The World in a Nutshell: Concise Range Queries

Ke Yi, Xiang Lian, *Student Member, IEEE*, Feifei Li, and Lei Chen, *Member, IEEE*

Abstract—With the advance of wireless communication technology, it is quite common for people to view maps or get related services from the handheld devices, such as mobile phones and PDAs. Range queries, as one of the most commonly used tools, are often posed by the users to retrieve needful information from a spatial database. However, due to the limits of communication bandwidth and hardware power of handheld devices, displaying all the results of a range query on a handheld device is neither communication-efficient nor informative to the users. This is simply because that there are often too many results returned from a range query. In view of this problem, we present a novel idea that a concise representation of a specified size for the range query results, while incurring minimal information loss, shall be computed and returned to the user. Such a concise range query not only reduces communication costs, but also offers better usability to the users, providing an opportunity for interactive exploration. The usefulness of the concise range queries is confirmed by comparing it with other possible alternatives, such as sampling and clustering. Unfortunately, we prove that finding the optimal representation with minimum information loss is an NP-hard problem. Therefore, we propose several effective and nontrivial algorithms to find a good approximate result. Extensive experiments on real-world data have demonstrated the effectiveness and efficiency of the proposed techniques.

Index Terms—Spatial databases, range queries, algorithms.

1 Introduction

Spatial databases have witnessed an increasing number of applications recently, partially due to the fast advance in the fields of mobile computing and embedded systems and the spread of the Internet. For example, it is quite common these days that people want to figure out the driving or walking directions from their handheld devices (mobile phones or PDAs). However, facing the huge amount of spatial data collected by various devices, such as sensors and satellites, and limited bandwidth and/or computing power of handheld devices, how to deliver *light* but *usable* results to the clients is a very interesting, and of course, challenging task.

Our work has the same motivation as several recent works on finding good representatives for large query answers, for example, representative skyline points in [1]. Furthermore, such requirements are not specific to spatial databases. General query processing for large relational databases and OLAP data warehouses has posed similar challenges. For example, approximate, scalable query processing has been a focal point in the recent work [2], where the goal is to provide *light*, *usable* representations of the query results early in the query processing stage such that an *interactive* query process is possible. In fact, Jermaine et al. [2] argued to return concise representations of the final query results in every possible stage of a long-running query evaluation. However, the focus of [2] is on join queries in the relational database and the approximate representation is a random sample of the final query results. Soon we will see that the goal of this work is different and random sampling is not a good solution for our problem.

For our purpose, *light* refers to the fact that the representation of the query results must be small in size, and it is important for three reasons. First of all, the client-server bandwidth is often limited. This is especially true for mobile computing and embedded systems, which prevents the communication of query results with a large size. Moreover, it is equally the same for applications with PCs over the Internet. In these scenarios, the response time is a very critical factor for attracting users to choose the service of a product among different alternatives, e.g., Google Map versus Mapquest, since long response time may blemish the user experience. This is especially important when the query results have large scale. Second, clients' devices are often limited in both computational and memory resources. Large query results make it extremely difficult for clients to process, if not impossible. This is especially true for mobile computing and embedded systems. Third, when the query result size is large, it puts a computational and I/O burden on the server. The database indexing community has devoted a lot of effort in designing various efficient index structures to speed up query processing, but the result size imposes an inherent lower bound on the query processing cost. If we return a small representation of the whole query results, there is also the potential of reducing the processing cost on the server and getting around this lower bound. As we see, simply applying compression techniques only solves the first problem, but not the latter two.

Usability refers to the question of whether the user could derive meaningful knowledge from the query results. Note

- *K. Yi, X. Lian, and L. Chen are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China. E-mail: {yike, xlian, leichen}@cse.ust.hk.*
- *F. Li is with the Department of Computer Science, Florida State University, Tallahassee, FL 32306. E-mail: lifeifei@cs.fsu.edu.*

Manuscript received 14 Oct. 2008; revised 3 May 2009; accepted 16 Sept. 2009; published online 9 Feb. 2010.
Recommended for acceptance by M. Garofalakis.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2008-10-0544.
Digital Object Identifier no. 10.1109/TKDE.2010.35.

that more results do not necessarily imply better usability. On the contrary, too much information may do more harm than good, which is commonly known as the *information overload* problem. As a concrete example, suppose that a user issues a query to her GPS device to find restaurants in the downtown Boston area. Most readers having used a GPS device could quickly realize that the results returned in this case could be almost useless to the client for making a choice. The results (i.e., a large set of points) shown on the small screen of a handheld device may squeeze together and overlap. It is hard to differentiate them, let alone use this information! A properly sized representation of the results will actually improve usability. In addition, usability is often related to another component, namely, *query interactiveness*, that has become more and more important. Interactiveness refers to the capability of letting the user provide feedback to the server and refine the query results as he or she wishes. This is important as very often, the user would like to have a rough idea for a large region first, which provides valuable information to narrow down her query to specific regions. In the above example, it is much more meaningful to tell the user a few areas with high concentration of restaurants (possibly with additional attributes, such as Italian versus American restaurants), so that she could further refine her query range.

1.1 Problem Definition

Motivated by these observations, this work introduces the concept of *concise range queries*, where *concise* collectively represents the *light, usable, and interactive* requirements laid out above. Formally, we represent a point set using a collection of bounding boxes and their associated counts as a concise representation of the point set.

Definition 1. *Let P be a set of n points in $\mathbb{R}^2$. Let $\mathcal{P} = \{P_1, \dots, P_k\}$ be a partitioning of the points in P into k pairwise disjoint subsets. For each subset P_i, let R_i be the minimum axis-parallel bounding box of the points in P_i. Then, the collection of pairs $\mathcal{R} = \{(R_1, |P_1|), \dots, (R_k, |P_k|)\}$ is said to be a* concise representation *of* size k *for P, with $\mathcal{P}$ as its* underlying partitioning.

We will only return $\mathcal{R}$ as a concise representation of a point set to the user, while the underlying partitioning $\mathcal{P}$ is only used by the DBMS for computing such an $\mathcal{R}$ internally. Note that the definition above can be easily extended to a set of objects of arbitrary shapes, rather than just points. In particular, in Section 4, we will apply the same notion on a set of rectangles. But for now we will focus on point sets. Clearly, for fixed dimensions, the amount of bytes required to represent $\mathcal{R}$ is only determined by its size k (as each box R_i could be captured with its bottom left and top right corners).

There could be many possible *concise representations* for a given point set P and a given k. Different representations could differ dramatically in terms of quality, as with $\mathcal{R}$, all points in a P_i are replaced by just a bounding box R_i and a count $|P_i|$. Intuitively, the smaller the R_is are, the better. In addition, an R_i that contains a large number of points shall be more important than one containing few points. Thus, we use the following "information loss" as the quality measure of $\mathcal{R}$:

Definition 2. *For a concise representation $\mathcal{R} = \{(R_1, |P_1|), \dots, (R_k, |P_k|)\}$ of a point set P, its* information loss *is:*

$$L(\mathcal{R}) = \sum_{i=1}^{k} (R_i.\delta_x + R_i.\delta_y)|P_i|, \tag{1}$$

where $R_i.\delta_x$ and $R_i.\delta_y$ denote the x-span and y-span of R_i, respectively, and we term $R_i.\delta_x + R_i.\delta_y$ as the extent *of R_i.*

The rationale behind the above quality measure is the following. In the concise representation $\mathcal{R}$ of P, we only know that a point p is inside R_i for all $p \in P_i$. Therefore, the information loss as defined in (1) is the amount of "uncertainty" in both the x-coordinate and the y-coordinate of p, summed over all points p in P.

A very relevant problem is the k-anonymity problem from the privacy preservation domain, which observed the problem from a completely different angle. In fact, both k-anonymity and the concise representation could be viewed as clustering problems with the same objective function (1). After obtaining the partitioning $\mathcal{P}$, both of them replace all points in each subset P_i with its bounding box R_i. However, the key difference is that k-anonymity requires each cluster to contain at least k points (in order to preserve privacy) but no constraint on the number of clusters, whereas, in our case, the number of clusters is k while there is no constraint on cluster size. Extensive research on the k-anonymity [3], [4], [5] has demonstrated the effectiveness of using (1) as a measure of the amount of information loss by converting the point set P into $\mathcal{R}$.

Now, with Definitions 1 and 2, we define *concise range queries*.

Definition 3. *Given a large point set P in $\mathbb{R}^2$, a* concise range query *Q with* budget *k asks for a concise representation $\mathcal{R}$ of size k with the minimum information loss for the point set $P \cap Q$.*

Therefore, the user can specify a k according to the bandwidth and/or computing power of her mobile device, and retrieve the query results in a concise format meeting her constraints. On the other hand, we can also fix the information loss L and seek for a concise representation with a minimum k (the size of the output), which is a complement problem of Definition 3. Formally, we have:

Definition 4. *Given a large point set P in $\mathbb{R}^2$, a* complement concise range query *Q with* budget *L asks for a concise representation $\mathcal{R}$ of a minimum size, i.e., minimum k, for the point set $P \cap Q$ with information loss $L(\mathcal{R}) \le L$.*

We will focus on Definition 3 and discuss in Section 5 on how to extend the proposed solutions to complement concise range queries.

1.2 Summary of Contributions

Therefore, the goal of a concise range query is to find a concise representation, with the user-specified size, for all the points inside the query range. Ideally, one would like to have a concise representation of minimum information loss. We first give a dynamic programming algorithm that finds the optimal solution in one dimension in Section 3.1. Unfortunately, this optimization problem in two or more

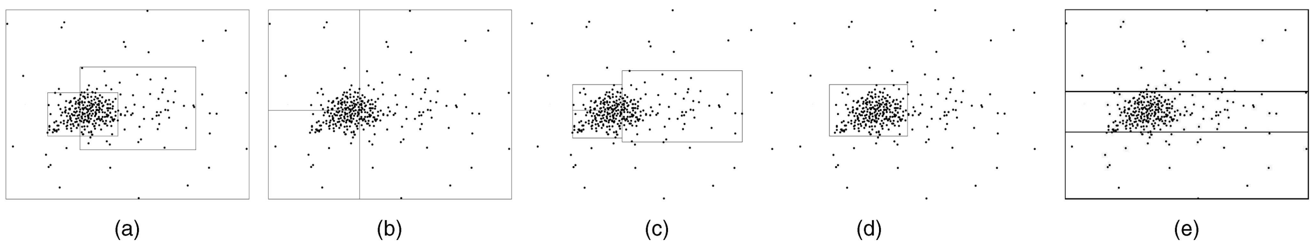


Fig. 1. Different alternatives for defining the concise representation, $k = 3$. (a) Our result. (b) k-means. (c) k-means without outliers. (d) Density based clustering. (e) $MinSkew$.

dimensions is NP-hard. In Section 3.2, we present a nontrivial reduction from PLANAR 3-SAT to the concise representation problem and prove its NP-hardness. Nevertheless, in our applications, the optimal solution is often unnecessary while efficiency is typically important. Thus, in Section 3.3, we focus on designing efficient yet effective algorithms that find good (but not optimal) concise representations.

The definition as stated requires us to first find the point sets $P \cap Q$ before trying to solve the optimization problem, i.e., we first need to process Q as if it were a standard "exact" range query. Since in our setting Q is typically large, and we are not aiming at the optimal solution due to the NP-hardness, this process is often expensive and unnecessary. Noticing the fact that we can evaluate $L(\mathcal{R})$ by only looking at $\mathcal{R}$ itself without knowing $P \cap Q$, we can actually carry out the optimization process without computing $P \cap Q$ in its entirety. Then, in Section 4, we explore how to speed up query processing by using an existing R-tree built on the data set P. We present an adaptive R-tree traversal algorithm that is much more efficient than answering the query exactly, and also produces high-quality concise representations of the query results.

