Efficient and Progressive Algorithms for Distributed Skyline Queries over Uncertain Data

Xiaofeng Ding and Hai Jin, *Senior Member*, *IEEE*

Abstract—The skyline operator has received considerable attention from the database community, due to its importance in many applications including multicriteria decision making, preference answering, and so forth. In many applications where uncertain data are inherently exist, i.e., data collected from different sources in distributed locations are usually with imprecise measurements, and thus exhibit kind of uncertainty. Taking into account the network delay and economic cost associated with sharing and communicating large amounts of distributed data over an internet, an important problem in this scenario is to retrieve the global skyline tuples from all the distributed local sites with minimum communication cost. Based on the well-known notation of the probabilistic skyline query over centralized uncertain data, in this paper, we propose the notation of distributed skyline queries over uncertain data. Furthermore, two communication- and computation-efficient algorithms are proposed to retrieve the qualified skylines from distributed local sites. Extensive experiments have been conducted to verify the efficiency, the effectiveness and the progressiveness of our algorithms with both the synthetic and real data sets.

Index Terms—Skyline, distributed database, uncertain data.

1 Introduction

With the fast development of computing infrastructures and easily available network services, data are increasingly stored and processed distributively [12], [17], [19], [30]. More and more applications collect data from distributed sites and compute query results based on the collective view of the data from all local sites.

In many application domains, due to the large amounts of data stored and the network delay incurred, as well as the economic cost associated with data sharing and communication [1], it is often very expensive to communicate the entire data set from each local site to the centralized data server for query processing. As a result, many query semantics in such applications that rarely need transmitting every piece of data in the local sites have been well studied to speed up the computation and communication efficiency [13], [14], [15], [16]. For example, a ranking query is implemented and only the top-k records (i.e., the k largest score according to a predefined function) are retrieved [15], [29]; a skyline query is processed to obtain those points that preferred according to a customer's preference [14].

Skyline operator is an important query type, lots of efforts have been conducted to speed up the query efficiency, from both the database and networking communities [2], [3], [4], [6], [7], [14], [17], [20], [21], [31], [34] recently. Given a set of multidimensional points, a skyline query returns those points that are not dominated by other points. A point p is said to dominate another point p', if the coordinate of p on each dimension is not larger than that of p', and is strictly smaller on at least one dimension. Fig. 1 shows a classical scenario where each point in the 2-dimensional space (x, y) corresponds to a hotel record. Specially, the x-dimension represents the price of the hotel room, and the y-dimension captures its distance from the beach. The skyline hotels are those with low prices and short distances to the beach, i.e., the dots P_1, P_3, and P_5 in the figure.

However, none of the aforementioned works deal with distributed query over uncertain data, which has emerged as a major data type in many applications. Interestingly enough, many cases where data uncertainty inherently exists are distributed, e.g., distributed sensor networks with imprecise measurements [18], multiple geographically distant data sources for information integration based on fuzzy similarity scores [20]. Recently, Li et al. [1] has studied top-k queries in a communication-efficient manner on probabilistic data from multiple, distributed sites. On the other hand, as another important research direction in such a context, skyline queries, which focusing attention on the most preferable records, have not been studied before. It is not surprise that study distributed skyline queries on probabilistic data will quickly attract a lot of interests.

Distributed skyline computation over uncertain data has many important applications. For instance, consider the stock market application where customers may want to select good deals (transactions) for a particular stock over all the distributed stock exchange centers. A deal is recorded by two attributes (price, volume) where price is the average price per share in the deal and volume is the number of shares. In such scenario, before making trade decisions, a customer may want to know the top deals over

• *The authors are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: {xfding, hjin}@hust.edu.cn.*

Manuscript received 9 Sept. 2010; revised 23 Dec. 2010; accepted 6 Mar. 2011; published online 18 Mar. 2011.
Recommended for acceptance by Q. Ling.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2010-09-0493.
Digital Object Identifier no. 10.1109/TKDE.2011.77.

