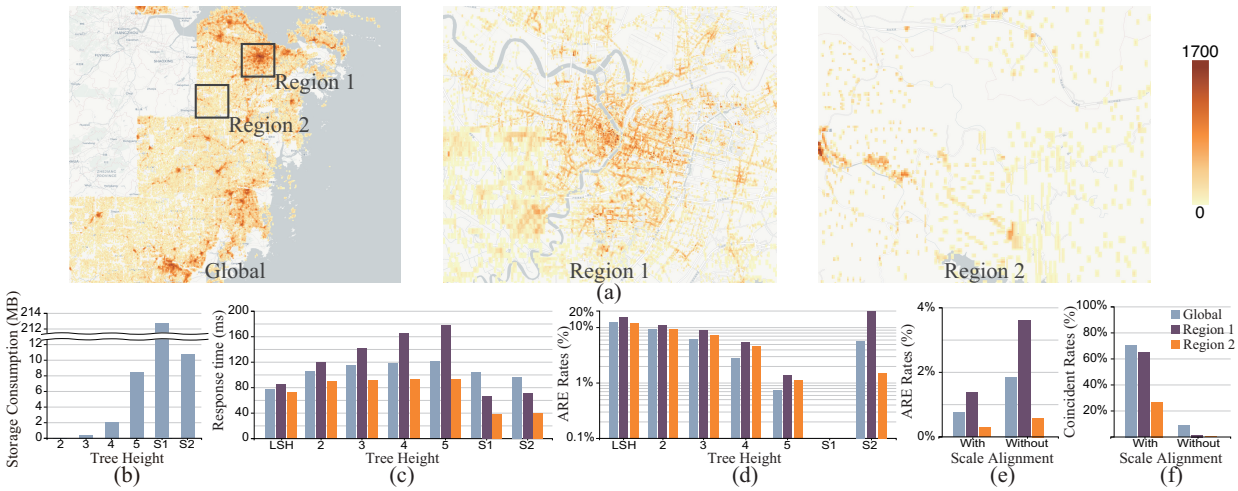


Figure 8. Applying RSATree in querying a heatmap of Urban-POI distribution in three regions. From left to right: Global region, Region 1 with high density, and Region 2 with low density. (a) Map view, (b) storage consumption, (c) Response time, and (d) ARE rates when using R-tree with different heights and LSH, as well as two raw SATs. The scale alignment scheme is used in (a-d). (e) ARE and (f) coincident rates of querying in an RSATree with and without the scale alignment scheme.

dimension" splitting strategy. The cores of high-density regions are constantly split from the middle when using k-d tree to partition the data space. This process makes the generated subspaces to be irregularly distributed, and a large number of data points gather in the corners of the rectangular area. Meanwhile, k-d tree and quadtree are designed to contain the entire data space by leaving large subspaces partitioned in the marginal area of space. These problems make the use of His inefficiently because most data are gathered in a few cells. These problems also tremendously increase the errors because the values of high-density regions are shared by empty spaces. By contrast, R-tree is flexible in selecting leaf nodes.

Supporting Categorical Dimensions. Categorical dimensions can be regarded as the numerical dimension, in which the only difference is that their values are discretely distributed across the entire domain. However, this dimension does not produce the desired effect. With the scale alignment scheme, we can take each category as a "scale", which yields fair results when the number of categories is large (> 100). We process the categories as the dimensions of the IH when the number of categories is small. Visual query requires to accumulate the results of all relevant categories after normal calculation. This process results in low accuracy loss at a cost of an acceptable computational overhead.

Sampling for Progressive Construction. As described in Section 4.2.4, RSATree use uniform sampling to select data points for constructing the "skeleton" of the R-tree in the progressive construction process. We choose uniform sampling because it has the best adaptability. Analysts do not need to have prior knowledge of the data before applying uniform sampling. Thus the whole preprocessing can be completed automatically (only a sample rate is required). However, in some cases, uniform sampling shows its limitations. As shown in Figure 6 (c), the curve of inserted record number and storage consumption for the Flight dataset does not perfrom like the SPLOM dataset (Figure 5) and other real datasets (Figure 6 (a) (b)). The storage consumption keeps growing fast after entering the progressive construction. The reason may be that the uniform sampling fails to capture the data distribution well. The same problem may occur when applying filtering and grouping across multiple attributes. These operations may fundamentally change the underlying data distributions observed, rendering the original approximation irrelevant. In these cases, a stratified sampling [2] or machine learning [41] approach might be better.

7.2 Limitations

Similar to regular data cubes, the memory consumption of RSATree increases with the dimension. Although the distribution-aware partitioning of subspaces can alleviate the considerable increase to some extent, the effect in the case of high dimensions is not ideal due to the nature of R-tree. The effectiveness of R-tree will be affected by the sparsity of the data and the complexity of data distribution when it is used on high-dimensional data. Therefore, RSATree does not support data with more than five dimensions well.