We discuss some extensions of concise range queries in Section 5. In Section 6, we demonstrate the effectiveness and efficiency of the proposed techniques with extensive experiments on real data sets. A survey of related works appears in Section 7 before concluding the paper.

2 Limitation of Other Alternatives

2.1 Clustering Techniques

There is a natural connection between the concise range query problem and the many classic clustering problems, such as k-means, k-centers, and density-based clustering. In fact, our problem could be interpreted as a new clustering problem if we return the underlying partitioning $\mathcal{P}$ instead of the concise representation $\mathcal{R}$. Similarly, for existing clustering problems, one could return, instead of the actual clusters, only the "shapes" of the clusters and the numbers of points in the clusters. This will deliver a small representation of the data set as well. Unfortunately, as the primary goal of all the classic clustering problems is *classification*, the various clustering techniques do not constitute good solutions for our problem. In this section, we argue why this is the case and motivate the necessity of seeking new solutions tailored specifically for our problem.

Consider the example in Fig. 1, which shows a typical distribution of interesting points (such as restaurants) near a city found in a spatial database. There are a large number of points in a relatively small downtown area. The suburbs have a moderate density while the points are sparsely located in the countryside. For illustration purposes, we suppose the user has a budget $k = 3$ on the concise representation.

The concise representation following our definition will partition this data set into three boxes as in Fig. 1a (we omit the counts here). The downtown area is summarized with a small box with many points. The suburb is grouped by a larger box that overlaps with the first box (note that its associated count does not include those points contained in the first box) and all the outliers from the countryside are put into a very large box. One can verify that such a solution indeed minimizes the information loss (1). The intuition is that in order to minimize (1), we should partition the points in such a way that small boxes could have a lot of points while big boxes should contain as few as possible. If adding a new point to a cluster increases the size of its bounding box, then we need to exercise extra care, as it is going to increase the "cost" of all the existing points in the cluster. In other words, the cost of each point in a cluster C is determined by the "worst" points in C. It is this property that differentiates our problem with all other clustering problems, and actually makes our definition an ideal choice for obtaining a good concise representation of the point set.

The result of using the modified k-means approach is shown in Fig. 1b. Here, we also use the bounding box as the "shape" of the clusters. (Note that using the (center, radius) pair would be even worse.) Recall that the objective function of k-means is the sum of distance (or distance squared) of each point to its closest center. Thus, in this example, this function will be dominated by the downtown points, so all the three centers will be put in that area, and all the bounding boxes are large. This obviously is not a good representation of the point set: It is not too different from that of, say, a uniformly distributed data set.

One may argue that the result in Fig. 1b is due to the presence of outliers. Indeed, there has been a lot of work on outlier detection, and noise-robust clustering [6]. However, even if we assume that the outliers can be perfectly removed and hence the bounding boxes can be reduced, it still does not solve the problem of putting all three centers in the downtown (Fig. 1c). As a result, roughly 1/3 of the downtown points are mixed together with the suburban points. Another potential problem is, what if some of the outliers are important? Although it is not necessary to pinpoint their exact locations, the user might still want to know their existence and which region they are located in.

Our representation (Fig. 1a) with $k = 3$ only tells the existence of these outliers. But as we increase k, these outliers will eventually be partitioned into a few bounding boxes, providing the user with more and more information about them.

Last, Fig. 1d shows the result obtained by a density based clustering approach. A typical density based clustering, such as CLARANS [7], discovers the clusters by specifying a clustering distance ϵ. After randomly selecting a starting point for a cluster, the cluster starts to grow by inserting neighbors whose distance to some current point in the cluster is less than ϵ. This process stops when the cluster cannot grow any more. This technique, when applied to our setting, has two major problems. First, we may not find enough clusters for a given k (assume that there is a support threshold on the minimum number of points in one cluster). In this example, we will always have only one cluster. Second, the clusters are quite sensitive to the parameter ϵ. Specifically, if we set ϵ small, then we will obtain only the downtown cluster (Fig. 1d); if we set ϵ large, then we will obtain the cluster containing both the downtown and the suburb. Neither choice gives us a good representation of the point set.

Finally, we omit the result from k-centers. Since k-centers is trying to create clusters that minimize the maximum distance of any point to its cluster center, it is easy to see that this produces even worse results than k-means with respect to being good concise representations of the input data set. In addition, histogram [28], [29], [30] is also a popular tool for summarizing data in several buckets (partitions); however, the construction of histogram has a different goal from our problem. We will give a detailed discussion later in Section 7.

In summary, none of the clustering techniques work well for the concise range query problem since the primary goal of clustering is classification. An important consequence of this goal is that they will produce clusters that are disjoint. To the contrary, as shown in Fig. 1a, our goal is to build the partitioning $\mathcal{P}$ that minimizes the information loss. Hence, we need to look for new algorithms and techniques for the concise range query problem, which consciously build the partitioning $\mathcal{P}$ to minimize the information loss.

2.2 Histogram

Our work is also related to histogram construction [28], [29], [30]. Specifically, a histogram consists of several buckets, each of which stores the frequency of data points in it. The histogram has been widely used as a tool for selectivity estimation [28], [29]. Ioannidis and Poosala [28] studied the V-optimal histogram for 1D data, whereas Acharya et al. [29] investigated the histogram, namely $MinSkew$, in multidimensional space for spatial data. The construction of V-optimal/$MinSkew$ histogram aims to find partitions to minimize the error of selectivity estimation (or called spatial-skew), which is defined as summing up the multiplication of frequency and the statistical variance of the spatial densities of all points grouped within that bucket. Thus, the goal of V-optimal/$MinSkew$ histogram is different from ours (i.e., minimizing the information loss). Another major difference is that $MinSkew$ does not allow overlapping buckets, whereas we do. Fig. 1e illustrates an example of $MinSkew$, resulting in disjoint partitions of the data set. Note, however, that here the three obtained buckets only minimize the spatial-skew, rather than the information loss (which is defined as summing up the multiplication of frequency and the extent of each bucket).

2.3 Random Sampling

Random sampling is another tempting choice, but it is easy to see that it is inferior to our result in the sense that, in order to give the user a reasonable idea on the data set, a sufficient number of samples need to be drawn, especially for skewed data distributions. For example, using $k = 3$ bounding boxes roughly corresponds to taking six random samples. With a high concentration of points in the downtown area, it is very likely that all six samples are drawn from there.

Indeed, random sampling is a very general solution that can be applied on any type of queries. In fact, the seminal work of [2] proposed to use a random sample as an approximate representation of the results of a join, and designed nontrivial algorithms to compute such a random sample at the early stages of the query execution process. The fundamental difference between their work and ours is that the results returned by a range query in a spatial database are strongly correlated by the underlying geometry. For instance, if two points p and q are returned, then all the points in the database that lie inside the bounding box of p and q must also be returned. Such a property does not exist in the query results of a join. Thus, it is difficult to devise more effective approximate representations for the results of joins than random sampling. On the other hand, due to the nice geometric and distributional properties exhibited by the range query results, it is possible to design much more effective means to represent them concisely. Our work is exactly trying to exploit these nice spatial properties, and design more effective and efficient techniques tailored for range queries.

3 The Base Algorithms

In this section, we focus on the problem of finding a concise representation for a point set P with minimum information loss. First in Section 3.1, we show that in one dimension, a simple dynamic programming algorithm finds the optimal solution in polynomial time. However, this problem becomes NP-hard in two dimensions as we show in the Section 3.2. Then, we settle for efficient heuristic algorithms for the problem in Section 3.3 for two or higher dimensions.

3.1 Optimal Solution in One Dimension

We first give a dynamic programming algorithm for computing the optimal concise representation for a set of points P lying on a line. Later, in Section 3.3, we will extend it to higher dimensions, leading to an efficient heuristic.

Let $p_1, \dots, p_n$ be the points of P in sorted order. Let $\mathcal{P}_{i,j}$ represent the optimal partitioning underlying the best concise representation, i.e., with the minimum information loss, for the first i points of size j, $i \geq j$. The optimal solution is simply the concise representation for $\mathcal{P}_{n,k}$, and $\mathcal{P}_{n,k}$ could be found using a dynamic programming approach. The key observation is that in one dimension, the optimal partitioning always contains segments that do

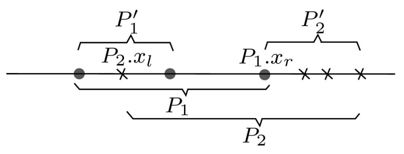


Fig. 2. Proof of Lemma 1.

not overlap, i.e., we should always create a group with consecutive points without any point from another group in-between. Formally, we have

Lemma 1. *$\mathcal{P}_{i,j}$ for $i \le n, j \le k$ and $i \ge j$ assigns $p_1, \ldots, p_i$ into j nonoverlapping groups and each group contains all consecutive points covered by its extent.*

Proof. We prove by contradiction. Suppose this is not the case and $\mathcal{P}_{i,j}$ contains two groups P_1 and P_2 that overlap in their extents as illustrated in Fig. 2. Let $P_i.x_l$ and $P_i.x_r$ denote the leftmost and rightmost points in P_i. Without loss of generality, we assume $P_1.x_l \le P_2.x_l$. Since P_1 intersects P_2, we have $P_2.x_l \le P_1.x_r$. If we simply exchange the membership of $P_1.x_r$ and $P_2.x_l$ to get P_1' and P_2', it is not hard to see that both groups' extents shrink and the numbers of points stay the same. This contradicts with the assumption that $\mathcal{P}_{i,j}$ is the optimal partitioning. □

Thus, $\mathcal{P}_{i,j}$ is the partitioning with the smallest information loss from the following $i-j+1$ choices: $(\mathcal{P}_{i-1,j-1}, \{p_i\})$, $(\mathcal{P}_{i-2,\,j-1},\ \{p_{i-1},\ p_i\}), \ldots, (\mathcal{P}_{j-1,\,j-1},\ \{p_j, \ldots, p_i\})\}$. Letting $L(\mathcal{P}_{i,j})$ be the information loss of $\mathcal{P}_{i,j}$, the following dynamic programming formulation becomes immediate:

$$L(\mathcal{P}_{i,j}) = \min_{1 \le \ell \le i-j+1} (L(\mathcal{P}_{i-\ell,j-1}) + \ell \cdot |p_i - p_{i-\ell+1}|), \qquad (2)$$

for $1 \le i \le n$, $2 \le j \le k$ and $j \le i$. The base case is $\mathcal{P}_{i,1} = \{\{p_1, \ldots, p_i\}\}$ for $1 \le i \le n$. Since computing each $L(\mathcal{P}_{i,j})$ takes $O(n)$ time, the total running time of this algorithm is $O(kn^2)$.

Theorem 1. *In one dimension, the concise representation with the minimum information loss for a set of points P can be found in $O(kn^2)$ time.*

We notice that there is a 1D approach given in [3], which is similar to the problem mentioned in this section with the only difference on the condition for the dynamic programming.

3.2 Hardness of the Problem in 2D

Not surprisingly, like many other clustering problems, for a point set P in $\mathbb{R}^2$, the problem of finding a concise representation of size k with minimum information loss is NP-hard. Also, similar to other clustering problems in euclidean space, the NP-hardness proof is quite involved. Below, we give a carefully designed construction that gives a reduction from PLANAR 3-SAT to the concise representation problem.

