Eurographics Conference on Visualization (EuroVis) 2018
J. Heer, H. Leitte, and T. Ropinski
(Guest Editors)

Volume 37 (2018), Number 3

ConcaveCubes: Supporting Cluster-based Geographical Visualization in Large Data Scale

Mingzhao Li[1], Farhana Choudhury[1], Zhifeng Bao[1], Hanan Samet[2] and Timos Sellis[3]

[1]RMIT University, Melbourne, Victoria, Australia
[2]University of Maryland, College Park, Maryland, United States
[3]Swinburne University of Technology, Hawthorn, Victoria, Australia

Abstract

In this paper we study the problem of supporting effective and scalable visualization for the rapidly increasing volumes of urban data. From an extensive literature study, we find that the existing solutions suffer from at least one of the drawbacks below: (i) loss of interesting structures/outliers due to sampling; (ii) supporting heatmaps only, which provides limited information; and (iii) no notion of real-world geography semantics (e.g., country, state, city) is captured in the visualization result as well as the underlying index. Therefore, we propose ConcaveCubes, a cluster-based data cube to support interactive visualization of large-scale multidimensional urban data. Specifically, we devise an appropriate visualization abstraction and visualization design based on clusters. We propose a novel concave hull construction method to support boundary based cluster map visualization, where real-world geographical semantics are preserved without any information loss. Instead of calculating the clusters on demand, ConcaveCubes (re)utilizes existing calculation and visualization results to efficiently support different kinds of user interactions. We conduct extensive experiments using real-world datasets and show the efficiency and effectiveness of ConcaveCubes by comparing with the state-of-the-art cube-based solutions.

1. Introduction

In recent years, both the volume and the availability of urban data related to various social issues, such as real estate, crime, population, etc., are rapidly increasing. However, as the scale of data increases (e.g., over one million data points), existing information visualization methods, including directly visualizing individual data points on a map, suffer not only from large memory consumption, but also have perceptual and interactive scalability problems [LJH13]. In particular, users' perceptual and cognitive capacities are overwhelmed by data over-plotting, and users' interaction with large-scale datasets can easily lead to high latency.

A straightforward approach is to employ various sampling methods [DP12] to reduce the data to be displayed. However, they possibly elide interesting structures or outliers, thereby preventing users from querying the data to cater to their own preference in the data exploration stage. Therefore, one recent research trend is to design data cubes and pre-compute possible data aggregations to support efficient visualization, such as imMens [LJH13], Nanocubes [LKS13], and Hashedcubes [PSSC17]. As a result, they enable fast query processing for interactive visualizations of large and multidimensional data.

While these data cube structures are effective for solving the scalability problem, we observe that they only support heatmaps [Boj09] based on binned aggregation in terms of visualizing geographical features. Although Liu et al. [LJH13] have shown that heatmaps have advantages vis-à-vis sampled symbol maps (e.g., heatmaps can show the overall geographical distribution while sampled symbol maps cannot), the information that heatmaps can provide is often very limited (Example 1 in Section 3.4).

To deal with the limitation of heatmaps, we describe a design space for geographical visualization based on the result of data clustering (rather than relational aggregation). Following Tobler's First Law of Geography - *Everything is related to everything else, but near things are more related than distant things* [Tob70], we first cluster geographical data points that are close in distance (assuming Euclidean distance in contrast to other distances such as the Hausdorff distance [NJS11]) and share similar features, and then represent each cluster with a geographical bubble on top of the map. It is called bubble-based cluster maps as shown in Figure 2(a). Cluster maps, which is hard to support by existing data cubes (e.g., imMens [LJH13], Nanocubes [LKS13], Hashedcubes [PSSC17]), can provide additional information compared to heatmaps or binned plots. We compare existing map designs and discuss why we chose cluster maps, rather than heatmaps or binned plots, etc. in Section 3.4.

One problem of the existing cluster map design is that each cluster is often visualized based on bubbles, with the size and the colour (sometimes the shape as well) of each bubble representing different features of the data. Although bubble-based cluster maps can

present information that heatmaps and binned plots cannot, the locations of properties in each cluster could not be precisely represented, i.e., properties actually inside of the circle might not belong to the cluster (Example 2 in Section 3.4). To solve such problem, we provide a new map design to represent the geographical boundary of each cluster, i.e., boundary-based cluster maps (similar but the same as choropleth maps [BMPH97]).

A key question arising for the boundary-based cluster maps is *what is an appropriate choice of the polygons used to present the boundary of clusters?* To answer this question, we reviewed existing methods (e.g., KNN-based method [MS07], χ-shapes [DKWG08] and α-Concave Hull [ADM17]) that generate different polygons (hulls) based on the included geographical points. However, all of these methods suffer from at least one of the following problems: (i) large empty areas inside the hull, (ii) too complex shapes, and (iii) overlaps among the polygons representing the different clusters (e.g., Figure 3(d)). The problem are explained in greater detail in Section 4, with a summary of envisioned features of an appropriate choice in Table 1.

We propose a new concave hull algorithm, which generates a polygon (hull) to present each cluster based on the geographical locations of all data points in the cluster. In particular, our algorithm is based on a global Delaunay triangulation [Sam06], followed by a separation of points based on the different clusters; thus it is able to avoid all of the three aforementioned problems.

Following our proposed cluster-based visualization design, a subsequent challenge is: *what kind of a data structure is efficient to support such visualization design and user interactions on large-scale data?* Accordingly, we propose a tree-structured index called ConcaveCubes, which organizes the pre-computed hierarchical clustering result in data cubes. Compared to state-of-the-art visualization-driven cube structures, (i) ConcaveCubes supports cluster maps which existing structures cannot; (ii) ConcaveCubes exploits real-world geographic semantics (e.g., country, state, city) rather than using grid-based aggregations; (iii) instead of calculating the clusters on demand, it utilizes existing calculation and visualization results to efficiently support different kinds of user interactions, such as zooming & panning, filtering, and granularity control.

To summarize, we make the following contributions.

1. We present a cluster-based visualization abstraction of geographical data (Section 3).
2. We propose an algorithm to efficiently generate concave hulls that include geographical points in each cluster as compactly as possible. As a result, it is able to avoid large empty areas inside the hull, complex shapes and overlaps among different clusters (Section 4).
3. We propose a cluster-based data cube (ConcaveCubes) to efficiently support interactive response to users' visualized exploration on large-scale geographic multidimensional data (Section 5).
4. We conduct extensive experiments using real-world datasets, and compare ConcaveCubes with state-of-the-art cube-based data structures to verify the efficiency and effectiveness of ConcaveCubes (Section 6).

2. Related Work

In this section, we review the literature on large-scale data visualization (with a focus on that in geographic related data), which can be broadly classified into two categories: (1) data reduction methods; (2) visualization methods. It is worth noting that many studies exploit techniques in both categories to achieve a scalable solution.

2.1. Data reduction methods

We refer to data reduction methods as those techniques that aim to reduce the size of the data before the data enters the visual rendering process. We consider three main directions in reducing the scale of data for different scenarios or purposes.

Sampling & filtering methods. Traditional data reduction methods such as sampling [DP12] & filtering [AS94] are widely used to select a smaller subset of data before applying standard visualization techniques. However, as argued by Lins et al. [LKS13], sampling & filtering might elide interesting structures or outliers, and usually cannot provide an overview of the data distribution.

Approximate query processing methods. Several approximate algorithms (M4 [JJHM14], VDDA [JJHM16], etc.) are proposed by exploiting the fact that a display has a limited number of pixels based on its resolution [Kei96]. These algorithms return approximate results that rasterize to the same image as the exact query result would. However, they can only be applied to a very limited number of visualization methods, e.g., line charts and scatter plots.

Data cube aggregation methods. Data cubes [GCB*97] have been intensively studied in the area of data warehouses. In this technique, aggregation results of the raw data are pre-computed on predefined dimensions to support data exploration. However, a complete data cube is often too large to fit in memory and query in real-time. One recent research direction is to design visualization-constrained data cubes that are constrained by the number of pixels used in visualization [Kei96]. Different methods have been proposed to reduce the size of the data cubes for information visualization based on binned aggregation [CLNL87]. Lins et al. [LKS13] have suggested a design principle that *scalability should be limited by the chosen resolution of the visualization data, not the number of records*. Based on this principle, they proposed imMens, which decomposes the full cube into sub-cubes to minimize data memory usage and thus reduce the query latency. However, imMens can only support brushing and linking [Kei02] scenarios with at most four dimensions. Nonocubes [LKS13], on the other hand, support an arbitrary number of dimensions; but it might suffer from query latency and has a high memory cost, since it uses a large amount of binned aggregation. GaussianCubes [WFW*17] and TopKube [MLKS18] extend the idea of Nanocubes by supporting more types of queries, but still suffer from the high query latency as the number of dimensions increases. Hashedcubes [PSSC17] sorts data in advance based on judiciously selected pivots, thereby on-the-fly aggregation computation can be supported and the storage cost is reduced significantly.

