Mining Thick Skylines over Large Databases

Wen Jin[1], Jiawei Han[2], and Martin Ester[1]

[1] School of Computing Science, Simon Fraser University
{wjin,ester}@cs.sfu.ca
[2] Department of Computer Science, Univ. of Illinois at Urbana-Champaign
hanj@cs.uiuc.edu

Abstract. People recently are interested in a new operator, called *skyline* [3], which returns the objects that are not dominated by any other objects with regard to certain measures in a multi-dimensional space. Recent work on the skyline operator [3, 15, 8, 13, 2] focuses on efficient computation of skylines in large databases. However, such work gives users only *thin skylines*, i.e., single objects, which may not be desirable in some real applications. In this paper, we propose a novel concept, called *thick skyline*, which recommends not only skyline objects but also their nearby neighbors within ε-distance. Efficient computation methods are developed including (1) two efficient algorithms, *Sampling-and-Pruning* and *Indexing-and-Estimating*, to find such thick skyline with the help of statistics or indexes in large databases, and (2) a highly efficient *Microcluster-based algorithm* for mining thick skyline. The *Microcluster-based method* not only leads to substantial savings in computation but also provides a concise representation of the thick skyline in the case of high cardinalities. Our experimental performance study shows that the proposed methods are both efficient and effective.

1 Introduction

In query-answering, people recently are interested in a new operator, called *skyline operator*[3]. *Given a set of n objects, the* ***skyline*** *refers to those that are not dominated by any other object. An object p* ***dominates*** *another object q, noted as $p \succ q$, if p is as good or better in all dimensions and better in at least one dimension.* A typical example is illustrated in Figure 1, showing the skyline of hotels with dimensions of the Distance (to the beach) and the Price. The hotels (a, b, c, d, e, f) are the skylines ranked as the *best* or *most satisfying* hotels.

The skyline operator can be represented by an (extended) SQL statement. An example Skyline Query of New York hotels corresponding to Figure 1 in SQL would be: *SELECT * FROM Hotels WHERE city='New York' SKYLINE OF Price min, Distance min*, where *min* indicates that the Price and the Distance attributes should be minimized. For simplicity, we assume that skylines are computed with respect to *min* conditions on all the dimensions, though it can be a combination with other condition such as *max*[3].

Most existing work on skyline queries has been focused on efficient computation of skyline objects in large databases [3, 15, 8, 13, 2]. However, the results

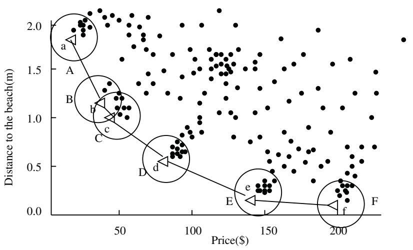

Fig. 1. Thick Skyline of N.Y. hotels.

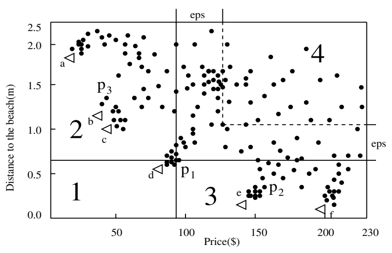

Fig. 2. Sampling objects to pruning.

obtained by the skyline operator may not always contain satisfiable information for users. Let's examine an example: Given the hotels in Figure 1, a conference organizer needs to decide the conference location. He usually will be interested in the following questions: 1. Can we find a bunch of *skyline* hotels which are nearby so as to provide good choices for the attendees? 2. If a skyline hotel is occupied, is there any nearby hotel which, though not ranked as high as skyline hotel, can still be a good candidate?

Apparently, the above questions cannot be answered directly by pure skyline computation as candidates which have similar attribute to skylines are not provided. Another problem in most of the existing studies is that they are based on the assumption of small skyline cardinality [13, 3, 8]. However, skyline objects could be many, making it inconvenient for users to browse and manually choose interesting ones. To address the problem of either two few or too many skyline objects, it seems to be natural to consider a compact and meaningful structure to represent the skyline and its neighborhood.

In this paper, we propose an extended definition of *skyline*, develop a novel data mining technique to skyline computation, and study the interesting patterns related to the skyline. The concept of skyline is extended to *generalized skyline* by pushing a user-specific constraint into skyline search space. For simplicity, the user-specific constraint is defined as the ε-neighbor of any skyline object. *Thick skyline* composed of the generalized skyline objects is the focus of the paper.