In the classical 3-SAT problem, there are n Boolean variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$, where each clause is a disjunction of three variables or their negations. The problem is to decide if there exists an assignment of variables such that all clauses evaluate to true. Assume that we map each variable and each clause

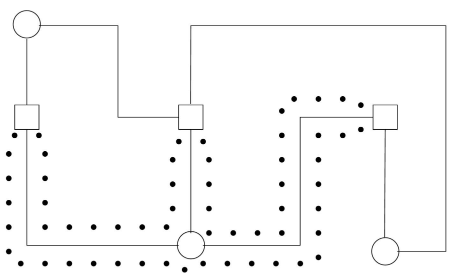


Fig. 3. The graph G of the PLANAR 3-SAT instance, where a circle represents a variable and a square represents a clause. The figure shows the chain of points surrounding the three edges incident to one of the variables.

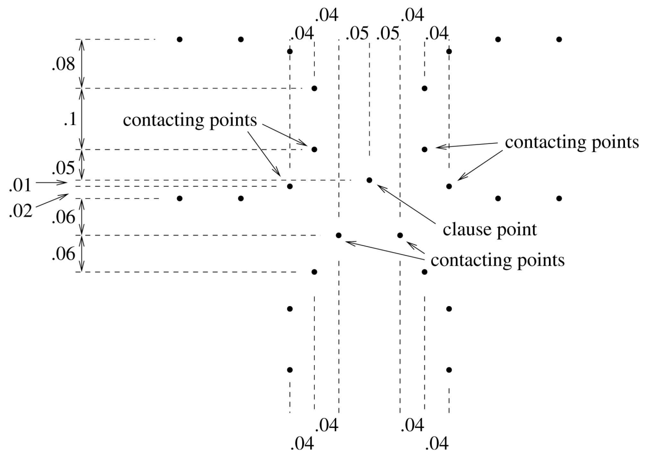


Fig. 4. Joining three chains at a clause.

to a vertex of an undirected graph G. Then, two nodes u and v in graph G are connected if and only if u represents a clause that contains the variable (or its negation) represented by v. If G is planar, the problem is known as PLANAR 3-SAT. It is an NP-complete problem [8]. The problem remains NP-complete even if each variable and its negation appear in at most three clauses. Note that in this case, any vertex in G has degree at most 3.

Given an instance of PLANAR 3-SAT, we construct a point set P as follows: First, since any vertex in G has degree at most 3, we can find a planar embedding of G such that all the vertices lie on grid points and all the edges follow nonintersecting grid paths. Such an embedding can be found in $O(n+m)$ time and the resulting embedding has area $O((n+m)^2)$ [9].

For each variable x_i, we build a chain of an even number of points $(p_{i,1}, \ldots, p_{i,2t_i})$ surrounding the three incident edges of the variable vertex (Fig. 3), such that the ℓ_1 distance between any two consecutive points in the chain (including between $p_{i,2t_i}$ and $p_{i,1}$) is 0.1. We also create a point in P at each clause vertex C_j, where we join the three chains corresponding to the three variables in the clause using the configuration in Fig. 4. Suppose x_i (or $\bar{x}_i$) appears in C_j, and consider the two nearest ℓ_1-neighbors in the chain of x_i. We refer to them as C_j's *contacting points* in the chain of x_i. Note that the ℓ_1 distances from C_j to them are both 0.14. If x_i appears in C_j, then we design the chain such that the two contacting points are $x_{i,2l-1}$ and $x_{i,2l}$ for some integer l; if $\bar{x}_i$ appears in C_j, then the two contacting points are $x_{i,2l}$ and $x_{i,2l+1}$ for some integer l. This is always doable by appropriately detouring the chain with additional points. This completes our construction of the point set P. It is obvious that it takes polynomial time. Finally, we set $k = \sum_i t_i$.

The basic intuition in our reduction is the following. We will partition each chain into t_i pairs of points; this gives us k subsets of P. There are two ways to partition the chain, namely $\{\{p_{i,1}, p_{i,2}\}, \{p_{i,3}, p_{i,4}\}, \ldots, \{p_{i,2t_i-1}, p_{i,2t_i}\}\}$ and $\{\{p_{i,2t_i}, p_{i,1}\}, \{p_{i,2}, p_{i,3}\}, \ldots, \{p_{i,2t_i-2}, p_{i,2t_i-1}\}\}$. Both of them have the same minimum information loss for the chain, and they will correspond to assigning "true" and "false" to the variable. Next, we add each clause point C_j into one of the pairs in one of its joining chains. By the way we choose the contacting points, if x_i appears in C_j and $x_i = true$, we will be able to add C_j into the pair of the two contacting points in x_i's chain, resulting in a bounding box with an extent of 0.19. Similarly, we can do the same if $\bar{x}_i$ appears in C_j and $x_i = false$. But if none of the variables satisfies the clause, C_j will have to be added to a pair consisting of a contacting point and a noncontacting point, which will result in a bounding box with an extent >0.19. Thus, we will be able to decide if the original PLANAR 3-SAT instance is satisfiable or not by looking at the information loss of the optimal concise representation of P. In the sequel, we formalize the idea by proving that the optimal partitioning of P will exactly behave in the way we have prescribed above.

First, one can verify that the constructed point set P has the following properties:

Property 1. *The ℓ_1 distance between any two consecutive points in any chain is 0.1, while the distance between any other pair of points is >0.1.*

Property 2. *The bounding box for a clause point and the two contacting points from one of its joining chains have extent 0.19. The bounding box for any other three points has extent >0.19.*

Property 3. *The extent of the bounding box is ≥0.24 for any four points, ≥0.32 for any five points, ≥0.36 for any six points, and ≥0.38 for any seven points.*

Let $\mathcal{R}_{opt}$ be the optimal concise representation for P of size k. We first prove the following lemmas:

Lemma 2. *$L(\mathcal{R}_{opt}) \geq 0.2k + 0.37m$, and the lower bound can be attained only if each clause point is grouped together with the two contacting points from one of its joining chains, while each of the remaining points in P is grouped together with one of its adjacent neighbors in its chain.*

Proof. Please refer to Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.35. □

Lemma 3. *The PLANAR 3-SAT instance has a satisfying assignment if and only if $L(\mathcal{R}_{opt}) = 0.2k + 0.37m$.*

Proof. Please refer to Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.35. □

The following hardness result is an immediate consequence of Lemma 3:

Theorem 2. *Given a point set $P \subset \mathbb{R}^2$ and an integer k, the problem of finding a concise representation $\mathcal{R}$ of size k for P with the minimum information loss is NP-hard.*

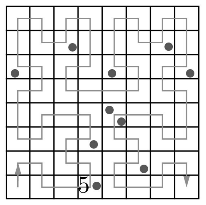


Fig. 5. The third-order Hilbert curve in two dimensions.

3.3 Heuristics for Two or More Dimensions

Given the hardness result, it is very unlikely that we can find an optimal concise representation $\mathcal{R}$ for a given point set P in polynomial time in two or more dimensions. Thus, in this section, we try to design efficient heuristic algorithms that produce an $\mathcal{R}$ with low information loss, although not minimum. Since our problem is also a clustering problem, it is tempting to use some popular clustering heuristic, such as the well-known k-means algorithm, for our problem as well. However, since the object function makes a big difference in different clustering problems, the heuristics designed for other clustering problems do not work for our case. The k-anonymity problem does share the same object function with us, but the clustering constraint there is that each cluster has at least k points, while we require that the number of clusters is k. These subtle but crucial differences call for new heuristics to be tailored just for the concise representation problem.

3.3.1 Algorithm HGroup

Given the optimal algorithm in one dimension, a straightforward idea is to use a function $\mathbb{R}^d \rightarrow \mathbb{R}$ to map the points of P from higher dimensions down to one dimension. Such an ordering function must somehow preserve the proximity relationships among the points. Many techniques exist for such a purpose (see [10] for a detailed review), among them the space-filling curves [11] have proved to be a popular choice. A space-filling curve traverses the space in a predetermined order. The most widely used space-filling curve is the Hilbert curve. The hth-order Hilbert curve has 2^{hd} cells in d dimensions and visits all the cells in the manner shown in Fig. 5. Each cell will be assigned a Hilbert value in sequence starting from 0, and all points falling inside the cell will get the same Hilbert value. For example, the point shown in Fig. 5 has a Hilbert value of 5.

Our Hilbert-curve based algorithm, called *HGroup*, is shown in Algorithm 1. The basic idea is to first compute the Hilbert value for each point in P, and sort all points by this value, mapping them to one dimension. Then, we simply group these points using our 1D dynamic programming algorithm from Section 3.1. More precisely, (2) becomes

$$L(\mathcal{P}_{i,j}) = \min_{1 \leq \ell \leq i-j+1} (L(\mathcal{P}_{i-\ell,j-1}) + \ell \cdot \text{extent}\ (\{p_{i-\ell+1}, \ldots, p_i\})),$$

where the *extent* is calculated using points' Hilbert values in one dimension. Thus, the running time of *HGroup* is still $O(kn^2)$, where n is the size of P. Of course, the optimality of the algorithm will be lost, and the quality of the result will depend on how well the Hilbert curve preserves the neighborhood information among points in the original, higher dimensional space [12]. It is not hard to imagine that the performance of *HGroup* will not be the best one can hope for, as important spatial information could be missed by the mapping of points from two dimensions into one dimension. Hence, it is natural to design better algorithms that work in two dimensions directly. On the other hand, *HGroup* is appealing for its simplicity and efficiency.

Algorithm 1. The algorithm *HGroup*

```
Compute the Hilbert value $h(p_i)$ for each point $p_i \in P$;
Sort $P$ by $h(p_i)$ and map it to one dimension;
Find the partitioning $\mathcal{P}$ using dynamic programming;
Build the concise representation $\mathcal{R}$ for $\mathcal{P}$ and return;
```

As a final note, the mapping from multidimensional data to 1D ones via Hilbert or iDistance [10] is also studied in [3]. Thus, we do not claim the *HGroup* approach is our major contribution.

3.3.2 Algorithm IGroup

Below we present another, more direct algorithm in two or more dimensions. It is an iterative algorithm that finds the k groups $P_1, \dots, P_k$, one at a time. We call this algorithm *IGroup*. In each iteration, we start with a *seed*, randomly chosen from the remaining points, and greedily add points into the group one by one. In the ith iteration, we first initialize the group P_i to include only the seed. Let U be the set of remaining points. Let $\rho(X)$ and $A(X)$ be the extent and area of the minimum bounding box of the set of points X, respectively. As we add points into P_i, we keep an *estimated total information loss* $\tilde{L}(P_i)$ for the current P_i, defined as

$$\tilde{L}(P_i) = \rho(P_i)|P_i| + 2\sqrt{A(U)/(k-i)} \cdot |U|. \quad (3)$$

Note that the first term in $\tilde{L}(P_i)$ is the information loss of P_i, while the second term is an estimate of the information loss on the remaining points U, assuming they are uniformly distributed in the bounding box of U.

When deciding which point $p \in U$ should be added to P_i, we first compute $\tilde{L}(P_i \cup \{p\})$ for all $p \in U$, and choose the one that minimizes $\tilde{L}(P_i \cup \{p\})$. If for all $p \in U$, $\tilde{L}(P_i \cup \{p\}) \geq \tilde{L}(P_i)$, then we stop adding points to P_i. The intuition is that, if we have $k-i$ bounding boxes left to group the points in U, then one partitioning that is always doable is to draw $k-i$ squares, each of dimensions (roughly) $\sqrt{A(U)/(k-i)} \times \sqrt{A(U)/(k-i)}$, to enclose all the points, which results in an information loss equal to the second term of $\tilde{L}(P_i)$. If $\tilde{L}(P_i)$ cannot be further reduced by adding more points, we should probably stop and start a new group.