Our proposed ConcaveCubes is in line with the data cube methods. As compared to state-of-the-art imMens [LJH13], Nanocubes [LKS13], and Hashedcubes [PSSC17], our ConcaveCubes distin-

guishes from them in three aspects. First, instead of storing individual geographic points, ConcaveCubes first clusters those data items that are close in distance and share similar features, and then builds indexes based on the clusters, i.e., cluster-based data cubes. Therefore, ConcaveCubes supports the visualization of clusters on top of the map, while existing methods only support heatmaps. Second, ConcaveCubes exploits real-world geographic semantics (e.g., country, state, city) rather than using grid-based aggregations. Third, instead of calculating everything on demand, ConcaveCubes utilizes existing calculation and visualization results to efficiently support different kinds of user interactions, such as zooming, panning and filtering.

2.2. Visualization reduction methods

Different techniques are proposed to address the perceptual and interactive scalability problem at the visualization end. In particular, the following three types of methods are proposed.

Visual reduction. Since the screen solution is limited, retinal variables to represent different attributes in a large dataset will easily clutter visually. To this end, researchers have proposed a number of methods, such as pixel-oriented visualization methods [Kei96], alpha blending [JS98], spatial displacement methods [TGC03], and dimension re-ordering [PWR04] for parallel coordinates. However, all of these techniques still need to scan and draw each data item, which will cause interactive scalability problem.

Progressive methods. Progressive visual analytics [SPG14] presents visualization results in a progressive way, so that the system will be responsive to users' interactions immediately. The method is also associated with approximate query processing techniques, which provide query result estimations with bounded errors [GCZ*17]. However, experiments conducted by Turkay et al. [TKBH17] have shown that possible wrong judgments are easily made before rendering is finished. Even with uncertainty in estimates [Fis11] reported during the progressive rendering, the result remains unconvincing until the whole progress is completed.

Predictive methods. Predictive visual analytics [LGH*17] often use query pre-fetching techniques, where the performance depends on the level of predictability of future queries. For example, the Atlas system [CXGH08] stores large historical time-series data, allows six directions of exploration (pan left/right, scroll up/down, and zoom in/out). The algorithm is based on the observation that a sense of momentum is associated with the direction of exploration. However, these techniques only work well in those domains where user interactions are naturally limited.

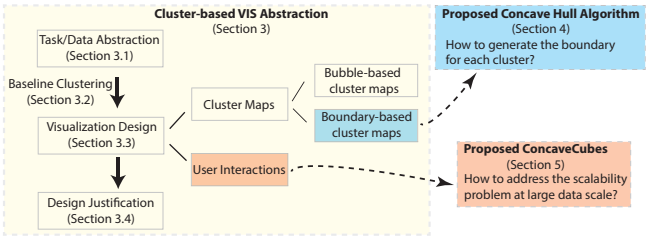


Figure 1: An overview of our solution in Sections 3, 4 & 5.

3. Cluster-based Visualization Abstraction

Figure 1 presents the structure of this section, and how it is linked with Section 4 and Section 5.

3.1. Data and task abstraction

For ease of explanation, we describe our work using Australia's real estate data as an example of multidimensional urban data. The real estate data include five profiles, i.e., basic profile, census profile, education profile, facility profile, and transportation profile (detailed in our previous work [LBS*18]). Each property is associated with its geo-spatial information (latitude and longitude), categorical dimensions (e.g., property type), and numerical dimensions (e.g., price).

We follow Shneiderman's visual information-seeking mantra [Shn96] to define our tasks, i.e., *overview first, zoom and filter, then details-on-demand*. We illustrate the abstraction of tasks based on a series of real-life questions: (1) *What is the difference of the property prices in different suburbs of Victoria (overview)?* (2) *What if the user only concerns about 3-bedroom houses (filtering)?* (3) *What is the difference between the houses in blocks of the same suburb (zooming & granularity control)?* (4) *Based on the user-selected attributes, how do the houses in a block differ from the rest of the houses in the same suburb, or the rest of the houses in that entire state (details-on-demand, highlighting & linking)?*

3.2. A baseline solution

Before describing our cluster-based visualization design, we introduce a baseline clustering solution. Here, we first subdivide different dimensions into four main types based on (i) whether they are directly linked to locations, and (ii) whether they directly affect the clustering result.

- **Geo-dependent dimensions**: including geographical locations represented by latitude and longitude, and geo-semantics such as *state*, *suburb*. These dimensions are directly linked to geography, and will directly affect the clustering result.
- **Additional geo-dependent measures**: e.g., the distance to the nearest train station, the median income of the region, etc. These dimensions are directly related to geography. However, once a group of real estates is close in distance, they share similar values of the other dimensions. Therefore, it is not necessary to consider those dimensions in the clustering process.
- **Geo-independent dimensions**: e.g., price, property type, bedroom number. They are major factors in the clustering process which are not directly linked to the geo-locations.
- **Additional geo-independent measures**: e.g., number of parking spots, whether containing air conditioning, etc. Users may want to check these factors for individual houses but they are not important enough to be considered in the clustering process.

Now we present the steps of a straightforward method to cluster the properties that are close in distance while also sharing similar features: **(i)** We group properties based on geo-independent dimensions, e.g., group all 3-bedroom houses with price between 0.9-1million. **(ii)** We further group the properties based on one level of geo-semantics (geo-dependent dimensions), e.g., all 3-bedroom houses in the same suburb will be grouped together. **(iii)** For each

group, we apply DBScan [EKSX96] to divide the properties into different clusters based on the geo-locations, e.g., the properties within the same group after step 2 and close in distance (e.g., <150m) will be in the same cluster.

3.3. Visualization design

To support the visualization tasks (Section 3.1), we design two map-based views and a linked multidimensional view. In the map-based views, we visualize the clusters on top of the map. When a cluster is highlighted in the map-based view, a linked multidimensional view presents the detailed information of the cluster based on user-selected measures. Justification of using cluster maps instead of other map designs (e.g., heatmaps that existing visualization-based data cubes support) will be presented in Section 3.4.

Map-based view 1: bubble-based cluster maps. A straightforward visual encoding to visualize clusters is to use bubble maps. Comparing to heatmaps that Nanocubes [LKS13] and Hashedcubes [PSSC17] support, bubble maps have at least one more retina variable (i.e., the size of the bubble) to encode an extra dimension (another variable is the shape). An example is shown in Figure 2(a), where the colour of each bubble represents the average price of properties in each cluster, and the size of the bubble represents the total number of properties in the cluster. We also use alpha blending [JS98] to reduce visual cluttering for intersecting bubbles.

Map-based view 2: boundary-based cluster maps. As boundaries are important measurements of the geographic information, our second map-based view draws concave hull (a polygon that covers a group of points) [MS07] on top of the map to illustrate positions of the properties in each cluster (Figure 2(b)). How to generate a concave hull based on a group of geographical points is presented in Section 4. The colour of the concave hull is based on an aggregated value of user-defined dimensions for all the properties in each cluster. This is done as follows: suppose that a user selects m dimensions, and there are n clusters. Let $\tilde{x}_{i,j}$ be the median value of all properties in cluster c_j ($j \in [0, n-1]$) at dimension i ($i \in [0, m-1]$). The aggregated value of a cluster c_j is computed as a weighted sum:

$$aggr_{c_j} = \sum_{i=0}^{m-1} w_i \cdot N(\tilde{x}_{i,j}) \; / \; \sum_{i=0}^{m-1} w_i \qquad (1)$$

where, $N(\tilde{x}_i)$ corresponds to the normalized value of $\tilde{x}_{i,j}$ between $[0,1]$ and computed as: $N(\tilde{x}_{i,j}) = (\tilde{x}_{i,j} - \min_j \tilde{x}_{i,j})/(\max_j \tilde{x}_{i,j} - \min_j \tilde{x}_{i,j})$, where w_i is the weight of the i-th dimension and is defined by the user. In our implementation, the weight of each dimension is generated using the Random Weighted Genetic Algorithm (RWGA) [MI95] after the user gives the importance rank of all dimensions. In particular, when the user selects only one dimension (e.g., price), the colour is mapped based on the selected dimension.