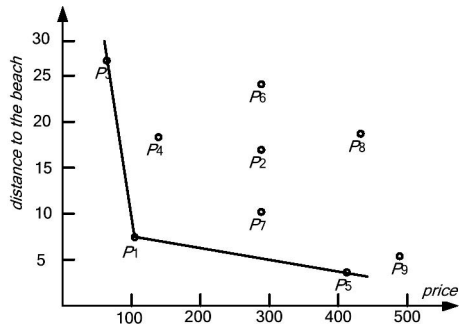

Fig. 1. A general Skyline example.

all the distributed local sites. Generally speaking, deal a is better than another deal b if a involves a higher volume and is cheaper than those of b. Nevertheless, recording errors caused by systems or human beings may make failed deals be recorded successful, and vise versa; consequently each deal recorded in the database has a probability to be true. Therefore, a set of deals recorded in the database may be treated as a set of uncertain elements and some customers may only want to know "top" deals (skyline) among the entire deals over distributed sites; and thus we have to take the uncertainty of each deal into consideration. This is a scenario of distributed skyline queries over uncertain data.

We observe that uncertainty and distributed skyline query processing have been each studied quite extensively but separately [8], [9], [12], despite the fact that the two often arise concurrently in many applications. In this paper, we study the problem of distributed skyline queries over uncertain data, and propose computation- and communication-efficient processing algorithms.

As an overview, we propose an algorithm, called *distributed skyline over uncertain data* (DSUD), which computes the global skylines in a distributed, uncertain environment with economical bandwidth cost. Furthermore, an enhance version of DSUD algorithm (e-DSUD) is also proposed to further speedup the query efficiency and reduce the communication cost. The efficiency of e-DSUD stems from a highly optimized feedback mechanism, where the central server transmits to each local site precious information that prevents the delivery of a large number of nonskyline points. Besides low-bandwidth consumption, both DSUD and e-DSUD algorithms achieve excellent progressiveness in outputting the final skyline points.

Our contributions. We study skyline queries for distributed probabilistic data. We design communication efficient algorithms for retrieving the skyline tuples with the global skyline probability larger than a threshold from distributed sites, with computation overhead also as a major consideration. In summary, our contributions are as follows:

- In this paper, we give the formal definition of distributed skyline queries, in the context of uncertain data.
- We study a general framework to answer the DSUD query, and propose effective and efficient feedback mechanisms, which can be seamlessly integrated into our query procedure, to help reduce the communication cost and find the most promising candidates for the skyline query.
- We give the formal cost analysis of the feedback mechanism used in our processing framework, using the expect number of skyline tuples and the total number of tuples in the feedback.
- We conduct experiments under different parameter settings to verify the effectiveness and efficiency of our proposed feedback mechanisms as well as the progressiveness of the corresponding query processing algorithms. The results show that our algorithms are efficient, robust and progressive.

This paper extends an earlier published conference paper [32] in several substantial ways. First, we provide more details about the background and the related work. Second, based on the general framework we proposed, the drawback of choosing inappropriate feedbacks in the query processing procedure is analyzed, which makes the feedback mechanism in our method more reliable. Third, we add how to continuously maintain the skyline results when data updates happen at local sites, and the scenario of distributed retrieval of specified subspace skyline is also discussed. Fourth, some optimization techniques are integrated into the implementation issues to achieve better performance. Finally, we reconsider the experimental settings and conduct extensive experiments to evaluate the performance of our proposed algorithms.

The rest of the paper is organized as follows: Section 2 reviews the previous work that is directly related to ours. Section 3 formally defines the problem and introduces the baseline approach, and Section 4 explains the general framework for efficient distributed skyline query processing. Section 5 presents the detail DSUD algorithm and its enhanced version, which adopts an efficient feedback mechanism; in addition, the update strategy is discussed to maintain the skyline. Section 6 introduces the implementation issue, and Section 7 evaluates the proposed algorithms with experiments. Section 8 concludes the paper with directions for future work.

2 Related Work

Section 2.1 first reviews previous distributed skyline query processing techniques over precise data. Then, the related work of probabilistic skyline queries is introduced in Section 2.2.

2.1 Distributed Skyline Queries

In the literature, a number of interesting methods have been made to address distributed skyline retrieval [20], [22], [23], [24], [25]. Balke et al. [23] proposed the first distributed skyline algorithm, by considering the underlying relation is vertically partitioned between local severs, i.e., each server keeps only an attribute of the relation. In particular, it assumes that a d-dimensional database is vertically partitioned into d lists, where each list contains the values of a dimension in ascending order. Based on the rationale of the Threshold Algorithm (TA), their algorithm performs sorted accesses over the d lists in a round-robin manner, until a point p comes up in all of them. Up to now, those points unseen in any list can be pruned, as they must be dominated by p, thus cannot be in the final skyline set. In the contrast, Wu et al. [25] assumed a constrained

Tuples	Value	Probability
t_1	$(v_{11}, v_{12}, \ldots, v_{1d})$	$P(t_1)$
t_2	$(v_{21}, v_{22}, \ldots, v_{2d})$	$P(t_2)$
t_3	$(v_{31}, v_{32}, \ldots, v_{3d})$	$P(t_3)$
$\vdots$	$\vdots$	$\vdots$
t_N	$(v_{N1}, v_{N2}, \ldots, v_{Nd})$	$P(t_N)$

Fig. 2. The uncertainty data model.