When we stop adding points and obtain a P_i, we record that together with its $\tilde{L}(P_i)$ achieved in the end, and call it a candidate. To improve quality, we carry out the same process with a number of different seeds, and choose the best candidate that has attained the lowest $\tilde{L}(P_i)$. Then, we proceed to the next group P_{i+1}. Finally, for the last group P_k, we simply put all the remaining points into it. We give the details of the complete algorithm *IGroup* in Algorithm 2.

Algorithm 2: The algorithm *IGroup*

```
$U \leftarrow P$;
$s \leftarrow$ number of seeds to try;
for $i = 1, \dots, k-1$ do
    $\tilde{L}_{best} = \infty$;
    $U' \leftarrow U$;
    for $j = 1, \dots, s$ do
        $U \leftarrow U'$;
        $p_s \leftarrow$ randomly chosen seed from $U$;
        $P'_i \leftarrow \{p_s\}$;
        $U \leftarrow U - \{p_s\}$;
        while true do
            Let $p = \arg\min_p \tilde{L}(P'_i \cup \{p\})$;
            if $\tilde{L}(P'_i \cup \{p\}) < \tilde{L}(P'_i)$ then
                $P'_i \leftarrow P'_i \cup \{p\}$;
                $U \leftarrow U - \{p\}$;
            else break;
        if $\tilde{L}(P'_i) < \tilde{L}_{best}$ then
            $\tilde{L}_{best} \leftarrow \tilde{L}(P'_i)$;
            $P_i \leftarrow P'_i$;
    $U \leftarrow U - P_i$;
    output $P_i$;
```

There are $k-1$ iterations in the algorithm. In each iteration, we check each of the n points and choose the best one to add to the current group. In the worst case, we could check all the points $O(n)$ times. Each iteration needs to be repeated for s times with s randomly chosen seeds. So the worst case running time of *IGroup* is $O(skn^2)$.

4 Query Processing with R-Trees

In order to use the algorithms of Section 3.3 to answer a concise range query Q with budget k from the client, the database server would first need to evaluate the query as if it were a standard range query using some spatial index built on the point set P, typically an R-tree. After obtaining the complete query results $P \cap Q$, the server then partitions $P \cap Q$ into k groups and returns the concise representation. However, as the main motivation to obtain a concise answer is exactly because $P \cap Q$ is too large, finding the entire $P \cap Q$ and running the algorithms of Section 3.3 are often too expensive for the database server. In this section, we present algorithms that process the concise range query without computing $P \cap Q$ in its entirety. The idea is to first find k' bounding boxes, for some $k' > k$, that collectively contain all the points in $P \cap Q$ by using the existing spatial index structure on P. Each of these bounding boxes is also associated with the count of points inside. Then, we run a weighted version of the algorithm in Section 3.3, grouping these k' bounding boxes into k larger bounding boxes to form the concise representation $\mathcal{R}$. Typically $k' \ll |P \cap Q|$, so we could expect significant savings in terms of I/O and CPU costs as compared with answering the query exactly. Therefore, adopting concise range queries instead of the traditional exact range queries not only solves the bandwidth and usability problems, but also leads to substantial efficiency improvements.

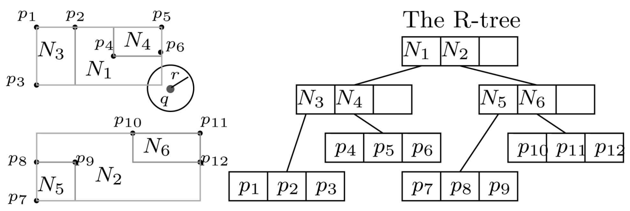


Fig. 6. The R-tree.

The algorithms presented in this section in general work with any space partitioning index structure; for concreteness, we will proceed with the R-tree, which is arguably the most widely used spatial index structure. The R-tree [13] and its variants (R*-tree in particular [14]) all have the following structure. Suppose the disk block size is B. We first group $\leq B$ points in proximity area into a minimum bounding rectangle (MBR); these points will be stored at a leaf on the R-tree. These MBRs are then further grouped together level by level until there is only one left. Each node u in the R-tree is associated with the MBR enclosing all the points stored below, denoted by $\text{MBR}\ (u)$. Each internal node also stores the MBRs of all its children. An example of the R-tree is illustrated in Fig. 6. Different R-tree variants only differ in the rules how the MBRs or points are grouped together.

The standard range query Q can be processed using an R-tree as follows: We start from the root of the R-tree, and check the MBR of each of its children. Then, we recursively visit any node u whose MBR intersects or falls inside Q. When we reach a leaf, we simply return all the points stored there that are inside Q.

In this section, we in addition assume that each node u in the R-tree also keeps n_u, the number of the points stored below its subtree. Such counts can be easily computed and maintained in the R-tree.

4.1 Basic Ideas

In the following, for brevity we will also say a point is the MBR of itself. The basic idea of our R-tree-based algorithms is to evaluate Q in a way similar to a standard range query, but stop early in the tree so that we find k' MBRs to feed to the algorithm in Section 3.3, for some $\alpha \cdot k \leq k' \ll |P \cap Q|$, where $\alpha > 1$ is some constant. The observation is that, since the R-tree always tries to group close objects together with MBRs, these MBRs should be a good representation of the query results. However, on the other hand, since the primary goal of the R-tree is fast query processing, the MBRs do not constitute a good concise representation of the query result. For example, most R-tree variants try to minimize the overlap between MBRs, but as we have seen in Section 2, overlapping among the R_is is often necessary and beneficial for reducing the information loss. So, we do not want to just use these MBRs as the R_is in the concise representation $\mathcal{R}$. Instead, we would like to find $k' \geq \alpha k$ MBRs so that there is a chance for our grouping algorithm of Section 3.3 to further optimize the concise representation.

4.2 Algorithm R-BFS

The straightforward way to find k' such MBRs is to visit the part of the R-tree inside Q in a BFS manner, until we reach a level where there are at least αk MBRs. We call this algorithm *R-BFS*. In particular, for any node u whose MBR is complete inside Q, we directly return MBR(u) together with n_u. For any node u whose MBR is partially inside Q, we return the intersection of the MBR(u) and Q, while the associated count is estimated as[1]

$$n_u \cdot \frac{Area(\text{MBR}\ (u) \cap Q)}{Area(Q)}, \tag{4}$$

assuming uniform distribution of the points in MBR(u).

4.3 Algorithm R-Adaptive

The above BFS traversal treats all nodes alike in the R-tree and will always stop at a single level. But, intuitively, we should go deeper into regions that are more "interesting," i.e., regions deserving more user attention. These regions should get more budget from the k bounding boxes to be returned to the user. Therefore, we would like a quantitative approach to measuring how "interesting" a node in the R-tree is, and a corresponding traversal algorithm that visits the R-tree adaptively.

In the algorithm *R-Adaptive*, we start from the root of the R-tree with an initial budget of $\kappa = \alpha k$, and traverse the tree top-down recursively. Suppose we are at a node u with budget κ, and u has b children $u_1, \ldots, u_b$ whose MBRs are either completely or partially inside Q. Let the counts associated with them be $n_1, \ldots, n_b$. Specifically, if MBR(u_i) is completely inside Q, we set $n_i = n_{u_i}$; if it is partially inside, we compute n_i proportionally as in (4).

If $\kappa \leq b$, then we call the base algorithms of Section 3.3 to group them into k larger MBRs, and return them. Otherwise, we allocate the budget κ into $\kappa_1, \ldots, \kappa_b$, and assign κ_i to u_i. For any $\kappa_i = 1$, we directly return MBR(u_i) with n_i. If $\kappa_i \geq 2$, we recursively visit u_i with budget κ_i.

It now remains to specify how we allocate the budget into $\kappa_1, \ldots, \kappa_b$. Intuitively, if n_i is larger, then κ_i should also be larger; on the other hand, if MBR(u_i) is larger, κ_i should be larger too, since if two MBRs contain the same number of points, then we should allocate more budget to the larger one in order to minimize the total information loss. So, the allocation of budget should depend on a delicate balance between both n_i and the MBR(u_i). Below we derive such an allocation policy assuming that all the points inside each MBR(u_i) are uniformly distributed and each MBR(u_i) is *fat*, i.e., having a bounded aspect ratio.

Let the area of MBR(u_i)$\cap Q$ be A_i. If MBR(u_i) is fat, then when the n_i points inside are grouped into κ_i bounding boxes, each bounding box has extent roughly $2\sqrt{A_i/\kappa_i}$ and contains n_i/κ_i points. So, the information loss for u_i is $2\sqrt{A_i/\kappa_i} \cdot n_i/\kappa_i \cdot \kappa_i = 2n_i\sqrt{A_i/\kappa_i}$. The total loss is thus

$$L = 2n_1\sqrt{A_1/\kappa_1} + \cdots + 2n_b\sqrt{A_b/\kappa_b}.$$

We now minimize L under the constraint $\kappa_1 + \cdots + \kappa_b = \kappa$ but allowing the κ_is to be real numbers.

1. Strictly speaking, when using the estimate (4), we will only return an approximate count for $|P_i|$ in the concise representation. But, such an approximation is usually tolerable.

By the Cauchy-Schwartz inequality, we have

$$\begin{aligned} L &= 2\frac{\left(\sum_i n_i\sqrt{A_i/\kappa_i}\right)\left(\sum_i \frac{\sqrt{\kappa_i}}{n_i^{1/3}A_i^{1/6}}\right)}{\sum_i \frac{\sqrt{\kappa_i}}{n_i^{1/3}A_i^{1/6}}} \\ &\geq 2\frac{\left(\sum_i n_i^{1/3}A_i^{1/6}\right)^2}{\sum_i \frac{\sqrt{\kappa_i}}{n_i^{1/3}A_i^{1/6}}}, \end{aligned} \tag{5}$$

while the denominator is

$$\begin{aligned} \sum_i \frac{\sqrt{\kappa_i}}{n_i^{1/3}A_i^{1/6}} &\leq \sqrt{\left(\sum_i \kappa_i\right)\left(\sum_i \frac{1}{n_i^{2/3}A_i^{1/3}}\right)} \\ &= \sqrt{\kappa\left(\sum_i \frac{1}{n_i^{2/3}A_i^{1/3}}\right)}. \end{aligned} \tag{6}$$

Both inequalities (5) and (6) become equality when $\kappa_i \propto n_i^{2/3}A_i^{1/3}$, for $i = 1, \dots, b$. Therefore, we set

$$\kappa_i = \left\lceil \frac{n_i^{2/3}A_i^{1/3}}{\sum_j n_j^{2/3}A_j^{1/3}}\kappa \right\rceil. \tag{7}$$

We use a ceiling in (7) since each κ_i needs to be an integer, and the R-tree algorithm is allowed to return more MBRs than the budget. The final size k in the concise representation will be enforced by the algorithm of Section 3.3. The procedure of the recursive call $visit(u, \kappa)$ is outlined in Algorithm 3. The process starts by calling $visit$(root of the R-tree, αk).