Multidimensional view: line charts + histograms. We design a linked multidimensional view to visualize the statistical information of a selected (highlighted) cluster and compare it with others. For example, in Figure 2(b), after the user selects a cluster in Brighton, the statistical information of this cluster based on seven user-defined dimensions is shown as seven histograms in the multidimensional view; also, the information in the entire state (Victoria) on this seven dimensions are shown as seven line charts for comparison.

Interaction design. Based on our task abstraction, we support the following user interactions:

- **Zooming & panning**: we support zooming & panning on map.
- **Filtering**: we support filtering data values on each dimension. The cluster will be re-calculated as a result of filtering. Filtering is performed using the selection panel shown in the top right conner of Figure 2(b) that consists of sliders and drop-down menus.
- **Granularity control**. We support map-based visualization in multi-granularity scales. In our implementation, after users zoom in/out on top of the map, the granularity level will also be changed. However, users can also change to a finer or coarser level of granularity by selecting from the selection panel while staying at the same zooming level.
- **Other interactions**, such as highlighting & linking (i.e., selecting and highlighting a cluster on the map and linking the information in the multidimensional view).

3.4. Design justification: why cluster-based visualization?

There are various map designs to visualize geographical data, including symbol maps, heatmaps, bubble maps, cluster maps, etc. We observe that, existing visualization-driven data cube structures (e.g., imMens [LJH13], Nanocubes [LKS13], Hashedcubes [PSSC17]) only support heatmaps or dot maps based on binned aggregation. Although Liu et al. [LJH13] have shown that heatmaps have advantages in comparison to sampled symbol maps (e.g., heatmaps can show the overall geographical distribution while sampled symbol maps cannot), the information that heatmaps can provide is still very limited, as illustrated in Example 1.

Example 1 Figures 3(a) and 3(b) show a visualization of Melbourne's real estate data using heatmap and binned plots (a variation of heatmaps), respectivelty. Heatmap, which is the only form of visualization supported by existing cube structures, can only present the density of real estates in different regions (by colour). Binned plots, which can be supported by the existing methods by slight adjustments, can provide price distribution (by colour) (Figure 3(b)); however, the plots are not easy to interpret as each binned block does not have real-life semantic meanings. For example, the real estates in a binned block may not be in the same suburb and/or may have different key features (e.g., they may belong to different school zones).

On the other hand, cluster maps, which are difficult to support by existing data cubes, provide more information (with semantic meanings) in comparison to heatmaps or binned plots: Example 2.

Example 2 We visualize Melbourne's real estate data using bubble-based cluster maps (Figure 3(c)). Each bubble represents a group of real estate properties which are clustered based on both geo-locations and two other features (corresponding to suburbs & secondary school zones). The colour and size of each bubble represent the median price and total number of properties in each cluster, respectively. Compared to heatmaps and binned plots, bubble-based cluster maps (i) represent one more dimension, and (ii) have a real semantic meaning of the visual mark - each bubble represents a group of houses that are in the same suburb and school zone as well as close in distance.

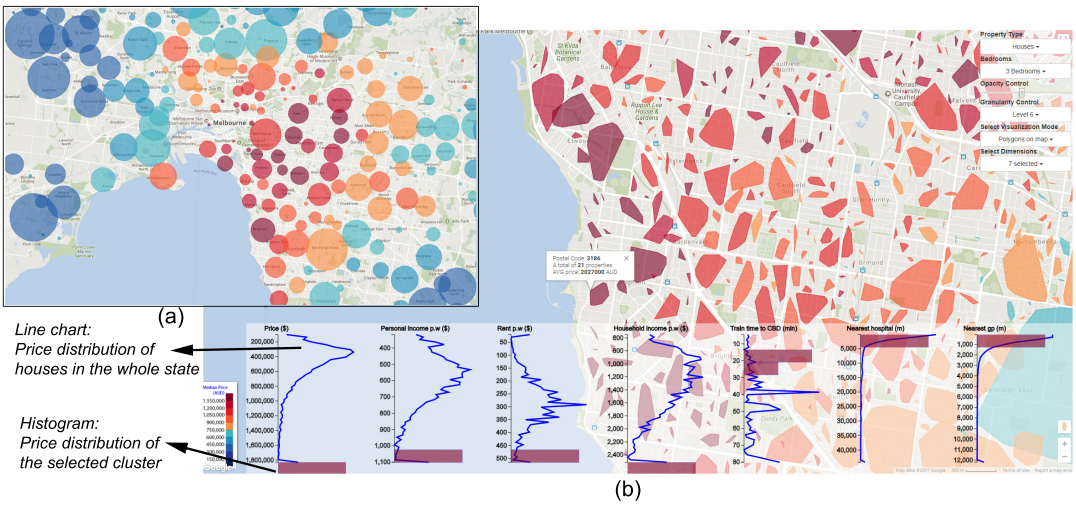


Figure 2: Illustration of our visualization design. (a) map view 1: bubble-based cluster map; (b) map view 2: boundary-based cluster map, and the multidimensional view with a cluster highlighted (properties in the selected cluster is compared to that of the entire state based on 7 user-selected measures (sold price, personal income, rent price, household income, distance to CBD, nearest hospital and nearest doctor.

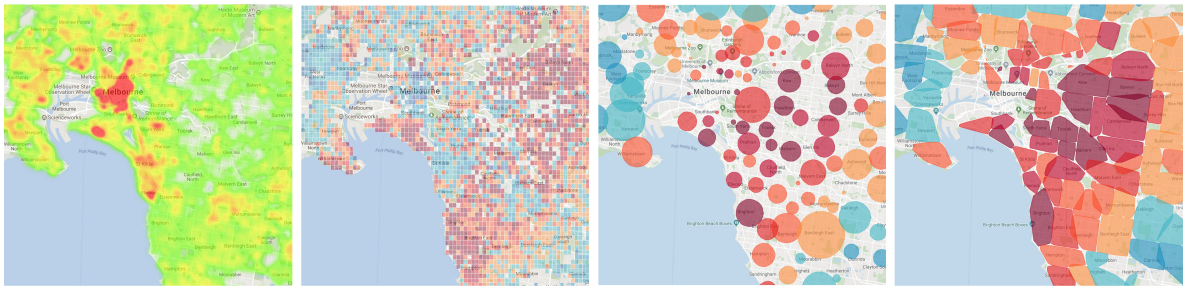


Figure 3: A comparison of 4 geographical visualizations based on the real estate data in Melbourne. (a) Heatmaps and (b) binned plots, which imMens [LJH13], Nanocubes [LKS13] and Hashedcubes [PSSC17] support; (c) bubble-based cluster maps and (d) boundary-based cluster maps that our proposed ConcaveCubes supports.

Although traditional bubble-based cluster maps can present information that heatmaps and binned plots cannot, the locations of properties in each cluster are not precisely represented, i.e., properties actually inside a circle might not belong to the cluster. Our proposed boundary-based cluster maps aims to solve this problem, as shown in Example 3.

Example 3 We visualize the same data using boundary-based cluster maps (Figure 3(d)). Each cluster is represented by a colour-encoded hull (boundary of the cluster). Compared to the heatmaps or binned plots in Figure 3(a & b), boundary-based cluster maps present geographical information with semantic meanings. Comparing to our baseline bubble-based cluster maps which present one more measure using the size of each bubble, the boundary-based cluster maps present more precise location information.

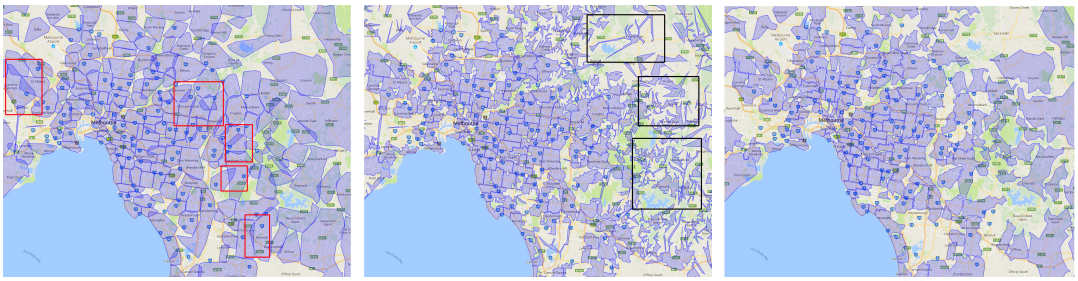
4. Algorithm for Generating Boundary-based Cluster Maps

In this section, with the clusters as the input, we present our algorithm to generate the boundary of each cluster that best represents the group of geographical points that are physically located in it. In particular, we first outline the desired features (from the perspective of visual perception) of a boundary generation algorithm. Next, we show that the concave hull based representation is the most suitable choice. Third, we propose a novel concave hull construction algorithm that meets all of these desired features. Finally, we compare the visualization result and the time complexity of our proposed algorithm with the existing methods.