Mining the thick skyline is computationally expensive since it has to handle skyline detection and nearest neighbor search, which both require multiple database scans and heavy computation. Can we design algorithms that remove the computational redundancy in skyline detection and nearest neighbor search as much as possible? Furthermore, even in the same database system, different configurations may be required for different applications. For example, some may only allow the dataset to exist as a single file, others may have additional support with different types of index, such as B-tree, R-tree or CF-tree. How can we develop nice approaches to cope with these situations respectively?

Our contributions in this paper are as follows:

- A novel model of *thick skyline* is proposed that extends the existing skyline operator based on the distance constraint of skyline objects and their nearest neighbors.

- Three efficient algorithms, *Sampling-and-Pruning*, *Indexing-and-Estimating* and *Microcluster-based*, are developed under three typical scenarios, for mining the thick skyline in large databases. Especially, the Microcluster-based method not only leads to substantial savings in computation but also provides a concise representation of thick skyline in the case of high cardinalities.
- Our experimental performance study shows that the proposed methods are both efficient and effective.

The remaining of the paper is organized as follows. Section 2 overviews related work on the skyline. Sections 3 gives the definition of thick skyline and describes our proposed three algorithms. The results of our performance study are analyzed in Section 4. Section 5 concludes the paper with the discussion of future research directions.

2 Related Work

The skyline computation originates from the maximal vector problem in computational geometry, proposed by Kung et al. [7]. The algorithms developed [9, 14] usually suits for a small dataset with computation done in main memory. One variant of maximal vector problem, which is related to but different from the notion of thick skyline, is the *maximal layers* problem[11, 4] which aims at identifying different layers of maximal objects.

Borzsonyi et al. first introduce the skyline operator over large databases [3] and also propose a divide-and-conquer method. The method based on [7, 12] partitions the database into memory-fit partitions. The partial skyline objects in each partition is computed using a main-memory-based algorithm [14, 9], and the final skyline is obtained by merging the partial results. In [15], the authors proposed two progressive skyline computing methods. The first employs a bitmap to map each object and then identifies skyline through bitmap operations. Though the bit-wise operation is fast, the huge length of the bitmap is a major performance concern. The second method introduces a specialized B-tree which is built for each combination list of dimensions that a user might be interested in. Data in each list is divided into batches. The algorithm processes each batch with the ascending index value to find skylines.

Kossmann et al. present an online algorithm, NN, based on the nearest neighbor search. It gives a big picture of the skyline very quickly in all situations. However, it has raw performance when large amount of skyline needs to be computed. The current most efficient method is BBS (b̲ranch and b̲ound s̲kyline), proposed by Papadias et al., which is a progressive algorithm to find skyline with optimal times of node accesses [13]. Balke et al. [2] in their paper show how to efficiently perform distributed skyline queries and thus essentially extend the expressiveness of querying current Web information systems. They also propose a sampling scheme that allows to get an early impression of the skyline for subsequent query refinement.

3 The Thick Skyline and Mining Algorithms

Definition 1. (Generalized Skyline) *Given a d-dimensional database X and the skyline objects set $\{s_1, s_2, \ldots, s_m\}$, the generalized skyline is the set of all the following objects:*

- *the skyline objects, and*
- *the non-skyline objects which are ε-neighbors of a skyline object.*

We categorize the generalized skyline object into three classes: (a) a single skyline object, called *outlying skyline object*, (b) a *dense skyline object*, which is in a set of nearby skyline objects, and (c) a *hybrid skyline object*, which is in a set consisting of a mixture of nearby skyline objects and non-skyline objects.

From the data mining point of view, we are particularly interested in identifying the patterns of skyline information represented by clusters of types (b) and (c), which leads to the definition of the thick skyline.

Definition 2. (Thick Skyline) *Given a d-dimensional database X, the thick skyline is composed of all dense skyline and hybrid skyline objects.*

In this section, we explore different approaches to mining thick skyline in a d-dimensional database X with size $|X|$ under three typical situations. The first approach applies sampling and pruning technique to the relational files, and exploits the statistics of the database. The second approach estimates and identifies the range of thick skyline based on general index structures in relational database, which is not only suitable for thick skyline computation, but also composable with other relational operators. The third approach exploits the special summarization structure of micro-clusters which is widely used in data mining applications, and finds the thick skyline using bounding and pruning technique.