R-tree is not perfect for all data and situations. Several complex data distributions cannot be captured because R-tree divides the space into orthogonal subspaces. This issue can be avoided to a certain extent by fine binning of SAT. However, this process still results in storage overhead, especially when the dimension is high. This may be addressed by a tighter integration with the interaction design. Falcon [48] shows that the dimensions of the required data cubes can be reduced to less than 3 by initializing only the data cubes slices associated with the active view. This can also enable a cold-start of exploration. Combining with such an interaction design may be a good solution, because R-tree performs better when the number of dimensions is lower.

To illustrate the capability of RSATree, we implement visual query interfaces for spatial-temporal data (Figure 1(a)) and visual specification (Figure 1(b)(c)), which are widely used [42, 44, 49]. However, our current implementation does not fully utilize the benefits of RSATree. We suppose that RSATree has the potential to combine with more novel visualizations and interactions for providing flexible exploratory analysis. For example, Sarvghad et al. [56] proposed an embedded interaction technique for flexible adjusting of data bins, whose scalability can be powered by RSATree.

8 Conclusions and Future Works

In this study, we propose RSATree, which is a novel data representation that supports efficient web-based aggregate query for large-scale tabular datasets. An RSATree returns approximate answers to generate instant visual feedback in interactive visualizations by reformulating, abstracting, and simplifying the input data into a nested three-level representation. The advantages of an RSATree include: 1) answering aggregate query of arbitrary ranges and 2) supporting flexible binning strategies; 3) moreover, its response time is low, and its storage cost is acceptable.

Several directions can be investigated in future work. First, a better partition algorithm, such as deep learning models that can adapt to the data distributions [37] may produce a better accuracy. Second, a mixed storage mode that can separately handle outliers may be required. Third, we plan to implement RSATree on GPU to improve its parallelism.

Acknowledgments

We wish to thank all the anonymous reviewers for their valuable comments, and all the participants for their active participation. The work is supported by the National Science & Technology Fundamental Resources Investigation Program of China (2018FY10090002), the National Natural Science Foundation of China (61772456, 61761136020, U1609217,61672538, 61872388, 61872389).

REFERENCES

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join Synopses for Approximate Query Answering. In *ACM SIGMOD Record*, vol. 28, pp. 275–286. ACM, 1999.

[2] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 29–42. ACM, 2013.

[3] C. Ahlberg. Spotfire: an Information Exploration Environment. *ACM SIGMOD Record*, 25(4):25–29, 1996.

[4] H. Ahn, N. Mamoulis, and H. M. Wong. A Survey on Multidimensional Access Methods. Technical report, Hong Kong University of Science and Technology, 1997.

[5] D. Barbará and M. Sullivan. Quasi-Cubes: Exploiting approximations in multidimensional databases. *ACM SIGMOD Record*, 26(3):12–17, 1997.

[6] L. Battle, R. Chang, and M. Stonebraker. Dynamic Prefetching of Data Tiles for Interactive Visualization. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 1363–1375. ACM, 2016.

[7] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Acm Sigmod Record*, vol. 19, pp. 322–331. Acm, 1990.

[8] M. Behrisch, F. Korkmaz, L. Shao, and T. Schreck. Feedback-Driven Interactive Exploration of Large Multidimensional Data Supported by Visual Classifier. In *Visual Analytics Science and Technology (VAST), 2014 IEEE Conference on*, pp. 43–52. IEEE, 2014.

[9] J. L. Bentley. Multidimensional Binary Search Trees in Database Applications. *IEEE Transactions on Software Engineering*, (4):333–340, 1979.

[10] E. Bertini and G. Santucci. Give chance a chance: modeling density to enhance scatter plot quality through random data sampling. *Information Visualization*, 5(2):95–110, 2006.

[11] M. Bostock, V. Ogievetsky, and J. Heer. D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[12] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages & Computing*, 8(2):215–260, 1997.

[13] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The International Journal on Very Large Data Bases*, 10(2-3):199–223, 2001.

[14] A. Chaudhuri, T. H. Wei, T. Y. Lee, H. W. Shen, and T. Peterka. Efficient Range Distribution Query for Visualizing Scientific Data. In *Visualization Symposium (PacificVis), 2014 IEEE Pacific*, pp. 201–208. IEEE, 2014.

[15] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. Narasayya. Overcoming limitations of sampling for aggregation queries. In *International Conference on Data Engineering*, pp. 534–542. IEEE, 2001.