4.1. An outline of desired features

We envision four features for an ideal cluster map: (1) no overlap between the visualization of any two clusters, (2) complex shapes should be minimized to enhance user's cognitive capability, (3) empty areas of each hull [GD06] (i.e., the area inside a hull that does not contain any real estate properties) should be minimized, (4) parameter tuning for boundary generation should be avoided.

Generating hulls (polygons) to represent the geographical boundary of each cluster is one of the approaches to achieve these features. We first review existing hull generation algorithms. Ebert et al. [EBN15] classify hulls into five different categories: (i) *rectangular hull*, (ii) *orthogonal convex hull*, (iii) *convex hull*, (iv) *concave hull*, and (v) *concave hull for several groups*. Since our target is to represent each cluster with a single polygon, *concave hull for several groups* (representing one cluster) is not appropriate in our case. Among the other categories, the concave hull is recognized as presenting the most precise shape recognition of an area [EBN15].



(a) (b) (c)

Figure 4: The result of an existing concave hull algorithm [PO12] (representing all the concave hull methods in Table 1) [Red - overlaps, black - complex shapes] and the result of using our proposed algorithm: (a) using the method in [PO12] with a threshold of 100m; (b) using the method in [PO12] with a threshold of 800m; (c) using our proposed concave hull construction approach.

Table 1: A comparison between our proposed algorithm and 7 existing hull construction algorithms.

		Overlaps	Complex Shape	Empty Area	Parameter Needed	Complexity
Convex Hull	Graham Scan [Jar73]	Yes	No	Yes	No	O($n$log$n$)
Convave Hull	χ-shapes [DKWG08]	Yes	Yes	No	Yes	O($n$log$n$)
	KNN-based method [MS07]	Yes	Acceptable	No	Yes	O(n^3)
	α-Concave Hull [ADM17]	Yes	Yes	No	Yes	O(nhlog$n$)
	Jin et al. [PO12]	Yes	Yes	No	Yes	O(nhlog$n$)
Others	Alani et al. [AJT01]	No	No	Yes	No	O($n$log$n$)
	Arampatzis et al. [AvKR*06]	No	No	Yes	No	O($n$log$n$)
Our proposed concave hull algorithm		No	No	No	No	O($n$lng$n$)

Therefore, we opt for a concave hull based representation for the positions of properties in each cluster.

Table 1 shows a comprehensive list of limitations of seven major hull construction algorithms. Although the convex hull algorithm does not generate complex shapes, it usually results in large empty areas and will easily cause overlaps. While concave hulls generated by χ-shapes [DKWG08] and the KNN-based method [MS07] can reduce empty areas, they either have overlaps or complex shapes. The Voronoi-based [AJT01] and the triangulation-based [AvKR*06] concave hull method fully partition the whole space, thus a huge empty area can be generated inside each hull.

Moreover, concave hulls are usually not unique, i.e., the constructed concave hulls can be different in different algorithms and/or with different parameters. Although different algorithms have been proposed to generate a concave hull, there is no standardization on the effectiveness of the concave hull, i.e., determining which concave hull algorithm generates a region that best represents a group of points is inconclusive [GD06]. Even with the same algorithm, determining the parameter settings is a difficult problem.

Figures 4(a & b) illustrate the common limitations of the existing Concave hull methods with real-life data. We have implemented two existing methods [PO12, MS07] and applied them to our real estate dataset, where each cluster includes properties in a postal area. The algorithm by Jin et al. [PO12] first constructs a convex hull, and then execute a "digging" process (progressively replace an outer edge with two close inner edges) to get a concave hull. The only parameter here is a threshold that controls when to terminate. We visualize the results on Google Maps, and observe that: a larger threshold as shown in Figure 4(a), may cause lots of overlaps in dense areas (e.g., CBD areas which have more properties); while a smaller threshold will result in very complex shapes in the less dense area and users may fail to recognize the shapes (Figure 4(b)). We also implemented the KNN-based method [MS07], which does not generate too complex shapes, but cause significant overlaps similar to Figure 4(a).

Algorithm 1: CONCAVE_HULL (P)

```
1.1  Input: A set P of geographical points, each p ∈ P has attributes {latitude,
     longitude, cluster}.
1.2  Output: A list of concave hull boundaries for each cluster.
1.3  DELAUNEY_TRIANGULATION(P)
1.4  TE ← Set of resulting triangulation edges
1.5  for each edge e = ⟨p1, p2⟩ ∈ TE do
1.6      if p1.cluster ≠ p2.cluster then
1.7          delete e from TE
1.8  Delete all inner edges from TE
1.9  while ∃p : degree(p) = 1 do
1.10     p1 ← the node that p connects;
1.11     if degree(p1) = 1 then
1.12         delete ⟨p, p1⟩ from TE;
1.13 while ∃p : ∃∠p′pp″ where p′ and p″ do not form a loop do
1.14     Ap ← All outer angles created at p;
1.15     ∠p′pp″ ← The smallest angle in Ap;
1.16     if ⟨p, p′⟩ is not an exposed line then
1.17         delete ⟨p, p′⟩ from TE;
1.18     if ⟨p, p″⟩ is not an exposed line then
1.19         delete ⟨p, p″⟩ from TE;
1.20     add ⟨p′, p″⟩ to TE;
1.21 c_edges ← BOUNDARY_EDGES_GROUP_BY_CLUSTERS(TE)
1.22 RETURN c_edges
```

4.2. Our concave hull construction approach

To address the limitation of existing concave hull generation methods, we propose a novel algorithm to meet all the four envisioned features. The pseudocode of our proposed approach is presented in Algorithm 1 and illustrated in Figure 5. The input of the algorithm is a set of geographical points, P. Each point $p \in P$ has

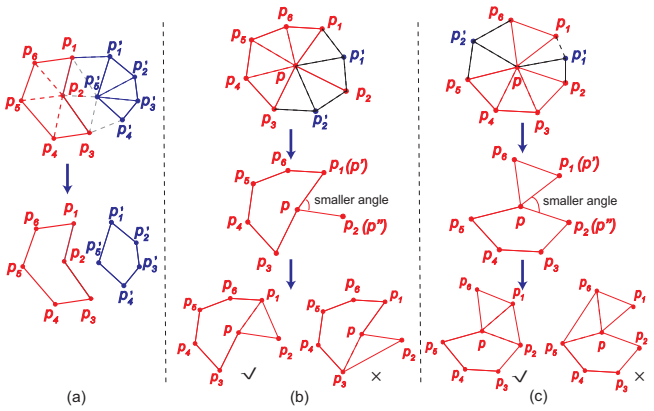


Figure 5: Illustration of the proposed Concave hull algorithm. (a) the process of separating clusters after a global triangulation; (b-c), two examples of non-Jordan boundaries: how the boundary is formed and how we eliminate it.

three attributes, where latitude and longitude jointly represent its geographical location, and cluster *p.cluster* indicates the cluster in which the point belongs. The output of the algorithm is a set of concave hulls. Each concave hull corresponds to a unique cluster ID, and its boundary should be a Jordan curve. The Jordan curve restriction is commonly used in all the methods in Table 1. The reason for adopting this restriction in our cases is quite straightforward: the boundary for each cluster should be a plane simple closed curve that divides the whole space into exactly two regions - the one inside the cluster, and the one outside it.

At a high level, the steps of the algorithm are: (i) Generate a Delaunay triangulation for P. (ii) To reduce overlaps, separate clusters by deleting edges that connect two nodes in different clusters. (iii) Remove the inner edges to get the outer boundary of the cluster, which ensures no additional empty area is included. (iv) If a non-Jordan curve is generated in this process, then apply our proposed novel edge replacement technique to remove such complex shapes.

We now describe the details of our algorithm. The terms 'point' and 'node' are used interchangeably for the rest of the algorithm.

As shown in Line 1.3 of Algorithm 1, we first generate a Delaunay triangulation for all the points in P. Next, we delete the edges that connect two nodes in different clusters to seperate the clusters (Line 1.5-1.7). We further detect the outer edges by deleting all edges that appear twice in the triangulation result (Line 1.8). As illustrated in Figure 5 (a), a concave hull would be generated for each cluster in most cases. However, since the triangulation is generated based on all points, after we separate the clusters, there could exist exposed lines (explained later), e.g., $\langle p, p_2 \rangle$ in Figure 5(b) or point connections [GD06] (e.g., p in Figure 5(c)), which make the hull a non-Jordan boundary.

Eliminating non-Jordan boundaries. The degree of each node of a Jordan boundary must be equal to 2 where the nodes form a closed loop. Therefore, if the resulting concave hull of a cluster in the previous step is a non-Jordan boundary, that implies there must be at least two nodes with degree not equal to 2.