3.1 Sampling-and-Pruning Method

Sampling-and-Pruning method runs with the support of a database system where statistics, such as order and quantile in each dimension, can be obtained from the system catalog. The identification of thick skyline relies on the comparisons between objects. Clearly, the access order of objects crucially determines the number of comparisons. Hence we wish to pick some objects with high dominating capacity at the beginning to prune many dominated objects. As nearest neighbor search method [8] is expensive [13], a sampling method is developed.

The basic idea is as follows. We first randomly sample k ($k \ll |X|$)objects S with high dominating capacity as initial "seeds". Several criteria are required during the sampling step: (1) It prefers to choose objects with smaller values in dimensions which appear to be more dominating, and (2) the k objects are not dominated by each other. Each object of S can be taken temporarily as "skyline" objects to compare with other objects.

If the values in each dimension are distributed independently, an alternative but more aggressive sampling method [1] can also be applied to construct each of the k sampling objects by choosing d (smaller) values in each dimension (i.e., such k objects may not necessarily be in the dataset). Figure 2 shows a 2-dimensional hotel dataset partitioned into regions 1, 2, 3 and 4 by a randomly sampled object p_1. The pruning capacity of this sampling can be analyzed probabilistically. Assuming there are n objects and the largest values in Price and Distance axis are s and t respectively. Obviously, if p_1 is chosen properly, region 1 should not be empty, which will lead to the pruning of region 4. Otherwise it is a poor sampling object. Suppose the coordinates of p_1 is (u, v), the probability of region 1 being empty is $(\frac{s \cdot t - u \cdot v}{s \cdot t})^n = (1 - \frac{u}{s} \cdot \frac{v}{t})^n$. If u, v are chosen as the $\left\lceil \sqrt{n \ln n} \right\rceil$th smallest value in Price and Distance respectively, i.e. u and v are relatively small according to criteria 1, then the probability is $(1 - \frac{\sqrt{n \ln n}}{n} \cdot \frac{\sqrt{n \ln n}}{n})^n = (1 - \frac{\ln n}{n})^n \leq e^{-\ln n}[10] = \frac{1}{n}$, which is very small.

In the thick skyline computation process, all those non-skyline objects need to be investigated during the comparison step. In order to avoid unnecessary comparisons, we introduce a strongly dominating relationship and a lemma is deduced for pruning many non ε-neighbors of any skyline.

Definition 3. (Strongly Dominating Relationship) *An object $p \in X$ strongly dominates another object $q \in X$, noted as $p \rhd q$, if $p + \varepsilon$ dominates q, i.e. $\forall i$, $1 \leq i \leq d$, $p_i + \varepsilon \leq q_i$, and $p_i + \varepsilon < q_i$ in at least one dimension. On the other hand, q is a strongly dominated object.*

Lemma 1. *Given a dataset X, objects p and q, if $p \rhd q$, then q cannot be a thick skyline object.*

The strongly dominating relationship is illustrated by Figure 2, where objects strongly dominated by p_1 are in the dashed-lines rectangle.

Due to the space limitation, we briefly describe the algorithm as follows: First, sampling data S are generated and temporarily added to thick skyline list. In the pruning process, if an object x is strongly dominated by an object s in S, it is removed. If it is not only a dominated object but also an ε-neighbor of s, it is added to the neighbor list of s. If x dominates s, remove s and its strongly dominated neighbors by x and add x into the list. Finally after the pruning process, the thick skyline of a small amount of remaining object can be computed using any method such as the Indexing-and-Estimating Method.

3.2 Indexing-and-Estimating Method

Based on database index such as B-tree, by combining range estimate of the batches in the "minimum dimension" index [15] with an elaborate search technique, we can find the thick skyline within one scan of the database.

Assume that X is partitioned into d lists such that an object $p = (p_1, p_2, \ldots, p_d)$ is assigned to the i-th list $(1 \leq i \leq d)$ if and only if p_i is the minimum among all dimensions. Table 1 shows an example. Objects in each list are sorted in ascending order of their minimum coordinate ($minC$, for short). A *batch* in the i-th list consists of objects having the same $minC$. In computing of skylines, initially the

Table 1. The index approach.