[16] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[17] H. Chen, W. Chen, H. Mei, Z. Liu, K. Zhou, W. Chen, W. Gu, and K.-L. Ma. Visual abstraction and exploration of multi-class scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1683–1692, 2014.

[18] W. Chen, F. Guo, and F.-Y. Wang. A survey of traffic data visualization. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):2970–2984, 2015.

[19] W. Chen, Z. Huang, F. Wu, M. Zhu, H. Guan, and R. Maciejewski. Vaud: A visual analysis approach for exploring spatio-temporal urban data. *IEEE Transactions on Visualization and Computer Graphics*, 24(9):2636–2648, 2017.

[20] F. C. Crow. Summed-Area Tables for Texture Mapping. In *ACM SIGGRAPH computer graphics*, vol. 18, pp. 207–212. ACM, 1984.

[21] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262. ACM, 2004.

[22] M. Derthick, J. Kolojejchick, and S. F. Roth. An Interactive Visualization Environment for Data Exploration. In *KDD*, pp. 2–9, 1997.

[23] J.-D. Fekete and R. Primet. Progressive analytics: A computation paradigm for exploratory data analysis. *arXiv preprint arXiv:1607.05162*, 2016.

[24] D. Fisher, I. Popov, S. Drucker, et al. Trust Me, I'm Partially Right: Incremental Visualization Lets Analysts Explore Large Datasets Faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1673–1682. ACM, 2012.

[25] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data mining and knowledge discovery*, 1(1):29–53, 1997.

[26] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pp. 47–57. ACM, 1984.

[27] C. G. Healey and B. M. Dennis. Interest Driven Navigation in Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1744–1756, 2012.

[28] J. Heer and B. Shneiderman. Interactive Dynamics for Visual Analysis. *Queue*, 10(2):30, 2012.

[29] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online Aggregation. In *Acm Sigmod Record*, vol. 26, pp. 171–182. ACM, 1997.

[30] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra. Fast summed-area table generation and its applications. In *Computer Graphics Forum*, vol. 24, pp. 547–555. Wiley Online Library, 2005.

[31] Z. Huang, Y. Lu, E. Mack, W. Chen, and R. Maciejewski. Exploring the sensitivity of choropleths under attribute uncertainty. *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[32] C. Jermaine, A. Dobra, S. Arumugam, S. Joshi, and A. Pol. The Sort-Merge-Shrink Join. *ACM Transactions on Database Systems (TODS)*, 31(4):1382–1416, 2006.

[33] S. Joshi and C. Jermaine. Materialized Sample Views for Database Approximation. *IEEE Transactions on Knowledge and Data Engineering*, 20(3):337–351, 2008.

[34] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and interactive cube exploration. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pp. 472–483. IEEE, 2014.

[35] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer. Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pp. 547–554. ACM, 2012.

[36] A. Kim, E. Blais, A. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid Sampling for Visualizations with Ordering Guarantees. *Proceedings of the VLDB Endowment*, 8(5):521–532, 2015.

[37] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pp. 489–504. ACM, New York, NY, USA, 2018. doi: 10.1145/3183713.3196909

[38] I. Lazaridis and S. Mehrotra. Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. In *ACM SIGMOD Record*, vol. 30, pp. 401–412. ACM, 2001.

[39] T.-Y. Lee and H.-W. Shen. Efficient Local Statistical Analysis via Integral Histograms with Discrete Wavelet Transform. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2693–2702, 2013.

[40] D. Li, H. Mei, Y. Shen, S. Su, W. Zhang, J. Wang, M. Zu, and W. Chen. Echarts: A declarative framework for rapid construction of web-based visualization. *Visual Informatics*, 2(2):136–146, 2018.

[41] Q. Lin, W. Ke, J.-G. Lou, H. Zhang, K. Sui, Y. Xu, Z. Zhou, B. Qiao, and D. Zhang. BigIN4: Instant, Interactive Insight Identification for Multi-Dimensional Big Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 547–555. ACM, 2018.

[42] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465, 2013.

[43] Z. Liu and J. Heer. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2122–2131, 2014.

[44] Z. Liu, B. Jiang, and J. Heer. imMens: Real-time Visual Querying of Big Data. In *Computer Graphics Forum*, vol. 32, pp. 421–430. Wiley Online Library, 2013.

[45] S. Martin and H.-W. Shen. Transformations for Volumetric Range Distribution Queries. In *Visualization Symposium (PacificVis), 2013 IEEE Pacific*, pp. 89–96. IEEE, 2013.

[46] H. Mei, Y. Ma, Y. Wei, and W. Chen. The design space of construction tools for information visualization: A survey. *Journal of Visual Languages & Computing*, 44:120–132, 2018.