Algorithm 3: Recursive call $visit(u, \kappa)$

```
Let $u_1, \dots, u_b$ be $u$'s children whose MBRs are
inside or partially inside $Q$;
Let $n_i$ = number of points inside MBR($u_i$)$\cap Q$;
**if** $b \geq \kappa$ **then**
    output MBR($u_i$)$\cap Q$ with $n_i$ for all $i$;
    **return**;
Let $A_i$ = *Area*(MBR($u_i$)$\cap Q$);
Compute $\kappa_i$ as in (9) for all $i = 1, \dots, b$;
**for** $i = 1, \dots, b$ **do**
    **if** $\kappa_i = 1$ **then**
        output MBR($u_i$)$\cap Q$ with $n_i$;
    **else**
        $visit(u_i, \kappa_i)$;
```

4.4 The Weighted Versions of the Base Algorithms

Our R-tree-based algorithms generate a number of MBRs, associated with counts, and pass them to the base algorithms of Section 3.3. As described, those algorithms can only process a point set. But, it is not difficult to adapt both *HGroup* and *IGroup* to handle a set of MBRs associated with counts (weights).

First, we simply take the centroids of each MBR to form the points in P and each point has a weight that is equal to the count of the corresponding MBR. Then, for the dynamic programming algorithm (which is used in algorithm *HGroup*), we change (2) to

$$L(\mathcal{P}_{i,j}) = \min_{\ell}(L(\mathcal{P}_{i-\ell,j-1}) + \text{extent}\ (\{p_{i-\ell+1}, \dots, p_i\}) \cdot \sum_{r=i-\ell+1}^{i} w(p_r)),$$

where $w(p)$ denotes the weight of p.

For the algorithm *IGroup*, we simply update (3) as

$$\tilde{L}(P_i) = \rho(P_i) \cdot \sum_{p\in P_i} w(p) + 2\sqrt{A(U)/(k-i)} \cdot \sum_{p\in U} w(p).$$

4.5 Discussions on the Shape of Query Region

Up to now, we always assume that the query region Q is an axis-parallel rectangle. In the case where the query region Q has other shapes (e.g., rotated rectangle, circle, or polygon), we can still apply our proposed approaches, *R-BFS* and *R-Adaptive*, with minor modifications. Specifically, for *R-BFS* approach, the estimated count associated with node (partially intersecting with Q) in (4) has to consider the intersecting area between MBR(u) and Q of any shape, which can be computed either by geometric position between the two, or by other estimation techniques such as histograms or sampling. Similarly, for *R-Adaptive*, we also need to consider the intersection between MBR(u) and Q of any shape. We can obtain the area of the resulting intersection region via techniques mentioned above, and estimate the budget assigned to the MBR node (as given by (7)).

4.6 Discussions on Other Spatial Data Types

We now discuss how to extend our solutions of answering concise range queries on data points to that on other spatial data types such as lines or rectangles. In particular, although our proposed approaches are originally designed for answering concise queries over data points, they are still applicable to spatial data with other shapes. In fact, in the literature of spatial databases [14], spatial objects are often simplified by rectangles (which bound spatial objects with arbitrary shapes). Our extension of concise range queries on lines or rectangles can be as follows: First, we bound lines/rectangles with minimum bounding rectangles (MBRs). Then, we index the resulting MBRs in an R-tree structure by using the standard "insert" operator. For any specified concise range Q, we can conduct the concise range query on such MBRs via R-tree in the same way as that over data points.

5 Extensions

5.1 Supporting Attributes

As we emphasized in Section 1, it is often useful if we can associate multiple counts with each bounding box, each of which represents the number of points with a particular attribute value, for example, Italian, Chinese, and American restaurants. Such counts can be easily supported. We can augment each node of the R-tree with the number of objects that are stored below for each attribute value. Then, after we have computed the concise representation $\mathcal{R}$, we can easily also compute the number of points in each P_i for each attribute value.

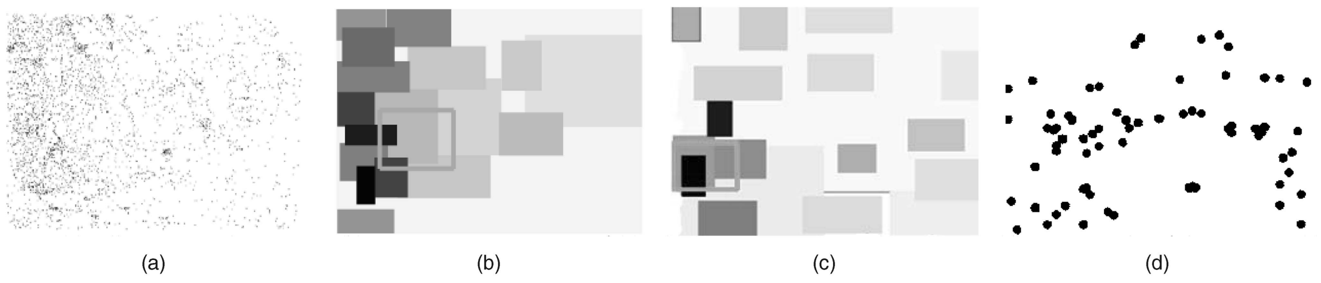


Fig. 7. Visualization of results. (a) All query results. (b) Concise representation of (a). (c) Concise representation of the query in (b). (d) Exact results of the query in (c).

5.2 Complement Concise Range Queries

As we mentioned in Section 1, the user may also be interested in asking a complement concise range query, i.e., she specifies a maximum allowed information loss L and then the goal is to compute a concise representation of the query results with minimum size that has an information loss no more than L.

Since by definition, the larger k is, the smaller the information loss is. Thus, we can conduct a binary search on k to find the minimum k that satisfies the user's requirement. For each k, we apply our algorithm in Sections 3 and 4 to find a concise representation $\mathcal{R}$ and compare with L. Thus, after a logarithmic number of steps, we can find a good $\mathcal{R}$ that meets the user's requirement. Note that, however, since our heuristic solutions cannot guarantee to find the minimum information loss for a given k, this binary search cannot guarantee to find the minimum k for the complement concise query problem either. This is not surprising, since the complement concise range query problem can be also shown to be NP-hard following our proof for the primary problem.

5.3 Progressive Refinement

Another useful yet simple extension of our algorithms is to support progressive refinement of the concise representation. In this case, the user with a slow network connection does not specify a k beforehand. Instead, the database server produces a series of concise representations of the query results with geometrically increasing sizes, e.g., $k = 10, 20, 40, 80, \ldots$. As time goes on, better and better representations are streamed to the user.

6 Experiments

6.1 Experimental Setup

We have implemented our two base algorithms, *HGroup* and *IGroup*, as well as the two R-tree traversal algorithms *R-BFS* and *R-Adaptive*. Specifically, we used the R*-tree [14] to index all the points in the data set. The *IGroup* algorithms first traverse the R-tree to produce a number of MBRs and feed them into the base algorithms, so we have, in total, two combinations: *R-BFS* + *IGroup*, *R-BFS* + *HGroup*, *R-Adaptive* + *IGroup*, and *R-Adaptive* + *HGroup*. The straw-men we compare against are the k-means clustering algorithm and the *MinSkew* histogram [29]. In particular, for k-means and *MinSkew*, we first obtain all the data points in the query region via R-tree, and then conduct either of the two methods to obtain clusters/partitions. All the algorithms were implemented in C++, and the experiments were conducted on a machine with a 4 GHz CPU and 1 G main memory. The page size of the R-tree is set to 4 KB.

6.2 Data Sets

We tested the query performance of our approaches over three real data sets, road networks from North America (NA), California (CA), and City of San Joaquin County (TG), and one synthetic data set, *Skew*. Specifically, for real data sets, NA and CA are from *digital chart of the world server*, whereas TG is from [31]. CA also contains a large number of points of interest (e.g., restaurants and resorts). These data sets are available online.[2] Points from the three real data sets have been normalized to lie within the unit square. The sizes for CA, NA, and TG are 125, 818, 175, 813, and 18,263, respectively. For the synthetic data set, *Skew*, we generate 100,000 data points within the unit square, where the coordinate on each dimension follows *Zipf* distribution (with skewness 0.8).

Note that, we evaluate the query performance on 2D real/synthetic point data sets mentioned above, since these spatial data follow typical data distributions in spatial databases. Moreover, the concise range query on these 2D spatial data has practical applications such as electronic maps like Google map or Mapquest, as mentioned in Section 1, where concise range answers have to be displayed on the small screens of handheld devices (e.g., PDA or mobile phone). Thus, we focus on evaluating 2D data sets in this section (though our proposed approaches can still hold for data sets with dimensionality greater than 2, as mentioned in Section 3.3).

6.3 Visualization of Results and Interactive Exploration

We first did some test queries to see if the concise representation indeed gives the user some intuitive high-level ideas about the query results. Fig. 7a shows all the points that are returned by a range query of size 0.1×0.1 on the CA data set. There are a total of 4,147 points in the full result, which corresponds to roughly 33 Kbytes of data (assuming each coordinate and each count take 4 bytes). Fig. 7b shows the concise representation of the results produced by our *R-Adaptive* + *IGroup* algorithm, where we used $k = 20$ and $\alpha = 8$. When plotting Fig. 7b, the gray scale is set to be proportional to the density of each bounding box (count/area). From the visualization results, we can see that even with k as small as 20, which corresponds to merely 400 bytes (suppose the user's bandwidth budget is 400 bytes), the concise result is already a pretty close approximation of the exact result. Compared with the exact results, the concise representation has an 80-fold reduction in size.

Starting from here, the user can interactively narrow her query down to areas of her interests, such as high-density

2. http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm.

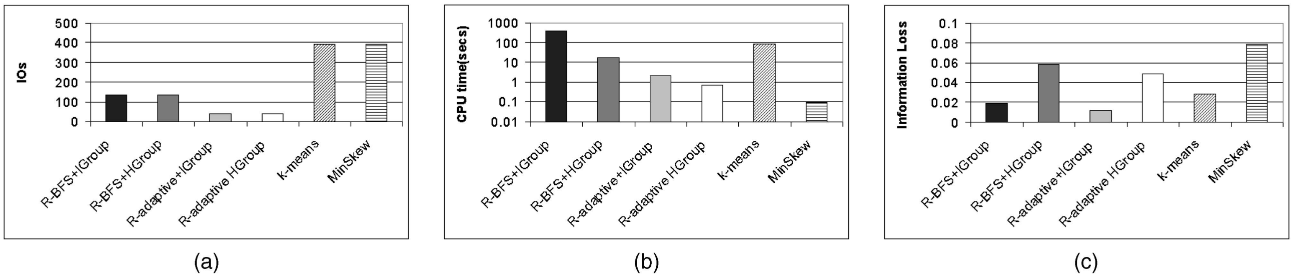


Fig. 8. Experimental results with different approaches on the CA data set. (a) I/O cost. (b) CPU time. (c) Information loss.