List1		List2	
$a(1,9)$	$minC = 1$	$k(9,1)$	$minC = 1$
$b(2,10)$	$minC = 2$	$i(3,2), m(6,2)$	$minC = 2$
$c(4,8)$	$minC = 4$	$h(4,3), n(8,3)$	$minC = 3$
$g(5,6)$	$minC = 5$	$l(10,4)$	$minC = 4$
$d(6,7)$	$minC = 6$	$f(7,5)$	$minC = 5$
$e(9,10)$	$minC = 9$		

first batches of all the lists are accessed and the one with the minimum $minC$ is processed. We assume the current minimum batches in the two lists of Table 1 are $minC_1$ and $minC_2$ respectively. Since $\{a\}$ and $\{k\}$ have identical $minC$, the algorithm picks $\{a\}$ and adds it to the skyline list. As the next batch $\{b\}$ has $minC_1 = 2$, $\{k\}$ in list 2 with $minC_2 = 1$ is processed and inserted into the list as it is not dominated by a. Then, the next batch handled is $\{b\}$ in list 1, where b is dominated by a in the list. Similarly, batch $\{i, m\}$ is processed and i is added to the skyline. At this step, no further batches need to be processed as the remaining objects in both lists are dominated by i and the skyline are $\{a, i, k\}$. During the search of a skyline, the range where its ε-neighbors exist is given in the following lemma.

Lemma 2. *Given d index lists of X, and a skyline object $p = (p_1, p_2, \ldots, p_d)$ is in the batch $minC = p_i$ of the ith list, then:*
(a) the ε-neighbors of p can only possibly exist in the batch range $[p_i - \varepsilon, p_i + \varepsilon]$ of the i-th list; and the batch range $\left[p_j - \varepsilon, p_j + \frac{\varepsilon}{\sqrt{2}}\right]$ of the j-th list ($j \neq i$);
(b) p does not have any ε-neighbor in jth list ($j \neq i$) if $(p_j - p_i) > \sqrt{2} \cdot \varepsilon$.

Proof. *(a) The proof of bounds in the i-th list and the lower bound in the j-th list are straightforward. Assume a ε-neighbor of p in the j-th list is $p' = (p'_1, p'_2, \ldots, p'_d)$ and p' exists in a batch with $minC = p'_j > p_j + \frac{\varepsilon}{\sqrt{2}}$, then $p'_i > p'_j > p_j + \frac{\varepsilon}{\sqrt{2}} > p_i + \frac{\varepsilon}{\sqrt{2}}$, we have $|p'_i - p_i| > \frac{\varepsilon}{\sqrt{2}}$ and $\left|p'_j - p_j\right| > \frac{\varepsilon}{\sqrt{2}}$, so $(\sum_{i=1}^{d} |p'_i - p_i|^2)^{\frac{1}{2}} > \varepsilon$, contradicting the definition. (b) As shown in Figure 3 (Eps is ε), all the objects in the j-th list can only appear in area I, while i-th in II. $dist(p, q) = p_j - p_i$, and $dist(p, o)$, is the shortest distance from p to any object in I. If $dist(p, o) > \varepsilon$, which means $dist(p, q) > \sqrt{2} \cdot \varepsilon$, then no ε-neighbor exist in I.*

In the dynamic scanning process of each list, if a skyline object p in the i-th list is found, how far shall we go back to find some of its ε-neighbors in the j-th list if the lower range bound is smaller than the $minC_j$ of the current batch? For those neighbors residing in batches greater than $minC_j$, we can certainly leave them to the remaining sequential scan. We show that only a ε length sliding window around the current batch $minC_j$ (denoted as SW_{minC_j}) needs to be maintained for each list, hence avoiding repeated backwards scans of the list and incur more overhead. The batch number $minC$ within the slide window SW_{minC_j} satisfies $minC_j - \varepsilon < minC < minC_j$.

Lemma 3. *Given d index lists of X, a skyline object $p = (p_1, p_2, \ldots, p_d)$ is found in the i-th list, while the current batch in the j-th list is $minC_j$ ($1 \le j \le d$ and $i \neq j$), if there are ε-neighbors of p existing in the batches with $minC \le minC_j$ in the j-th list, then they can only exist in the slide window SW_{minC_j}.*

Proof. *Since batch $minC_i$ in the i-th list is the one the skyline searching algorithm is now handling, $minC_i \ge minC_j$. Also we have $p_j > p_i$ and $p_i = minC_i$. The lower bound of the batch range $p_j - \varepsilon > p_i - \varepsilon = minC_i - \varepsilon \ge minC_j - \varepsilon$ which is covered by the slide window SW_{minC_j}.*

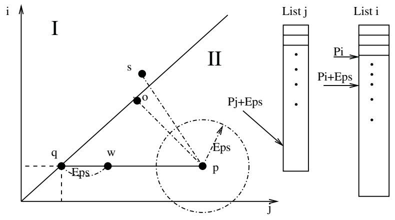

Fig. 3. Evaluate Neighborhood Scope.