We denote an edge as an 'exposed line' if the edge is not part of any loop (e.g., $\langle p, p_2 \rangle$ in Figure 5(b)). Let an angle $\angle p_i p p_j$ created at node p be denoted as an 'outer angle' where the other points p_i and p_j do not form a closed loop with each other (i.e., they are either involved in two different loops - $\angle p_1 p p_2$ or $\angle p_5 p p_6$ in Figure 5(c), or at least one of them is connected with an exposed line - $\angle p_1 p p_2$ or $\angle p_2 p p_3$ in Figure 5(b)). According to the properties of a Jordan curve, where the nodes of a cluster should form a single loop, there must not be any such outer angle. Thus we transform the problem of obtaining a Jordan curve from a non-Jordan one to the problem of eliminating any outer angles. We present the steps of our technique in the following:

Let p be a node such that at least one outer angle is created at p. We find the set A of all the outer angles created at p. We find the smallest angle from A (we will explain why we deal with the smallest angle in Example 4). Let that angle be $\angle p' p p''$ (Lines 1.14 - 1.15). As the nodes p' and p'' of an outer angle do not form a closed loop with each other, we create an edge by adding p' and p'' (Line 1.20). Then, if the edge $\langle p', p \rangle$ was not an exposed line, we remove $\langle p', p \rangle$ (as the degree of p' was at least 2, and its degree got increased by 1 because of the new edge). Similarly, if the edge $\langle p, p'' \rangle$ was not an exposed line, we remove $\langle p, p'' \rangle$ as well (Lines 1.16 - 1.19). We continue this process until there is no node left with any outer angle, i.e., the Jordan curve is obtained. We present the steps of eliminating non-Jordan boundaries in Lines 1.9 - 1.20 in Algorithm 1. There is a special case where there are only two nodes in a cluster. In such case, we disconnect them by removing the edge (Lines 1.9 - 1.12).

Example 4 Figures 5(b & c) present two examples of how the outer angles are formed and how we eliminate them. In Figure 5(b), since the red points and the blue points belong to different clusters, after we remove the gray edges to separate the two clusters (Lines 1.5-1.7), $\langle p, p_2 \rangle$ becomes an exposed line. There are two ways to eliminate the exposed line by dealing with either of the two outer angles $\angle p_1 p p_2$ or $\angle p_2 p p_3$. We choose to deal with the smaller angle $\angle p_1 p p_2$ (rather than $\angle p_2 p p_3$) by replacing pp_1 with $p_1 p_2$, which will generate a smaller interior angle (the interior angle $\angle p_2 p p_3 < \angle p_1 p p_2$ in the final hull). Based on the complexity function defined in [BKSB95], a smaller interior angle will reduce the chance of notches, which will also reduce the complexity of the concave hull. Similarly in Figure 5(c), dealing with the smaller outer angle $\angle p_1 p p_2$ will also generate a less complex hull.

Time complexity. The time complexity of the proposed algorithm is mainly affected by the triangulation process, which is O($n$log$n$) [DKWG08], where n is the number of points in P. The process of separating clusters and deleting all inner edges (Lines 1.5-1.8) depend on the number of edges in the triangulation result which is proved to be no larger than $3n$ [Sei95]. The complexity of the core algorithm loop (Lines 1.9-1.20) is based on the number of special cases - outer angles (s). In real case, $s \ll h \ll n$, where h is the number of points that lie on the final hull boundaries. Therefore, the overall complexity of the algorithm is O($n$log$n$).

The result. Figure 4(c) presents the result of concave hulls using our proposed algorithm (with the same real estate data as in Figure 4(a & b)). Our proposed algorithm, which does not need any parameters, is able to avoid all three problems: overlaps, complex shapes, and empty areas.

5. ConcaveCubes: Cluster-based Data Cubes index

In this section, we propose ConcaveCubes to support the visualization design (Section 3.3) on large-scale data from a data reduction perspective. We first give an overview of ConcaveCubes followed by an example, and then describe how we construct ConcaveCubes. Finally, we discuss how different user interactions are supported.

ConcaveCubes is a data cube structure that precomputes clusters to support visualization and user interactions. ConcaveCubes addresses the scalability issue from the data reduction perspective as: (i) the hierarchical structure supports accessing only the data points necessary for the current view during visualization and user interactions (explained in Section 5.3); (ii) the precomputed aggregations are used to reduce the on-the-fly computation without accessing the actual data points.

5.1. An example of ConcaveCubes

ConcaveCubes has a hierarchical structure with multiple levels, and each level corresponds a specific dimension type: recall the baseline clustering method defined in Section 3.2, we have subdivided dimensions into four types based on whether they are directly linked to geo-locations and whether they are going to affect the clustering result. To generate ConcaveCubes, each level of ConcaveCubes corresponds to one type of dimensions: as geo-independent dimensions, geo-semantics and geo-locations.

Figure 6 presents an example of ConcaveCubes with Australia's real estate data. The data is first sorted in the first 4 levels. The nodes at the last geo-semantic level will be split if they belong to different clusters at the next level. For example, the highlighted node (in orange) at level 4 contains all 2-bedroom units in Melbourne, Victoria. It is split into two nodes ($C1$ & $C2$) based on the clustering function at level 5, and $C1$ is further split into two nodes based on a finer clustering function at level 6. For each node in level 5 & 6, we store the statistical information of the properties in each cluster (shown as $[i,j]$ in Figure 6, where i and j indicate the start and end index of properties, respectively).

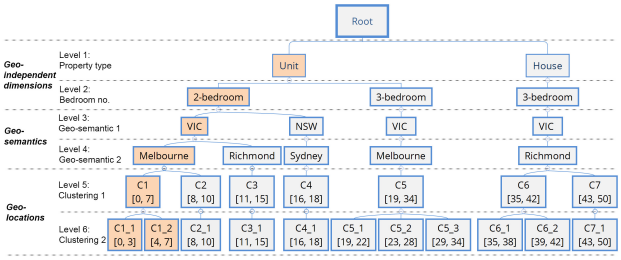


Figure 6: An example of ConcaveCubes with the real estate data.

5.2. Construction of ConcaveCubes

As shown in Figure 7, ConcaveCubes is constructed in four main steps: (1) data pre-processing, (2) a multi-criteria sorting on the categorical and geo-semantic dimensions, (3) hierarchical clustering, and (4) cluster-based calculations.

Data pre-processing. Before generating ConcaveCubes, we pre-process the raw data as follows: (i) for each geo-independent dimension and each geo-semantic level, we set an ordering of the dimensions; (ii) we divide the value of geo-independent dimensions

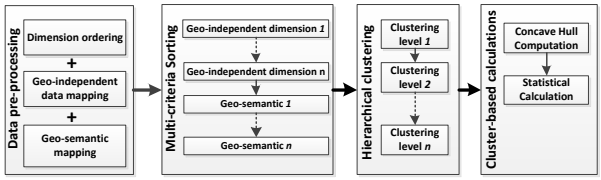


Figure 7: The procedure of constructing ConcaveCubes.

based on specific values or ranges, which convert them into categorical values; (iii) we map each data point to its corresponding geo-semantic region (e.g., a property might be mapped to Australia → Victoria → Melbourne).

Multi-criteria sorting. Similar to Hashedcubes, our ConcaveCubes also needs a multi-criteria sorting. Note that, our geographical level is different from that in state-of-the-art data cubes. In Nanocubes and Hashedcubes, the geographical features are processed based on the Quadtree, a grid-based hierarchical spatial data structure that recursively divides the space into four regions based on either its interior [ST86] or its boundary [SW84]. Such a structure is effective for querying on the geographical dimension, but might not preserve the geographical semantics. For example, if we partition the data based on geo-locations of properties using a Quadtree, in one of the middle levels, two properties from two different states may belong to the same grid, while they have no semantic relationship. Therefore, we design the geographical level of ConcaveCubes based on real-world geographical hierarchy, such as country, state, city, etc.

Hierarchical clustering. After the multi-criteria sorting, all properties having the same values in geo-independent dimensions and in the same geo-semantic area will be grouped together. However, properties in the same group can still be quite different. For example, in Figure 2(b), houses that are close to the beach are more expensive than others even when they are in the same suburbs. Therefore, we apply hierarchical clustering [Mur83] after the multi-criteria sorting to further divide the data into different groups. In our implementation, we use a hierachical DBScan [CMS13] by deceasing the threshold of the algorithm in a lower level. For example, at level 5 in Figure 6, we apply DBScan [EKSX96] with a threshold of 400m to cluster the properties based on Euclidean distance. Then at level 6, we re-apply DBScan with a smaller threshold (100m in our demonstration) to possibly divide some of the original clusters into multiple smaller clusters.