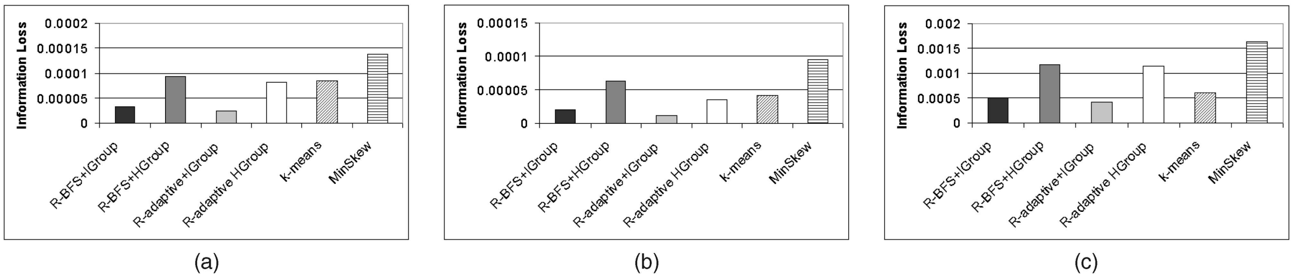


Fig. 9. Information loss with different approaches on (a) NA, (b) TG, and (c) *Skew* data sets.

areas or areas with medium density but closer to locations of interest, until the query range contains a small number of results, at which point all the exact results are returned. For example, Fig. 7c shows the concise representation of the zoomed-in query, the green box, on Fig. 7b, while Fig. 7d shows the exact results of the zoomed-in query, the green box, on Fig. 7c. Since the exact query results in Fig. 7d can be represented with less than 400 bytes, all the points inside the query are returned exactly.

Note that, the resulting concise representations can be overlapping with each other. However, this is usually not a problem for the user perception, as long as we render the higher density rectangles in front of the lower density ones, such as in Fig. 7.

6.4 Experimental Results with Different Approaches

We now compare six approaches $R\text{-}BFS + IGroup$, $R\text{-}Adaptive + IGroup$, $R\text{-}BFS + HGroup$, $R\text{-}Adaptive + HGroup$, k-means, and $MinSkew$ [29]. Fig. 8 illustrates the query performance of the six approaches on CA data set, in terms of the number of I/O cost (number of page accesses in the R-tree traversal), CPU time, and the average information loss per point in the concise representations returned. Note that, here the CPU time does not include the I/O cost. We fixed the query size at 0.1×0.1, $\alpha = 8$, and $k = 60$, and set the number of seeds in *IGroup* as $s = 10$.

From Fig. 8a, the I/O costs of $R\text{-}Adaptive + IGroup$ and $R\text{-}Adaptive + HGroup$ are the lowest among six approaches, followed by $R\text{-}BFS + IGroup$ and $R\text{-}BFS + HGroup$, and k-means and $MinSkew$ are the highest. This is because k-means and $MinSkew$ have to retrieve all the data points (falling into the query region) in the leaf nodes, which incurs the largest number of page accesses.

In Fig. 8b, the CPU time of $MinSkew$ is the lowest (about 0.1 second), due to the low cost of constructing the histogram. However, we can see from Fig. 8c that $MinSkew$ has the highest (worst) information loss among all approaches. The k-means algorithm incurs high CPU time (about 84 seconds) due to the data retrieval from the R-tree index and the clustering. Thus, k-means is not efficient in terms of CPU time (also true for the I/O cost), though the information loss in Fig. 8c is larger than that of *R-adaptive IGroup* and *R-BFS IGroup*. From the experimental results, the HGroup-based approaches have good response time compared with the IGroup-based ones, however, with a bit high information loss. Thus, there is a trade-off between the query efficiency and information loss here. In practice, if small response time is strictly required by applications, we can use the HGroup-based approach to obtain fast answers; otherwise, the IGroup-based one can be used.

Figs. 9a, 9b, and 9c show the average information loss per point on NA, TG, and *Skew* data sets, respectively, which has a similar trend to the CA data set in Fig. 8c. The only difference is that, for NA and TG data sets, k-means method has even worse information loss than $R\text{-}Adaptive + HGroup$. The I/O cost and CPU time on these three data sets are similar to CA, and thus omitted due to space limit. From the discussions above, we can see that k-means incurs low query efficiency, and is worse than $IGroup$ in terms of information loss; $MinSkew$ returns answers fast, however, with high information loss (since $MinSkew$ was not designed to minimize the information loss). Thus, they are not suitable for efficiently answering concise queries in applications like Google map on handheld devices. In the rest of this section, due to space limit, we only report the results on one real data set, CA, and one synthetic data set, *Skew*. The trends for the other two real data sets, NA and TG, are similar and thus omitted.

6.5 Experimental Results with Varying k

Next, we started to look into the performance of these six algorithms. In the first set of experiments, we fixed the query size at 0.1×0.1, $\alpha = 8$, and varied k from 20 to 100. Figs. 10a, 10b, and 10c present experimental results on

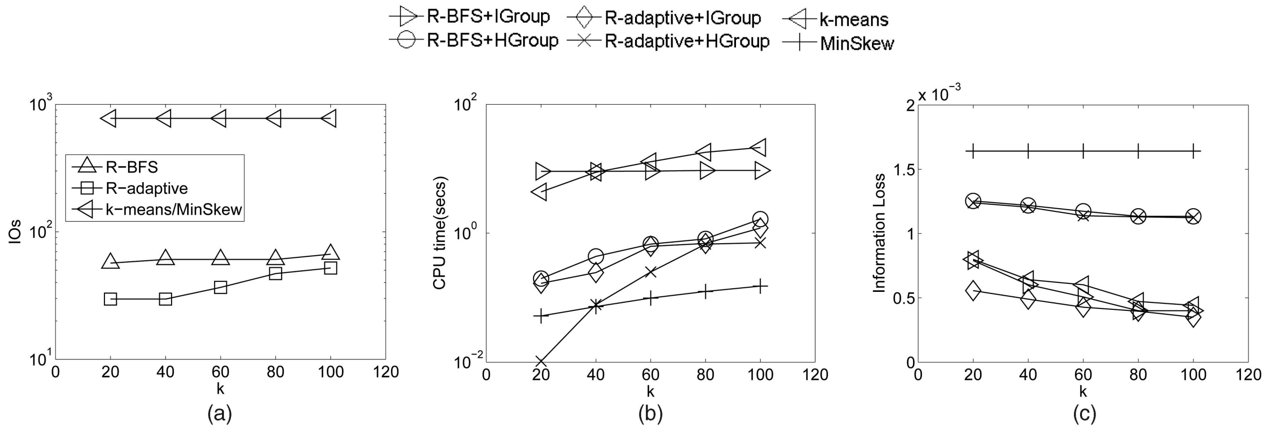


Fig. 10. Experimental results with varying k on the $Skew$ data set. (a) I/O cost. (b) CPU time. (c) Information loss.

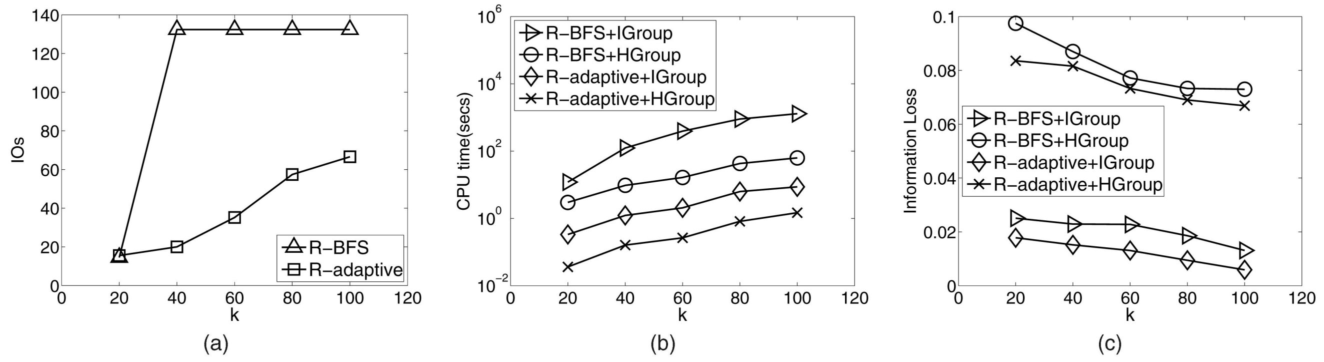


Fig. 11. Experimental results with varying k on the CA data set. (a) I/O cost. (b) CPU time. (c) Information loss.

synthetic $Skew$ data set, comparing the six approaches, whereas Fig. 11 shows the same set of experiments on the real CA data set. Similar to previous results, for the $Skew$ data set, the k-means and $MinSkew$ incur higher I/O cost than the other four algorithms in Fig. 10a. Moreover, as shown in Fig. 10b, the CPU time of k-means is greater than that of *R-adaptive IGroup*, *R-adaptive HGroup*, and *R-BFS HGroup*. Although the CPU time of $MinSkew$ is comparable to the *R-adaptive HGroup*, from Fig. 10c, $MinSkew$ has the worst information loss among all the six approaches. Note that, the information loss of $MinSkew$ in Fig. 10c decreases slowly for larger k values, compared with our approaches. This is because the $MinSkew$ is designed for minimizing spatial-skew (rather than information loss). Thus, the $MinSkew$ algorithm keeps on dividing buckets with the maximum decreases of spatial-skew (instead of maximum decreases of information loss). As a result, some large buckets with small variances (close to uniform), however, with high information losses, would not be divided by $MinSkew$ (which is exactly the goal of the histogram construction); on the other hand, those buckets after splitting can achieve lower information loss, however, they only contribute a small portion of the total information loss. As a consequence, when the number of buckets (i.e., k) becomes larger, the decreasing trend of information loss in $MinSkew$ is small, compared with other approaches, as confirmed in Fig. 10c.

In the subsequent experiments, for the sake of clearly illustrating the trend of our approaches, we will only report the results with our four algorithms $R\text{-}BFS + IGroup$, $R\text{-}Adaptive + IGroup$, $R\text{-}BFS + HGroup$, and $R\text{-}Adaptive + HGroup$. We next investigate the trend of our approaches for different k values on CA data set in Fig. 11. Specifically, Fig. 11a shows the I/O cost of the four algorithms. Since the base algorithms do not incur any I/O, we only show the I/O costs of the two R-tree algorithms. We can see that the number of page accesses of *R-BFS* increases quickly as k gets larger. This is because it visits the R-tree indistinguishably. Once a certain level of the R-tree has less than αk MBRs in the query range, it visits all the nodes one level down. Thus, the number of nodes accessed increases very quickly due to the large fan-out of the R-tree.

Fig. 11b shows the CPU time of the four algorithms. First, we observe that the two variants using *R-BFS* run much slower than the two *R-Adaptive* variants. This is expected, since the number of MBRs returned by *R-BFS* is much larger, and the running time is dominated by the grouping algorithms, which have complexities $O(kn^2)$ for *HGroup* and $O(skn^2)$ for *IGroup*, respectively, where n is the number of MBRs to be grouped. Comparing the two grouping algorithms, we observe that *HGroup* is faster than *IGroup*, which generally agrees with the analysis.

Fig. 11c shows the average information loss per point in the concise representations returned, where we observe the following. First, the two *IGroup*-based variants produce much better results than the two *HGroup*-based variants. The explanation is possibly that, since the Hilbert curve imposes a linear order on the 2D data set, some important geometric

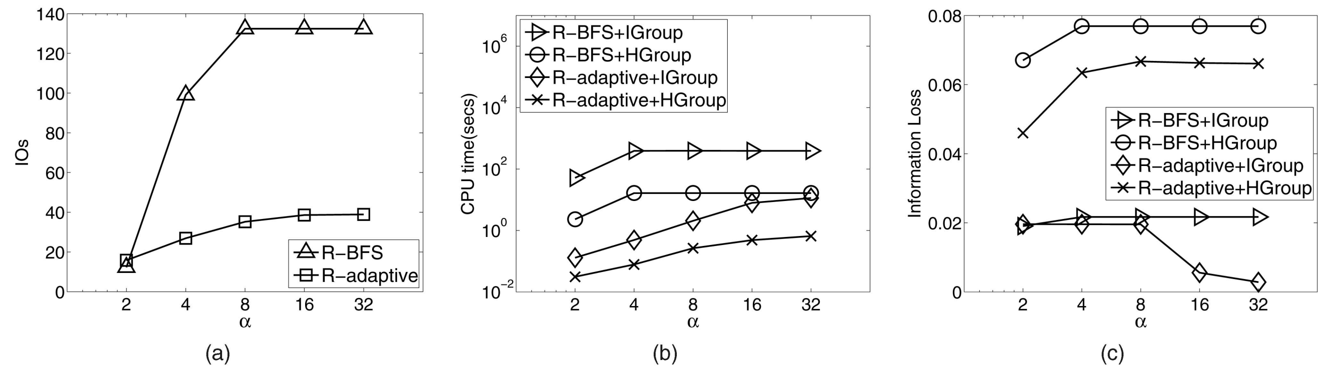


Fig. 12. Experimental results with varying α on the CA data set. (a) I/O cost. (b) CPU time. (c) Information loss.