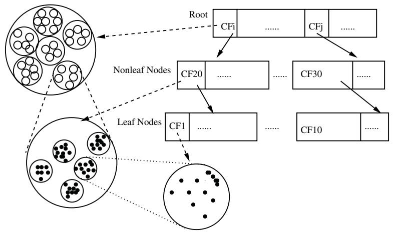

Fig. 4. Microclusters.

Lemma 3 also ensures an important property: Whenever an object is out of the slide window, it will never become an ε-neighbor for any skyline which enables us to find thick skyline within one scan. The algorithm pseudocode is as below.

Algorithm 1 An Indexing-and-Estimating Method.
Input: B-tree of d lists index and distance threshold ε.
Output: The thick skyline T.
Method:

```
1.  $S = \emptyset; T = \emptyset$;
2.  FOR $i = 1$ to $d$ DO;
3.      $SW_i = \emptyset; upper_i = |list_i|; minC_i = \min list_i$;
4.  WHILE $(Thin - Skyline - Search - Unfinished)$ DO;
5.      Choose the batch with $\min minC_1, \ldots, minC_d$, say $minC_k$;
6.      Check each object $p$ in this batch;
7.      IF $p$ is a skyline object THEN
8.          $S = S \cup \{p\}$;
9.          IF $(p_j - p_i) < \sqrt{2} \cdot \varepsilon$ THEN
10.             update $upper_j$ to $p_j + \frac{\varepsilon}{\sqrt{2}}$; check $SW_j$ for $\varepsilon$ neighbor ;
11.             IF any $q$ is a $\varepsilon$ neighbor THEN
12.                 $T = T \cup \{q\}$;
13.     ELSE IF $p$ is an $\varepsilon$-neighbor THEN
14.             $T = T \cup \{p\}$;
15.     Move $list_k$ to next batch and update $SW_k$;
16. WHILE $list_1 < upper_1 \vee \ldots \vee list_d < upper_d$ DO;
17.     scan objects to find $\varepsilon$ neighbors and add to $T$;
18. $T = T \cup S$; Output thick skyline $T$;
```

The algorithm initiates skyline list and ε-neighbors list (Step 1), current batches, slide windows and the upper bound batch to scan in each list (Step 2-3). Each object p in the minimum $minC_i$ is compared with the skyline list (Step 6). If p is a skyline object, the corresponding range is updated, and part of p's ε-neighbors may be found in the slide windows (Step 8-12), while others are left to the remaining access of the lists(Step 13-14). Step 16-17 calculates the possible ε-neighbors in the range when thin skyline search finishes. Finally, it will output both skyline objects and ε-neighbors (Step 18).

3.3 Microcluster-Based Method

In order to scale-up data mining methods to large databases, a general strategy is to apply data compression or summarization. For example, we can partition the database into microclusters based on CF-tree [16, 6].

Definition 4. (Microcluster) *A microcluster for a set of d-dimensional objects $X_1 \ldots X_n$, $X_i = (x_i^1 \ldots x_i^d)$, is defined as a $(4 \cdot d + 1)$-tuple ($\overline{CF1^x}$, $\overline{CF2^x}$, $\overline{CF3^x}$, $\overline{CF4^x}$, n), where $\overline{CF1^x}$, $\overline{CF2^x}$, $\overline{CF3^x}$, and $\overline{CF4^x}$ each represents a vector of d entries. The definition of each of these entries is as follows:*

- *The p-th entry of $\overline{CF1^x}$ is equal to $\sum_{j=1}^{n} x_j^p$.*
- *The p-th entry of $\overline{CF2^x}$ is equal to $\sum_{j=1}^{n} (x_j^p)^2$.*
- *The p-th entry of $\overline{CF3^x}$ is equal to $\min_{j=1}^{n}(x_j^p)$.*
- *For each dimension, the data with the minimum distance to the origin is maintained in $\overline{CF4^x}$.*
- *The number of data points is maintained in n.*

The centroid x_a and radius r_a of a microcluster mc_a can be represented as: $x_a = \frac{\overline{CF1^x}}{n}$, and $r_a = (\frac{\sum_{j=1}^{n}(x_j - x_a)^2}{n})^{\frac{1}{2}} = (\frac{\overline{CF2^x} + n \cdot x_a - 2 \cdot x_a \cdot \overline{CF1^x}}{n})^{\frac{1}{2}}$. The minimum distance $mdist_o$ from a microcluster to the origin is determined by $\overline{CF4^x}$.