Cluster-based concave hull calculation. We calculate a concave hull for each cluster in ConcaveCubes based on the algorithm defined in Section 4 to support the boundary-based cluster maps. Since out proposed concave hull method is based on a global triangulation, we apply the proposed algorithm on a group of clusters that share a parent-node. Instead of storing the locations for all data points, we store the concave hull boundaries. At last, for each cluster that has a concave hull (clusters with less than three points will not have a hull), we calculate the statistical information based on each geo-dependent and geo-independent measures (Section 3.2) and store them in the database. The statistical (aggregated) information is stored as views in the database, so that when a cluster is highlighted, the information of the selected dimension measures will be efficiently loaded to visualize.

5.3. Index operations

In this subsection, we illustrate the operations that ConcaveCubes provides to support the visualization design and user interactions defined in Section 3.3.

5.3.1. Initialization

The initialization of the visualization is a query on ConcaveCubes with four parameters, a default map window, geo-independent dimensions (e.g., 3-bedroom houses), a default clustering granularity (e.g., a 400m DBScan), and a focused measure (e.g., price). Based on the default settings, ConcaveCubes is traversed from the root and the following steps are executed: (i) the nodes matching the geo-independent dimensions are scanned, (ii) the nodes in geographic semantic levels within the map window are accessed, and (iii) the clusters at the default granularity level are obtained.

5.3.2. User interactions

The key idea for ConcaveCubes to support different user interactions is to reuse existing calculations and visualizations while updating for user interactions.

Interaction 1: zoom-in, zoom-out, panning. Zooming and panning directly influence the map window (w). The change of w corresponds to the level of geographic semantics in ConcaveCubes. Therefore, after users apply zooming in/out or panning, we calculate a set subtraction between the current (w_c) and previous (w_p) map windows, and then refresh the visualization based on the subtraction result. Since the boundary of a semantic geographical node often contains many geo-points, we store a minimum bounding rectangle for each geo-boundary. If the map window covers the entire rectangle, we include all the clusters in the lower level; otherwise, the next geographic semantic level is accessed.

Interaction 2: filtering. Filtering, i.e., selections, would affect the level of categorical dimensions. For example, if previously only 3-bedroom *units* in Melbourne are shown on the map, and the user selects to include 3-bedroom *houses* as well, then the corresponding clusters at level 5 will be added to the visualization result while the previous result remains. Note that, having multiple selections might result in some intersecting concave hulls. For each concave hull, we store a minimum bounding rectangle that covers it, so that we can effectively detect intersections and compute the union of the intersected concave hulls at runtime.

Interaction 3: granularity control. For example, in Figure 6, if the user changes the current granularity level from Level 5 to Level 6, the clusters that are not split will stay in the map window, but they will fall into multiple smaller clusters. For example, $C1$, $C5$ and $C6$ at level 5 will be replaced by their child clusters at level 6.

6. Experiments

In this section, we present an experimental evaluation of ConcaveCubes using several real-life datasets. We first describe the datasets and schemas, and report the storage and construction time of ConcaveCubes. We then present several visualization results, and compare our result with the result generated based on existing visualization-based data cubes. Finally we report the query time based on a set of real-world queries.

6.1. Datasets and schemas

We have collected five datasets that include two real estate datasets, a census dataset, and two location-based social network datasets. We summarize all of the schema variations and datasets in Table 2.

- **Real estate dataset**. We have crawled and cleaned the real estate data in Australia using the data profiles defined in our previous work [LBSY16]. We include two schemas in our experiments as shown in Table 2. The geo-semantic levels are based on ASGS (Australian Statistical Geography Standard [ABo16a]). Both the schemas include six levels of a hierarchical DBScan.
- **Location-based social network datasets**. Brightkite and Gowalla are two social network datasets with users' check-in locations [CML11]. For Brightkite, we build ConcaveCubes using three different schemas: two of them share the same categorical dimensions with imMens [LJH13], Nanocubes [LKS13], and Hashedcubes [PSSC17]. The geo-semantics are all mapped based on GADM (Database of Global Administrative Areas [Are12]).
- **Australian census dataset**. It includes four levels of geo-semantics based on ASGS. We simulated n data points which belong to each of the 57,523 SA1s (the smallest area for the release of census data [ABo16a]), where n is the population in each area. As a result, we generated 24.5 million data (the total number of Australian population at the 2016 census report [ABo16b]). Then, we select 100 features (including median age, income, etc.) from the 2016 census data of Australia.

We report the memory usage and construction time of ConcaveCubes in Table 2. Although the construction time of ConcaveCubes is much higher than that of Nanocubes and Hashedcubes, ConcaveCubes is much more memory efficient than the other two methods. On the one hand, for the construction time, the bottleneck of our construction process includes (i) geo-semantic mapping, (ii) hierarchical DBScan computation, and (iii) Concave hull calculation. The construction time of ConcaveCubes is proportional to the number of objects, and the number of hierarchical clustering levels. However, with more categorical dimensions or geo-semantics levels, the data entering the hierarchical DBScan process will be divided into more sub-groups, which results that the construction time of ConcaveCubes could possibly decrease. On the other hand, as examples of the memory efficiency of ConcaveCubes, it requires less than 30MB of memory for both the Brightkite and Gowalla datasets, where HashedCubes and requires more than 300MB [PSSC17] and Nanocubes needs more than 1GB [LKS13]. The reason behind this is that, we store the geo-semantic information for the geographic dimensions which are different with Nanocubes and Hashecubes, which use Quad-trees; as a result, much fewer data aggregations will be needed in the bottom layers of ConcaveCubes.

6.2. Visualization results

In addition to the visualization of real estate data shown in the previous sections (Figure 2 & 3) and demonstrated in [LBCS18], here we present visualization results based on the other datasets and compare them with existing methods.

Based on the Brightkite dataset, Figure 8 compares the visual-

Table 2: Overall summary of the relevant information for building ConcaveCubes.

dataset	objects (#)	clusters (#)	memory	time	geo-scope	geo-semantics	categorical dimensions	h-clustering	measures (#)
Real_estate_VIC	667k	50k	23.4MB	04m:16s	Victoria	3 levels	pro_type, bedroom	6 levels	36
Real_estate_AU	1.41m	104k	49.9MB	09m:08s	Australia	4 levels	pro_type, bedroom, year	6 levels	36
Brightkite	4.5m	489k	98.4MB	00m:58s	World	3 levels	hour, day, week[1]	-	1
		570k	117.3MB	01m:27s	World	3 levels	month, hour, day[2]	-	1
		33k	4.2MB	25m:00s	World	6 levels	-	3 levels	1
Gowalla	6.4m	37k	4.5MB	48m:03s	World	6 levels	-	3 levels	1
Census_AU	23.4m	57k	7.9MB	00m:35s	Australia	4 levels	-	-	100

[1] categorical dimensions used in Nanocubes and Hashedcubes; [2] categorical dimensions used in imMens and Hashedcubes.

ization results supported by ConcaveCubes with the visualization supported by Nanocubes, Hashedcubes, and imMens. The two visualizations present both global patterns (e.g., an overview of the check-ins across the United States) and local features (e.g., interstate highway travel). However, the boundary-based cluster maps, which ConcaveCubes supports, present important geo-semantic information which are missed in Heatmaps. For example, each polygon (cluster) in Figure 8(a) represents a group of data points within the same county and close in distance.

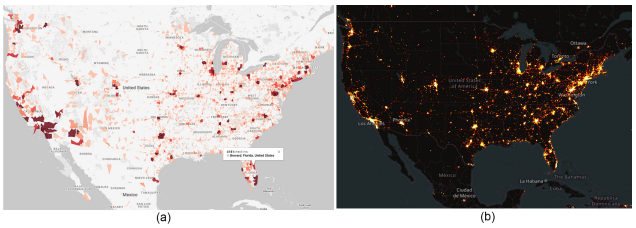


Figure 8: Visualization of the Brightkite dataset. (a) cluster maps (with a cluster selected) supported by our proposed ConcaveCubes: colour represents no. of check-ins; (b) heatmaps supported by Nanocubes, Hashedcubes, and imMens (screenshot is taken from the online demo of Nanocubes).

As shown in Figure 9(a), we generate a cluster map based on a group of simulated data points at each geo-semantic level in the Australia census data. ConcaveCubes is able to provide an efficient approximate solution of cholepleth map [BMPH97] by simulating geographic points in each region and calculating approximate boundaries based on the simulated points: comparing to the original dataset which takes more than 200MB to store the geographic boundaries, ConcaveCubes needs only 7.9MB of memory.