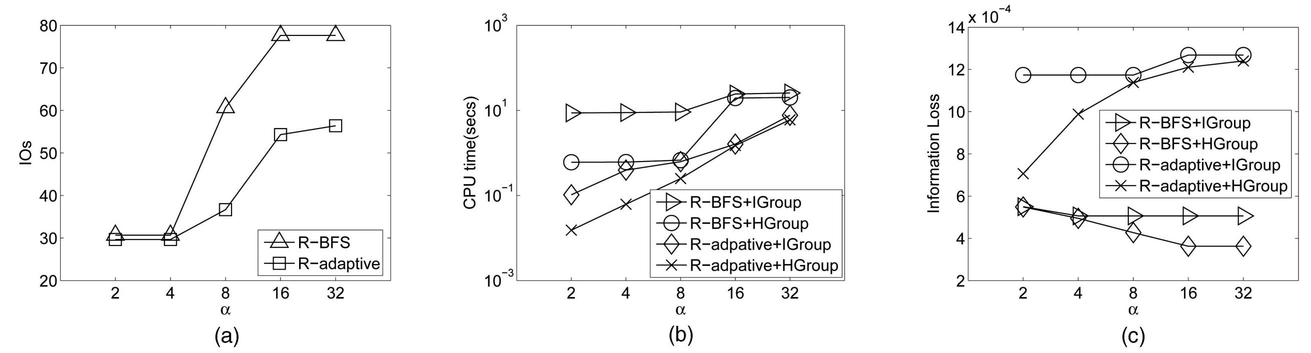


Fig. 13. Experimental results with varying α on the $Skew$ data set. (a) I/O cost. (b) CPU time. (c) Information loss.

properties are lost. While Hilbert-curve-based clustering algorithms [12] generally work well, our problem is fundamentally different from traditional clustering problems (as discussed in Section 2). Thus, simply extending the 1D dynamic programming algorithm to two dimensions by Hilbert curves does not work as well as *IGroup*, which is directly designed for two dimensions. Second, coupled with the same grouping algorithm, *R-Adaptive* works much better than *R-BFS*. This means that visiting the R-tree more selectively does not only save the traversal cost, but also leads to better quality of the results. In addition, we observe that larger ks improve the quality for all the algorithms. This is intuitive, since as the size of the concise representation increases, it becomes closer and closer to the exact results. For NA and TG data sets, the query performance is similar and omitted due to space limit.

6.6 Experimental Results with Varying α

In the second set of experiments, we fixed the query size at 0.1×0.1, $k = 60$, and varied α from 2 to 32. In Fig. 12, we plot the I/O cost, CPU time, and information loss for the four algorithms on the CA data set. The trends on the I/O cost and CPU time in this set of experiments are similar to those in the experiments with varying k. This is not surprising, since both the I/O cost and CPU time depend only on the product $\alpha \times k$. The larger this product is, the more costly the algorithms are. Thus, the combination of $R\text{-}Adaptive + IGroup$ is still the best in all algorithms in terms of information loss and its I/O cost.

For $R\text{-}Adaptive + IGroup$, we do see that larger αs lead to better results since the R-tree traversal can return more candidate MBRs for *IGroup* to choose to form the partitions (Fig. 12c). But we do not find the same trend with the other algorithms, which could mean that although *R-BFS* returns more candidate MBRs, they are of low quality. *HGroup* does not work well even with more freedom in grouping these MBRs, as long as the result size constraint, k, is fixed.

Furthermore, Figs. 13a, 13b, and 13c illustrate the query performance on synthetic $Skew$ data set with the same settings. The experimental results are similar to that of CA. We do not show the similar results on NA and TG data sets here due to space limit.

6.7 Experimental Results with Varying Query Size

In the last set of experiments, we fixed $k = 60$, $\alpha = 8$, and varied the query size from 0.05×0.05 to 0.2×0.2. The results on I/O cost, CPU time, and information loss on the CA data set are shown in Fig. 14. First, similar to the previous two sets of experiments, the I/O cost of the algorithms increases as the query range gets larger. But this time, the increase rate of *R-BFS* is slower than that when $\alpha \times k$ increases. This can be explained by the fact that when $\alpha \times k$ increases, whenever a certain level of the R-tree does not contain enough MBRs, $R\text{-}BFS$ simply goes one level down, increasing the number of MBRs returned exponentially. When $\alpha \times k$ is fixed but the query range gets larger, this increase is more "gradual": $R\text{-}BFS$ will not go deep into the R-tree but rather go wide. So, the number of page accesses increases more smoothly in this case. Nevertheless, the I/O cost of *R-BFS* is still much larger than that of *R-Adaptive*. In terms of CPU time, the trend is similar, i.e., all the algorithms take longer to run as the query gets larger. This is because of the high query selectivity, which requires more cost to retrieve and process data. Therefore, in practice, if small response time is strictly

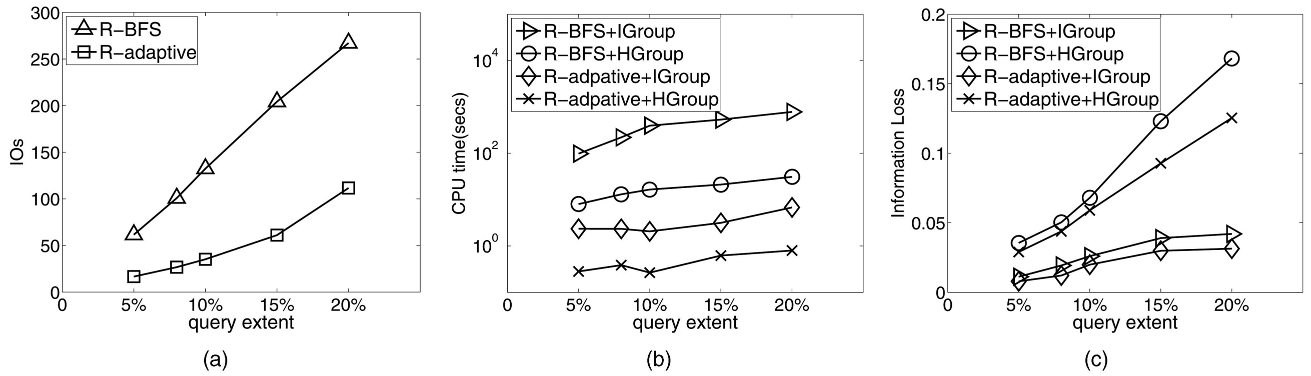


Fig. 14. Experimental results with varying query range on the CA data set. (a) I/O cost. (b) CPU time. (c) Information loss.

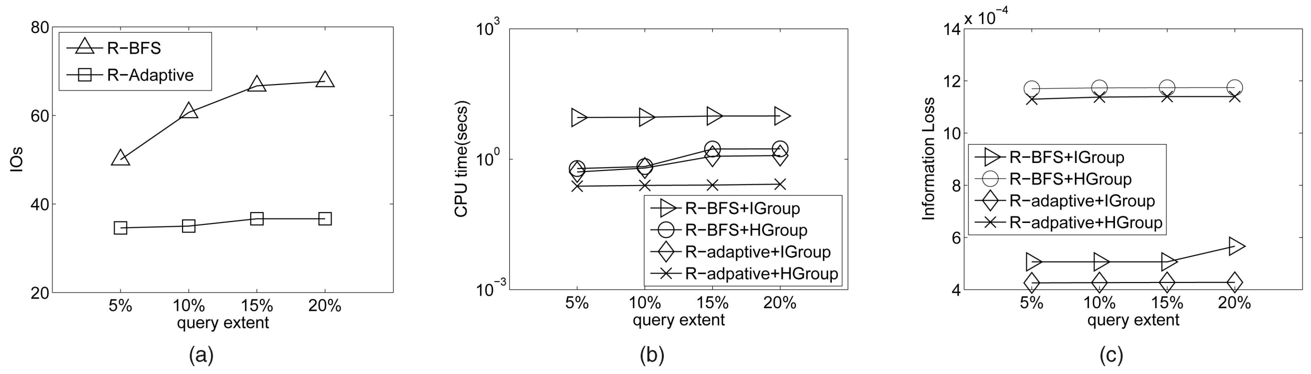


Fig. 15. Experimental results with varying query range on the *Skew* data set. (a) I/O cost. (b) CPU time. (c) Information loss.

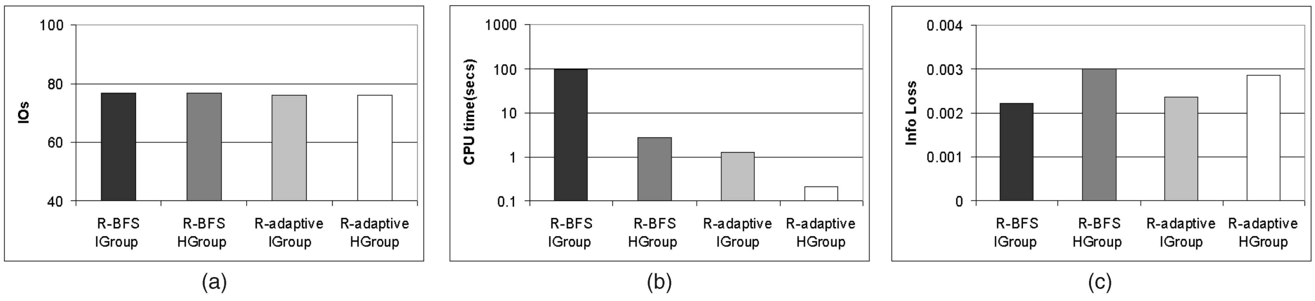


Fig. 16. Experimental results on *Skew* data set with objects of rectangular shape. (a) I/O cost. (b) CPU time. (c) Information loss.

required (or the query region is large), we can use the HGroup-based approach to obtain fast answers; otherwise, the IGroup-based one can be used. Moreover, the information loss also gets larger for larger queries, which is not surprising. But across all queries, $R\text{-}Adaptive + IGroup$ consistently beats all other alternatives.

Figs. 15a, 15b, and 15c show similar results on synthetic *Skew* data set under the same settings. For NA and TG data sets, the trends are similar and omitted due to space limit.

Finally, we also report the performance of our proposed four approaches with spatial objects of rectangular shape rather than data points. In particular, we first treat points in *Skew* data set as centers of objects, and then generate their (half) extents with a random value within [0, 1]. The results are reported in Fig. 16 in terms of I/O cost, CPU time, and information loss. We can see that the trend of our approaches on objects of rectangular shape is similar to that of point data sets.