For mining thick skyline, the database is partitioned into a set of microclusters with radius r_i (r_i can be around ε) in the leaf nodes of an extended CF-tree. Each non-leaf node represents a larger microcluster consisting of all its sub-microclusters as shown in Figure 4. There may exist overlap between microclusters and some methods [5, 6] can be used to remedy this problem.

The dominating relationship can be applied to the microclusters. For any two microclusters mc_a and mc_b, if $x_a \succ \overline{CF3_b^x}$, then $mc_a \succ mc_b$, that is, the objects in mc_b must be dominated by some objects in mc_a. As the number of microclusters is much less than that of objects, the computation cost is very low.

Let us now examine the neighborhood relationship between microclusters. Supposed object p is in a microcluster mc_a, the distance between mc_a and any microcluster mc_b is represented as: $dist_m(mc_a, mc_b) = dist(x_a, x_b) - r_a - r_b$. If $dist_m(mc_a, mc_b) < \varepsilon$, then mc_b and mc_a are *ε-neighboring microclusters* for p.

The basic idea of mining thick skyline is as follows. Instead of accessing every object in the dataset, we only need to identify the microclusters that contain skyline objects (called *skylining microclusters*), then find which microclusters are their ε-neighbors. The thick skyline objects can finally be determined from those microclusters. Skylining microclusters is an appropriate summarization of thick skyline in the case of large number of skylines or dynamic dataset.

The algorithm starts at the root node of the CF-tree and searches the microcluster in the ascending order of distance $mdist_o$. Initially, the algorithm selects the minimum one. Since the CF-tree is a hierarchical structure, the corresponding microcluster mc_i in the leaf node can be quickly located. mc_i is a skylining microcluster and is added to a heap h_1 sorted by the distance $mdist_o$. Then the algorithm selects the microcluster with the next minimum distance to the origin. If $\overline{CF3^x}$ of the selected microcluster is dominated by the centroid of any microcluster in h_1, it cannot contain skyline objects and we simply skip it. If it is strongly dominated by any microcluster in h_1, it can be pruned. Otherwise, the selected microcluster will be added to h_1. The algorithm continues until all of them are visited. As only the statistics of the microcluster is accessed, the cost is low.

Afterwards, the algorithm visits heap h_1, and extracts the microcluster mc'_i at the top of the heap. Within mc'_i, object $\overline{CF4^x}$ is the skyline object, and the remaining objects are examined for skylineness in the order of $mdist_o$ (property guaranteed by [8]). Then a group ε-neighbors search for all the skyline objects in mc'_i is launched by searching *ε-neighboring microclusters*. Using the extended CF-tree, we simply check whether mc'_i intersects with larger microclusters in the root node, then with the non-leaf nodes, and finally locate the desired microclusters in the leaf nodes. The search complexity is bounded by the tree height and the intersected number of microclusters in the tree. The objects in these neighboring microclusters are examined whether they are ε-neighbors of skyline objects in mc'_i. The microcluster mc'_i is then removed from h_1, and the algorithm terminates until h_1 is empty. Based on the above description, The pseudocode for the Microcluster-based algorithm is as follows.

Algorithm 2 A Microcluster-based Method.
Input: m microclusters, and the distance threshold ε.
Output: The thick skyline.
Method:

```
1.  $S = \emptyset$; $T = \emptyset$; $heap_1 = \emptyset$;
2.  WHILE any $mc_i$ with $\min mdist_o$ not visited DO
3.      IF $\neg(mc_j \in heap_1 \succ mc_i)$ THEN
4           Add $mc_i$ to $heap_1$;
5.  WHILE $heap_1$ is not empty DO
6.      Select $mc'_i$ at the top of $heap_1$;
7.      Select object $p$ in $mc'_i$ with $\min mdist_o$;
8.      IF $\neg(p \in S \succ p)$ THEN;
9.          Add $p$ to $S$;
10.     Find neighboring microclusters of $mc'_i$;
11.     Add $\varepsilon$-neighbors of skyline in $mc'_i$ to $T$;
12. Output thick skyline $S \cup T$;
```