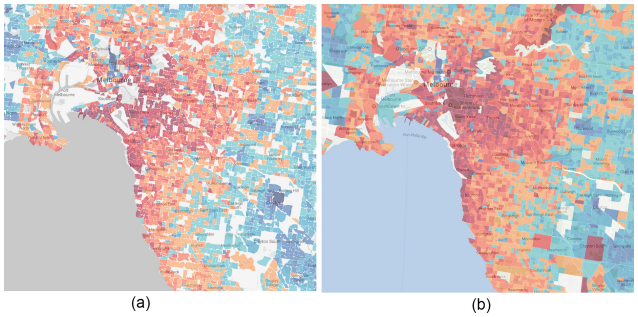


Figure 9: Visualization of the Australian census data: ConcaveCubes is efficient to support cluster maps (a), which could be an approximate solution of the traditional cholepleth maps (b).

6.3. Query time

We report the query time based on the real usage of the system. A total of 6,317 query requests were collected on the public ConcaveCubes web site [LCB*18], in which users performed different user interactions based on the real estate data in Victoria. To evaluate the query efficiency in a fixed environment, we recorded the query related to each user interaction and re-executed it in Google chrome on a quad-core i7-5500U CPU (2.4GHz) with 8GB RAM. Figure 10 shows the result based on six different types of user interactions.

Comparing to Hashedcubes which have only about 2% of the queries taking longer than 40ms (as stated in [PSSC17]), most of the queries in ConcaveCubes are responded within 25-50ms. The reason of why ConcaveCubes might take a slightly longer time is that: we exploit geo-semantic information in ConcaveCubes and display the boundaries for each cluster, while Hashedcubes do not need to deal with the boundaries. Nevertheless, all of our queries (from the experiments) are returned within 0.1s, which satisfies the perceptual processing time constant defined by Card et al. [CRM91]. Among all different types of user interactions, highlighting & linking is the most efficient operation: it retrieves the aggregation information of that cluster from ConcaveCubes directly and then displays them, so no additional calculation is required; granularity control takes slightly longer time than the other operations, since users might select a finer granularity in a high zoomed level, which means several levels of hierarchical clustering levels might need to be accessed.

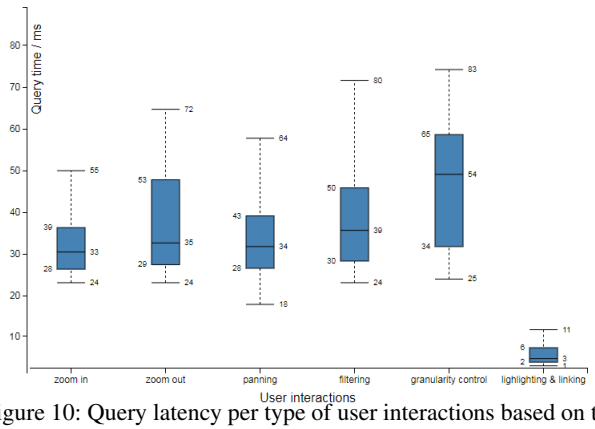


Figure 10: Query latency per type of user interactions based on the real estate dataset.

6.4. Discussion

Based on the experimental results, we have demonstrated that ConcaveCubes supports both bubble-based and boundary-based clus-

ter maps, which is not straightforward to be supported by existing methods (e.g., imMens, Nanocubes, Hashedcubes). Compared to heatmaps and binned plots, bubble-based cluster maps present additional information (with the color and the size of each bubble), and our proposed boundary-based cluster maps present more precise geographical information with semantic meanings.

Situations where cluster maps are most applicable. Our proposed boundary-based cluster maps can be regarded as a generalization of choropleth maps: the surface of each single color element is not directly the representation of an explicit surface delimitation (district limits for instance), but the result of the aggregation/clustering of data. Our methods are mostly suitable to apply when geo-referencing is just point-based or when the query needs to go beyond known space delimitations such as districts (real estate and social network examples). From the example of Census data, we also demonstrate that, even when the boundary of geo-semantics is given, ConcaveCubes can provide an approximate yet more efficient solution of the traditional choropleth maps.

Limitation of cluster maps. While cluster maps that the proposed ConcaveCubes support have advantages over other map designs, the method also has its limitations. For examples, bubble-based cluster maps lack of precise geographic information, and boundary-based ones might not present density information. One future research direction is to conduct a proper user study to compare different types of map designs (including the above-mentioned ones and some other map designs such as dot maps and cartogram maps) based on different visualization tasks.

Limitation of dimensions supported by ConcaveCubes. Apart from the disadvantages of the supported cluster maps, another main limitation of the proposed ConcaveCubes is the number of dimensions that it supports, which is a common problem for cube-based data structures. Similar to Nanocubes and Hashedcubes, ConcaveCubes can support at most 5-10 categorical dimensions. It is worth noting that, besides the geographic dimensions and categorical dimensions, ConcaveCubes is able to support temporal dimensions using the same method defined in [PSSC17] by converting the temporal dimensions into categorical dimensions.

Supporting dynamic data. Although ConcaveCubes is predefined in all the experiments, one possible extension of our work is to adapt the method with dynamic data. To insert, update or delete a data item in ConcaveCubes after it is already constructed, we only need to change the affected nodes in the hierarchical structure. For example, to insert a new 2-bedroom unit at Melbourne to the ConcaveCubes shown in Fig. 6, we need to find the corresponding nodes in Levels 1-4 that the new property belongs to (the highlighted nodes); then based on whether the new property is within the concave hull boundary of any existing clusters ($C1$ or $C2$), we update the corresponding node, or add a new node (if the new property does not belong to either of the clusters).

7. Conclusion

In this paper, we proposed ConcaveCubes, a data cube based scalable approach to support interactive visualization of large-scale multidimensional data with geographic semantic captured. In particular, we first presented a visualization abstraction for the geographical data based on clusters and proposed a visualization design of two map-based views (bubble-based as a baseline and boundary-based cluster maps as our desired map view) and a linked multidimensional view. For an effective boundary-based cluster map view, we proposed a novel concave hull construction algorithm, which addresses the limitation of existing methods by eliminating empty areas, complex shapes, and overlaps among different clusters. The ConcaveCubes utilizes the existing calculations based on the categorical and geo-semantic dimensions of the clusters and visualization results to efficiently support various types of user interactions. Extensive experiments on real-world datasets verified the efficiency and effectiveness of our proposed ConcaveCubes.

Acknowledgement

This work was partially supported by ARC DP170102726, DP170102231, DP180102050, and National Natural Science Foundation of China (NSFC) 61728204, 91646204. Zhifeng Bao is supported by a Google Faculty Award. Hanan Samet is supported in part by the National Science Foundation of the US under grant IIS-13-20791. The authors would also like to thank the anonymous EuroVis reviewers for their valuable comments and suggestions to improve the quality of the paper.

References

[ABo16a] ABO S.: Australian statistical geography standard (asgs): volume 1—main structure and greater capital city statistical areas. *Canberra: Australian Bureau of Statistics* (2016). 9

[ABo16b] ABO S.: Census of population and housing: Nature and content, australia. *Australian Bureau of Statistics* (2016). 9

[ADM17] ASAEEDI S., DIDEHVAR F., MOHADES A.: α-concave hull, a generalization of convex hull. *Theoretical Computer Science 702* (2017), 48–59. 2, 6

[AJT01] ALANI H., JONES C. B., TUDHOPE D.: Voronoi-based region approximation for geographical informaton retrieval with gazetteers. *International Journal of Geographical Information Science 15*, 4 (2001), 287–306. 6

[Are12] AREAS G. A.: Gadm database of global administrative areas, 2012. 9

[AS94] AHLBERG C., SHNEIDERMAN B.: Visual information seeking: tight coupling of dynamic query filters with starfield displays. In *SIGCH* (1994), pp. 313–317. 2

[AvKR*06] ARAMPATZIS A., VAN KREVELD M., REINBACHER I., JONES C. B., VAID S., CLOUGH P., JOHO H., SANDERSON M.: Web-based delineation of imprecise regions. *Computers, Environment and Urban Systems 30*, 4 (2006), 436–459. 6

[BKSB95] BRINKHOFF T., KRIEGEL H.-P., SCHNEIDER R., BRAUN A.: Measuring the complexity of polygonal objects. In *International Workshop on Advances in Geographic Information Systems* (1995), pp. 109–117. 7

[BMPH97] BREWER C. A., MACEACHREN A. M., PICKLE L. W., HERRMANN D.: Mapping mortality: evaluating color schemes for choropleth maps. *Annals of the Association of American Geographers 87*, 3 (1997), 411–438. 2, 10