7 Related Work

The motivation of this work is very similar to the recent work of Jermaine et al. [2]. The focus of [2] is to produce approximate results for long-running join queries in a relational database engine at early stages of the query execution process. The "approximation" defined there is a random sample of the final results. As we elaborated in Section 2, due to the nice geometric properties of range queries in spatial databases, it is important to design more effective and efficient methods than random sampling. The goal of this work is thus to derive such a concise representation for range queries with the minimal amount of information loss. We also make use of multidimensional indexing structures to speed up the query answering process. Hence, although the motivations are similar, our ideas and techniques are fundamentally different from those in [2]. With similar arguments, our work also bears the same motivation as finding the representative skyline points [1];

however, we focus on range queries rather than dominance points.

Section 2 has pointed out the close relationship between the concise representation problem and classic clustering problems. I/O-efficient clustering algorithms have been studied in [15], [16]. In particular, k-medoids (k-means with the constraint that the cluster center must be a point from the input data set) and k-centers have been extended to work for disk-based data sets using R-trees [17]. The basic idea is to explore the R-tree (in a top-down fashion) to the first level that provides enough nodes for clustering. Then, some main memory heuristics are applied to these nodes. Of course, additional information, such as the means of all points within the subtree of one node and the total number of points in the same subtree, must be stored in R-tree nodes to enable this process. Our work focuses on a completely different definition of clustering, as Section 2 has illustrated the limitations of using either k-means or k-centers for our problem. Furthermore, instead of simply retrieving one level of the R-tree that provides enough nodes for deriving k partitions, we have designed an adaptive strategy to explore R-tree nodes at different levels.

Density based clustering approaches, such as CLARANS [7] and DBSCAN [18], are another popular clustering definition. Classic-density-based clustering and other variants have been extended to work with disk-based data sets as well [15]. Several works have studied density-based clustering in the context of spatial databases and extended to road networks [19] and moving objects [20]. Similarly as above, Section 2 has examined why it is inappropriate to use it as the concise representation. Nevertheless, an interesting open problem for our work is to study the concise range queries on road networks [19] and for moving objects [20].

In addition, Xu et al. [5] studied the utility-based anonymization, which proposes heuristic local recoding methods for privacy preserving, considering the utility of attributes. Although the proposed framework for the greedy top-down method is somewhat similar to *IGroup*, the fundamental problem (i.e., k-anonymity) is different from ours. As mentioned in Section 1.1, the main difference of our work from k-anonymity [3], [4], [5] is that k-anonymity requires each cluster to contain at least k points (for the sake of privacy preserving) rather than the number of clusters.

Our work is also related to the data summarization techniques such as histogram [28], [29], [30]. Specifically, Ioannidis and Poosala [28] studied the V-optimal histogram for 1D data, whereas Acharya et al. [29] investigated the histogram, namely *MinSkew*, in multidimensional space for spatial data. Note that Section 3.1 discusses how to obtain optimal concise representations on 1D data, whose goal is, however, different from that of V-optimal histogram. That is, the goal of our problem is to minimize the information loss, which is defined as summing up the multiplication of frequency and the extent (i.e., the sum of spans of all dimensions) of each partition, rather than the estimation error. Furthermore, Jagadish et al. [30] worked on a dual problem of constructing a histogram, minimizing the space under the constraint of an acceptable error bound (for selectivity estimation). In contrast, our problem does not have the requirement of minimizing the space.

Finally, sampling-based techniques could be applied to derive concise representations for the results of range queries. Section 2 has argued that random sampling requires much larger sizes to be representative and they are also ineffective in capturing the outliers. The key observation is that sampling is a very general method that works for almost any type of data. Spatial data has special geometric properties that one should exploit. Other sampling techniques, such as the density biased sampling [21] and many others, have similar limitations in the context of range queries.

The basic idea of representing range queries in a concise format has been published as a short paper in ICDE '09 [22]. However, the only technical contents in [22] are the basic 1D dynamic programming algorithm and the *HGroup* algorithm. The NP-hardness result for two dimensions, the *IGroup* algorithm, the R-tree-based algorithms, the extensions, as well as the experiments, are all new in this paper.

8 CONCLUSION

A new concept, that of concise range queries, has been proposed in this paper, which simultaneously addresses the following three problems of traditional range queries. First, it reduces the query result size significantly as required by the user. The reduced size saves communication bandwidth and also the client's memory and computational resources, which are of highest importance for mobile devices. Second, although the query size has been reduced, the usability of the query results has been actually improved. The concise representation of the results often gives the user more intuitive ideas and enables interactive exploration of the spatial database. Finally, we have designed R-tree-based algorithms so that a concise range query can be processed much more efficiently than evaluating the query exactly, especially in terms of I/O cost. This concept, together with its associated techniques presented here, could greatly enhance user experience of spatial databases, especially on mobile devices, by summarizing "the world in a nutshell."

The concept of concise range queries is quite general. One can imagine that it could naturally extend to deal with moving objects [23], [20], [24], uncertainty and fuzziness in data [25], [26], [27], etc. However, designing efficient and effective query processing algorithms for these types of data remains a challenging open problem.

ACKNOWLEDGMENTS

Ke Yi is supported in part by the Hong Kong DAG grant (DAG07/08). Xiang Lian and Lei Chen are supported by the Hong Kong RGC Grant No. 611608 and the NSFC Projects under grant Nos. 60533110 and 60763001. Feifei Li is partially supported by the Startup Grant from the Computer Science Department, Florida State University.

REFERENCES

[1] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting Stars: The k Most Representative Skyline Operator," *Proc. Int'l Conf. Data Eng. (ICDE)*, 2007.

[2] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra, "Scalable Approximate Query Processing with the dbo Engine," *Proc. ACM SIGMOD*, 2007.

[3] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis, "Fast Data Anonymization with Low Information Loss," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 2007.

[4] G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy, D. Thomas, and A. Zhu, "Achieving Anonymity via Clustering," *Proc. Symp. Principles of Database Systems (PODS)*, 2006.

[5] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A.W.-C. Fu, "Utility-Based Anonymization Using Local Recoding," *Proc. ACM SIGKDD*, 2006.

[6] C. Böhm, C. Faloutsos, J.-Y. Pan, and C. Plant, "RIC: Parameter-Free Noise-Robust Clustering," *ACM Trans. Knowledge Discovery from Data*, vol. 1, no. 3, pp. 10-1-10-28, 2007.

[7] R.T. Ng and J. Han, "Efficient and Effective Clustering Methods for Spatial Data Mining," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 1994.

[8] D. Lichtenstein, "Planar Formulae and Their Uses," *SIAM J. Computing*, vol. 11, no. 2, pp. 329-343, 1982.

[9] R. Tamassia and I.G. Tollis, "Planar Grid Embedding in Linear Time," *IEEE Trans. Circuits and Systems*, vol. 36, no. 9, pp. 1230-1234, Sept. 1989.

[10] H.V. Jagadish, B.C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "iDistance: An Adaptive B+-Tree Based Indexing Method for Nearest Neighbor Search," *ACM Trans. Database Systems*, vol. 30, no. 2, pp. 364-397, 2005.

[11] H. Samet, *The Design and Analysis of Spatial Data Structures.* Addison-Wesley Longman Publishing Co., Inc., 1990.

[12] B. Moon, H.v. Jagadish, C. Faloutsos, and J.H. Saltz, "Analysis of the Clustering Properties of the Hilbert Space-Filling Curve," *IEEE Trans. Knowledge and Data Eng.*, vol. 13, no. 1, pp. 124-141, Jan. 2001.

[13] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD*, 1984.

[14] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD*, 1990.

[15] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," *Proc. ACM SIGMOD*, 1996.

[16] V. Ganti, R. Ramakrishnan, J. Gehrke, and A. Powell, "Clustering Large Datasets in Arbitrary Metric Spaces," *Proc. Int'l Conf. Data Eng. (ICDE)*, 1999.

[17] K. Mouratidis, D. Papadias, and S. Papadimitriou, "Tree-Based Partition Querying: A Methodology for Computing Medoids in Large Spatial Datasets," *VLDB J.*, vol. 17, no. 4, pp. 923-945, 2008.

[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *Proc. Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 1996.

[19] M.L. Yiu and N. Mamoulis, "Clustering Objects on a Spatial Network," *Proc. ACM SIGMOD*, 2004.

[20] C.S. Jensen, D. Lin, B.C. Ooi, and R. Zhang, "Effective Density Queries on Continuously Moving Objects," *Proc. Int'l Conf. Data Eng. (ICDE)*, 2006.

[21] C.R. Palmer and C. Faloutsos, "Density Biased Sampling: An Improved Method for Data Mining and Clustering," *Proc. ACM SIGMOD*, 2000.

[22] K. Yi, X. Lian, F. Li, and L. Chen, "The World in a Nutshell: Concise Range Queries," *Proc. Int'l Conf. Data Eng. (ICDE)*, 2009.

[23] P.K. Agarwal, L. Arge, and J. Erickson, "Indexing Moving Points," *Proc. Symp. Principles of Database Systems (PODS)*, 2000.

[24] Y. Tao, D. Papadias, and J. Sun, "The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 2003.

[25] N. Dalvi and D. Suciu, "Efficient Query Evaluation on Probabilistic Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 2004.

[26] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Evaluating Probabilistic Queries over Imprecise Data," *Proc. ACM SIGMOD*, 2003.

[27] A.D. Sarma, O. Benjelloun, A. Halevy, and J. Widom, "Working Models for Uncertain Data," *Proc. Int'l Conf. Data Eng. (ICDE)*, 2006.

[28] Y.E. Ioannidis and V. Poosala, "Balancing Histogram Optimality and Practicality for Query Result Size Estimation," *Proc. ACM SIGMOD*, 1995.

[29] S. Acharya, V. Poosala, and S. Ramaswamy, "Selectivity Estimation in Spatial Databases," *Proc. ACM SIGMOD*, 1999.

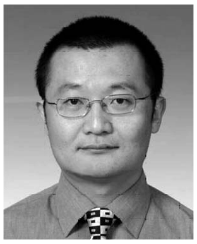
[30] H.V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik, and T. Suel, "Optimal Histograms with Quality Guarantees," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 1998.

[31] T. Brinkhoff, "A Framework for Generating Network-Based Moving Objects," *Geoinformatica*, vol. 6, pp. 153-180, 2002.

Ke Yi received the BE degree in computer science from Tsinghua University in 2001, and the PhD degree in computer science from Duke University in 2006. After spending one year at AT&T Labs as a researcher, he has been an assistant professor in the Department of Computer Science and Engineering at The Hong Kong University of Science and Technology since 2007. His research interests include algorithms, data structures, and databases.

Xiang Lian received the BS degree from the Department of Computer Science and Technology, Nanjing University, in 2003. He is currently working toward the PhD degree in the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology. His research interests include query processing over stream time series and uncertain databases. He is a student member of the IEEE.

Feifei Li received the BS degree in computer engineering from Nanyang Technological University in 2002, and the PhD degree in computer science from Boston University in 2007, respectively. He has been an assistant professor in the Computer Science Department at Florida State University since 2007. His research interests include data management, data structures, and databases.



Lei Chen received the BS degree in computer science and engineering from Tianjin University, China, in 1994, the MA degree from Asian Institute of Technology, Thailand, in 1997, and the PhD degree in computer science from the University of Waterloo, Canada, in 2005. He is now an assistant professor in the Department of Computer Science and Engineering at The Hong Kong University of Science and Technology. His research interests include uncertain databases, graph databases, multimedia and time-series databases, and sensor and peer-to-peer databases. He is a member of the IEEE.