4 Experiments

In this section, we report the results of our experimental evaluation in terms of efficiency and effectiveness. We compare the runtime performance and evaluate several factors such as the choice of ε. We focus on the cost in the computing stage instead of pre-processing stage such as index or CF-tree building. Following similar data generation methods in [3], we employ two types of datasets: independent databases where the attribute values of tuples are generated using uniform distributions and anti-correlated datasets where the attribute values are good in one dimension but are bad in one or all other dimensions. The dimensionality of datasets d is between 2 and 5, the value of each dimension is in the integer range [1, 1000] and the number of data (cardinality) N is between 100k and 10M. We have implemented the three proposed methods in C++. All the experiments are conducted on Intel 1GHZ processor with 512M RAM.

• Runtime Performance To investigate the runtime versus different dimensionalities, We use dataset with cardinality 1M. Figures 5 and 6 depict the result in independent and anti-correlated distribution respectively. In both cases, Indexing-and-Estimating method achieves best performance in small dimensionality(d=2), due to its list structure being most suitable for relative small dataset and skyline size. Microcluster-based method is best towards large dimensionality ($d > 2$) and large skyline size, and the Sampling-and-Pruning method ranks the third.

Figures 7 and 8 show the runtime w.r.t. varied cardinality in independent and anti-correlated distributed 3-d datasets respectively($\varepsilon = 1$). In both cases, Microcluster-based method starts to over compete Indexing-and-Estimating when $N > 600K$ due to its region pruning and good scalability of hierarchical structure. As there is no index to facilitate computation, Sampling-and-Indexing still ranks the third, but the run time is not bad even when cardinality N=10M.

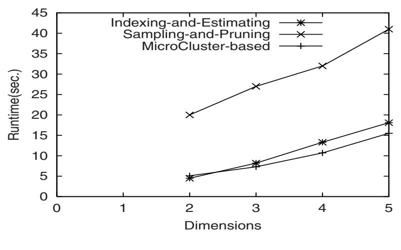

Fig. 5. Runtime vs. Dimensionality(I).

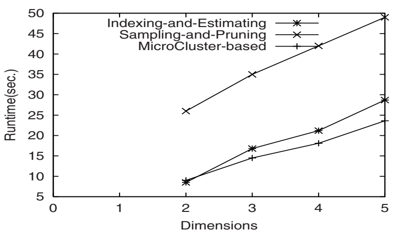

Fig. 6. Runtime vs. Dimensionality(II).

• The Effect of ε Obviously, the choice of ε values will affect the size of thick skyline. ε is usually small w.r.t. the domain bound, reflecting the "local neighborhood", and can be recommended by the system as an initial parameter for the future interaction. When we increase ε value from 1 to 30 in 1M independent 3-d dataset, Figure 12 and Figure 9 show that both the number of thick skyline and the run time of all algorithms increase. In particular, Microcluster-based method

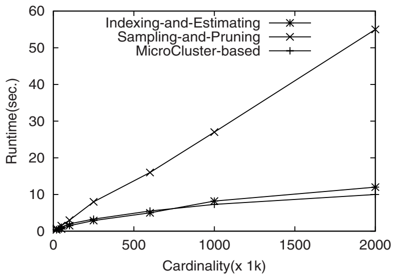

Fig. 7. Runtime vs. Cardinality(I).

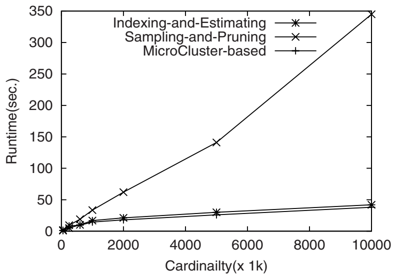

Fig. 8. Runtime vs. Cardinality(II).

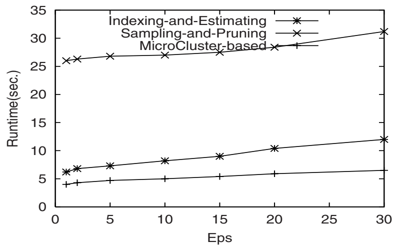

Fig. 9. Runtime vs. Eps.