[Boj09] BOJKO A. A.: Informative or misleading? heatmaps deconstructed. In *HCI International* (2009), Springer, pp. 30–39. 1

[CLNL87] CARR D. B., LITTLEFIELD R. J., NICHOLSON W., LITTLEFIELD J.: Scatterplot matrix techniques for large n. *Journal of the American Statistical Association 82*, 398 (1987), 424–436. 2

[CML11] CHO E., MYERS S. A., LESKOVEC J.: Friendship and mobility: user movement in location-based social networks. In *SIGKDD* (2011), pp. 1082–1090. 9

[CMS13] CAMPELLO R. J., MOULAVI D., SANDER J.: Density-based clustering based on hierarchical density estimates. In *PAKDD* (2013), Springer, pp. 160–172. 8

[CRM91] CARD S. K., ROBERTSON G. G., MACKINLAY J. D.: The information visualizer, an information workspace. In *SIGCHI* (1991), pp. 181–186. 10

[CXGH08] CHAN S.-M., XIAO L., GERTH J., HANRAHAN P.: Maintaining interactivity while exploring massive time series. In *VAST* (2008), pp. 59–66. 3

[DKWG08] DUCKHAM M., KULIK L., WORBOYS M., GALTON A.: Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition 41*, 10 (2008), 3224–3236. 2, 6, 7

[DP12] DROSOU M., PITOURA E.: Disc diversity: result diversification based on dissimilarity and coverage. *Proceedings of the VLDB Endowment 6*, 1 (2012), 13–24. 1, 2

[EBN15] EBERT T., BELZ J., NELLES O.: Interpolation and extrapolation: Comparison of definitions and survey of algorithms for convex and concave hulls. In *IEEE Symposium on Computational Intelligence and Data Mining* (2015), pp. 310–314. 5

[EKSX96] ESTER M., KRIEGEL H.-P., SANDER J., XU X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD* (1996), pp. 226–231. 4, 8

[Fis11] FISHER D.: Incremental, approximate database queries and uncertainty for exploratory visualization. In *IEEE Symposium on Large Data Analysis and Visualization* (2011), pp. 73–80. 3

[GCB*97] GRAY J., CHAUDHURI S., BOSWORTH A., LAYMAN A., REICHART D., VENKATRAO M., PELLOW F., PIRAHESH H.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery 1*, 1 (1997), 29–53. 2

[GCZ*17] GALAKATOS A., CROTTY A., ZGRAGGEN E., BINNIG C., KRASKA T.: Revisiting reuse for approximate query processing. *Proceedings of the VLDB Endowment 10*, 10 (2017), 1142–1153. 3

[GD06] GALTON A., DUCKHAM M.: What is the region occupied by a set of points? In *GIScience* (2006), pp. 81–98. 5, 6, 7

[Jar73] JARVIS R. A.: On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters 2*, 1 (1973), 18–21. 6

[JJHM14] JUGEL U., JERZAK Z., HACKENBROICH G., MARKL V.: M4: a visualization-oriented time series data aggregation. In *Proceedings of the VLDB Endowment* (2014), pp. 797–808. 2

[JJHM16] JUGEL U., JERZAK Z., HACKENBROICH G., MARKL V.: Vdda: automatic visualization-driven data aggregation in relational databases. *The VLDB Journal 25*, 1 (2016), 53–77. 2

[JS98] JERDING D. F., STASKO J. T.: The information mural: A technique for displaying and navigating large information spaces. *TVCG 4*, 3 (1998), 257–271. 3, 4

[Kei96] KEIM D. A.: Pixel-oriented visualization techniques for exploring very large data bases. *Journal of Computational and Graphical Statistics 5*, 1 (1996), 58–77. 2, 3

[Kei02] KEIM D. A.: Information visualization and visual data mining. *TVCG 8*, 1 (2002), 1–8. 2

[LBCS18] LI M., BAO Z., CHOUDHURY F., SELLIS T.: Supporting large-scale geographical visualization in a multi-granularity way. In *WSDM* (2018), pp. 767–770. 9

[LBS*18] LI M., BAO Z., SELLIS T., YAN S., ZHANG R.: Homeseeker: A visual analytics system of real estate data. *Journal of Visual Languages & Computing 45* (2018), 1–16. 3

[LBSY16] LI M., BAO Z., SELLIS T., YAN S.: Visualization-aided exploration of the real estate data. In *Australian Database Conference* (2016), pp. 435–439. 9

[LCB*18] LI M., CHOUDHURY F., BAO Z., SAMET H., SELLIS T.: ConcaveCubes online demo system. http://115.146.89.158/ConcaveCubes/, 2018. 10

[LGH*17] LU Y., GARCIA R., HANSEN B., GLEICHER M., MACIEJEWSKI R.: The state-of-the-art in predictive visual analytics. 539–562. 3

[LJH13] LIU Z., JIANG B., HEER J.: imMens: real-time visual querying of big data. *Computer Graphics Forum 32*, 3 (2013), 421–430. 1, 2, 4, 5, 9

[LKS13] LINS L., KLOSOWSKI J., SCHEIDEGGER C.: Nanocubes for real-time exploration of spatiotemporal datasets. *TVCG 19*, 12 (2013), 2456–2465. 1, 2, 4, 5, 9

[MI95] MURATA T., ISHIBUCHI H.: MOGA: multi-objective genetic algorithms. In *IEEE International Conference on Evolutionary Computation* (1995), pp. 289–294. 4

[MLKS18] MIRANDA F., LINS L., KLOSOWSKI J., SILVA C.: Topkube: a rank-aware data cube for real-time exploration of spatiotemporal data. *TVCG 24*, 3 (2018), 1394–1407. 2

[MS07] MOREIRA A., SANTOS M. Y.: Concave hull: a k-nearest neighbours approach for the computation of the region occupied by a set of points. In *International Conference on Computer Graphics Theory and Applications* (2007), pp. 61–68. 2, 4, 6

[Mur83] MURTAGH F.: A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal 26*, 4 (1983), 354–359. 8

[NJS11] NUTANONG S., JACOX E. H., SAMET H.: An incremental Hausdorff distance calculation algorithm. *PVLDB 4*, 8 (August 2011), 506–517. 1

[PO12] PARK J.-S., OH S.-J.: A new concave lull algorithm and concaveness measure for n-dimensional datasets. *Jounral of Information Science and Engineering 28* (2012), 587–600. 6

[PSSC17] PAHINS C. A., STEPHENS S. A., SCHEIDEGGER C., COMBA J. L.: Hashedcubes: simple, low memory, real-time visual exploration of big data. *TVCG 23*, 1 (2017), 671–680. 1, 2, 4, 5, 9, 10, 11

[PWR04] PENG W., WARD M. O., RUNDENSTEINER E. A.: Clutter reduction in multi-dimensional data visualization using dimension reordering. In *InfoVis* (2004), pp. 89–96. 3

[Sam06] SAMET H.: *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006. 2

[Sei95] SEIDEL R.: The upper bound theorem for polytopes: an easy proof of its asymptotic version. *Computational Geometry 5*, 2 (1995), 115–116. 7

[Shn96] SHNEIDERMAN B.: The eyes have it: a task by data type taxonomy for information visualizations. In *IEEE Symposium on Visual Languages* (1996), pp. 336–343. 3

[SPG14] STOLPER C. D., PERER A., GOTZ D.: Progressive visual analytics: user-driven visual exploration of in-progress analytics. *TCVG 20*, 12 (2014), 1653–1662. 3

[ST86] SAMET H., TAMMINEN M.: An improved approach to connected component labeling of images. In *CVPR* (Miami Beach, FL, June 1986), pp. 312–318. 8

[SW84] SAMET H., WEBBER R. E.: On encoding boundaries with quadtrees. *TPAMI 6*, 3 (May 1984), 365–369. 8

[TGC03] TRUTSCHL M., GRINSTEIN G., CVEK U.: Intelligently resolving point occlusion. In *InfoVis* (2003), pp. 131–136. 3

[TKBH17] TURKAY C., KAYA E., BALCISOY S., HAUSER H.: Designing progressive and interactive analytics processes for high-dimensional data analysis. *TVCG 23*, 1 (2017), 131–140. 3

[Tob70] TOBLER W. R.: A computer movie simulating urban growth in the detroit region. *Economic geography 46*, sup1 (1970), 234–240. 1

[WFW*17] WANG Z., FERREIRA N., WEI Y., BHASKAR A. S., SCHEIDEGGER C.: Gaussian cubes: real-time modeling for visual exploration of large multidimensional datasets. *TVCG 23* (2017), 681–690. 2