[47] X. Mingliang, L. Pei, L. Mingyuan, F. Hao, Z. Hongling, Z. Bing, L. Yusong, and Z. Liwei. Medical image denoising by parallel non-local means. *Neurocomputing*, 195:117–122, 2016.

[48] D. Moritz, B. Howe, and J. Heer. Falcon: Balancing interactive latency and resolution sensitivity for scalable linked visualizations. 2019.

[49] C. A. Pahins, S. A. Stephens, C. Scheidegger, and J. L. Comba. Hashedcubes: Simple, Low memory, Real-Time Visual Exploration of Big Data. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):671–680, 2017.

[50] Y. Park, M. Cafarella, and B. Mozafari. Visualization-Aware Sampling for Very Large Databases. In *IEEE 32nd International Conference on Data Engineering (ICDE)*, pp. 755–766. IEEE, 2016.

[51] V. Poosala and V. Ganti. Fast Approximate Query Answering Using Precomputed Statistics. In *International Conference on Data Engineering*, p. 252. IEEE, 1999.

[52] F. Porikli. Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 829–836. IEEE, 2005.

[53] S. Rahman, M. Aliakbarpour, H. K. Kong, E. Blais, K. Karahalios, A. Parameswaran, and R. Rubinfield. I've Seen Enough: Incrementally Improving Visualizations to Support Rapid Decision Making. *Proceedings of the VLDB Endowment*, 10(11):1262–1273, 2017.

[54] S. F. Roth, P. Lucas, J. A. Senn, C. C. Gomberg, M. B. Burks, P. J. Stroffolino, A. Kolojechick, and C. Dunmire. Visage: a User Interface Environment for Exploring Information. In *Information Visualization'96, Proceedings IEEE Symposium on*, pp. 3–12. IEEE, 1996.

[55] H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[56] A. Sarvghad, B. Saket, A. Endert, and N. Weibel. Embedded merge & split: Visual adjustment of data grouping. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):800–809, 2018.

[57] B. Shneiderman. Dynamic queries for visual information seeking. *IEEE software*, 11(6):70–77, 1994.

[58] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[59] J. W. Tukey. *Exploratory Data Analysis*. Reading, Mass., 1977.

[60] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. SEEDB: Efficient Data-Driven Visualization Recommendations to support Visual Analytics. *Proceedings of the VLDB Endowment*, 8(13):2182–2193, 2015.

[61] F. Wang, W. Chen, Y. Zhao, T. Gu, S. Gao, and H. Bao. Adaptively exploring population mobility patterns in flow visualization. *IEEE Transactions on Intelligent Transportation Systems*, 18(8):2250–2259, 2017.

[62] X. Wang, T. Gu, X. Luo, X. Cai, T. Lao, W. Chen, Y. Wu, J. Yu, and W. Chen. A user study on the capability of three geo-based features in analyzing and locating trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 2018.

[63] Z. Wang, N. Ferreira, Y. Wei, A. S. Bhaskar, and C. Scheidegger. Gaussian cubes: Real-time modeling for visual exploration of large multidimensional datasets. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):681–690, 2017.

[64] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, 2016.

[65] J. Xia, G. Jiang, Y. Zhang, R. Li, and W. Chen. Visual subspace clustering based on dimension relevance. *Journal of Visual Languages & Computing*, 41:79–88, 2017.

[66] J. Xia, F. Ye, W. Chen, Y. Wang, W. Chen, Y. Ma, and A. K. Tung. Ldsscanner: exploratory analysis of low-dimensional structures in high-dimensional datasets. *IEEE transactions on visualization and computer graphics*, 24(1):236–245, 2017.

[67] M. Xu, H. Wang, S. Chu, Y. Gan, X. Jiang, Y. Li, and B. Zhou. Traffic simulation and visual verification in smog. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(1):3, 2018.

[68] M. Xu, Y. Wu, Y. Ye, I. Farkas, H. Jiang, and Z. Deng. Collective crowd formation transform with mutual information–based runtime feedback. In *Computer Graphics Forum*, vol. 34, pp. 60–73. Wiley Online Library, 2015.

[69] E. Zgraggen, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska. How progressive visualizations affect exploratory analysis. *IEEE Transactions on Visualization & Computer Graphics*, (8):1977–1987, 2017.

[70] Z. Zhou, L. Meng, C. Tang, Y. Zhao, Z. Guo, M. Hu, and W. Chen. Visual abstraction of large scale geospatial origin-destination movement data. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):43–53, 2018.

[71] M. Zhu, W. Chen, J. Xia, Y. Ma, Y. Zhang, Y. Luo, Z. Huang, and L. Liu. Location2vec: a situation-aware representation for visual exploration of urban locations. *IEEE Transactions on Intelligent Transportation Systems*, 2019.