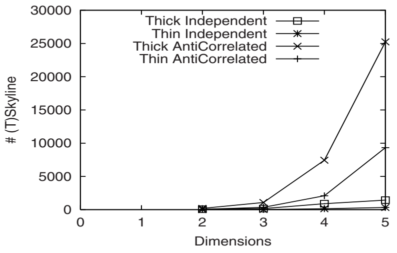

Fig. 10. Dimensionality vs. (T)Skyline.

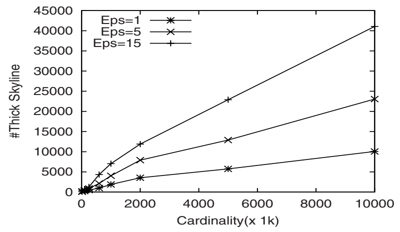

Fig. 11. Thick Skyline vs. Cardinality.

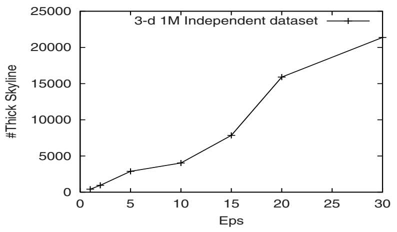

Fig. 12. Thick Skyline vs. Eps.

is always the best and keeps a good scalability. Indexing-and-Estimating method is better than Sampling-and-Pruning method in runtime.

• The Effect of Dimensionality and Cardinality The change of dimensionality will affect the size of thick skyline which is illustrated in Figure 10. With the cardinality of 100K, ε chosen as the square root of sum of 0.1% of the maximum value in each dimension, Figure 10 shows the size of both thin and thick skylines in different dimensions. We notice that if dimensionality increases, the number of the thick skyline objects increases dramatically for both independent and anti-correlated distributed dataset, with the latter having higher speed. The affect of cardinality is shown in Figure 11.

5 Conclusions

The paradigm of rank-aware query processing, in particular, the new *skyline operator*, has recently received a lot of attention in database community. In this paper, we propose a novel notion of *thick skyline* based on the distance constraint of a skyline object from its neighbors. The task of mining thick skyline is to recommend skyline objects as well as their ε-distance neighbors. We develop three algorithms, Sampling-and-Pruning, Indexing-and-Estimating and Microcluster-based, to find such thick skylines in large databases. Our experiments demonstrate the efficiency and effectiveness of the algorithms. We believe the notion of thick skyline and mining methods not only extends the skyline operator in database query, but also provides interesting patterns for data mining tasks. Future work includes mining thick skylines in data streams and combining the task with other regular data mining process.

References

1. J. L. Bentley, K. L. Clarkson, D. B. Levine. Fast linear expected-time alogorithms for computing maxima and convex hulls. In *SODA 1990*.
2. W.-T. Balke, U. Güntzer, J. X. Zheng. Efficient Distributed Skylining for Web Information Systems In *EBDT 2004*.
3. S. Borzsonyi, D. Kossmann, and K. Stocker, The Skyline Operator. In *ICDE 2001*.
4. T. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. Introduction to Algorithms, second edition. The MIT Press, 2001.
5. Alexander Hinneburg, Daniel A. Keim Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. VLDB 1999: 506-517
6. W. Jin, K. H. Tung, and J. Han. Mining Top-n Local Outliers in Very Large Databases. In *KDD 2001*.
7. H. T. Kung et. al. On finding the maxima of a set of vectors. *JACM*, 22(4), 1975.
8. D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB 2002*.
9. J. Matousek. Computing dominances in E^n. *Inf. Process. Lett.*, 38(5), 1991.
10. R. Motwani, P. Raghavan. Randomized Algorithms. *Cambridge Univ. Press*, 1995.
11. F. Nielsen. Output-sensitive peeling of convex and maximal layers. Thesis, 1996
12. F. P. Preparata, and M. I. Shamos. Computational Geometry: An Introduction. Springer-Verlag, 1985.
13. D. Papadias, Y. Tao, G. Fu, and B. Seeger. An Optimal and Progressive Algoroithm for Skyline Queries. In *SIGMOD'03*.
14. I. Stojmenovic and M. Miyakawa. An Optimal Parallel Algorithm for Solving the Maximal Elements Problem in the Plane. In *Parallel Computing, 7(2)*, 1988.
15. K. Tan et al. Efficient Progressive Skyline Computation. In *VLDB 2001*.
16. T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *SIGMOD'96*.