Tuples	Value	Probability
t_1	(80, 96)	0.8
t_2	(85, 90)	0.6
t_3	(75, 95)	0.8

World W	Probability P(W)
$W_1 = \{\emptyset\}$	0.2×0.4×0.2 = 0.016
$W_2 = \{t_1\}$	0.8×0.4×0.2 = 0.064
$W_3 = \{t_2\}$	0.2×0.6×0.2 = 0.024
$W_4 = \{t_3\}$	0.2×0.4×0.8 = 0.064
$W_5 = \{t_1,t_2\}$	0.8×0.6×0.2 = 0.096
$W_6 = \{t_1,t_3\}$	0.8×0.4×0.8 = 0.256
$W_7 = \{t_2,t_3\}$	0.2×0.6×0.8 = 0.096
$W_8 = \{t_1,t_2,t_3\}$	0.8×0.6×0.8 = 0.384

Fig. 3. An example of possible worlds.

horizontal partitioning determined by P2P overlay networks CAN [27]. Specifically, each local server is mapped to a rectangular region contained in the data space and is responsible for managing all and only the data points that fall into its scope. The constrained skyline retrieval they proposed aims at returning the skyline of the data points falling in a query range.

Recently, Both Wang et al. [14] and Cui et al. [33] developed efficient algorithms for retrieving the skyline in P2P networks BATON [28]. Similar to the work in [25], the algorithm demands that the data of the peers are organized using an overlay network. In contrast, Chen et al. [34] studied constrained skyline computation over distributed data sites without any overlay structures. They proposed a specific partition algorithm that divides all relevant sites into incomparable groups, such that a given query can be executed in parallel among all those groups. Vlachou et al. [24] discussed distributed skyline computation in various subspaces. Their algorithm can be used to fetch skylines of the full space, but it incurs significantly higher network bandwidth consumptions. Finally, Zhu et al. [22] studied the distributed skyline query when the underlying data set is horizontally partitioned onto a set of local servers. Their algorithm can solve the problem with low-bandwidth consumption. However, none of the above work can deal with the skyline queries over distributed data with uncertainty.

2.2 Probabilistic Skyline Queries

In the context of uncertain database, the probabilistic skyline query processing on uncertain data is first studied by Pei et al. [26]. Specifically, they define the probabilistic skyline query where each object contains discrete random instances. Effective pruning rules have been proposed to facilitate the search process, and bounding-pruning-refining processing framework is developed for an efficient probabilistic skyline computation. Furthermore, based on the reverse skyline semantics and the uncertain data model, Lian and Chen [6] studied an efficient and accurate query processing of reverse skyline query over uncertain data, considering both monochromatic and bichromatic cases.

Recently, Zhang et al. [2] investigated the problem of efficiently computing skyline against sliding windows over an uncertain data stream regarding given probability thresholds. They present a framework based on efficiently maintaining a candidate set. They also show that such a candidate set is the minimum information that needs to be kept to answer the skyline query. Furthermore, efficient techniques have been presented to process continuous queries. The above algorithms, however, are not applicable in a distributed environment where the network bandwidth has to be considered.

3 Preliminaries

In this section, we first introduce the uncertainty data model and its corresponding possible world semantics. Then, we define the problem of probabilistic skyline retrieval over distributed uncertain data and clarify the underlying assumptions. Finally, we describe a baseline approach and illustrate its deficiencies, thus strengthening the motivation of techniques proposed in Section 4.

Uncertainty data model and possible world semantics. We use D to denote an uncertain database, and D is composed of a set of N tuples t_i $(1 \leq \mathrm{i} \leq N)$. Each tuple t_i has a probability $P(t_i)$ $(0 < P(t_i) \leq 1)$ to occur, and v_{ij} $(1 \leq j \leq d)$ denotes the jth dimension value. The uncertainty data model is depicted in Fig. 2. Furthermore, based on this uncertainty data model, a possible world W is instantiated by taking a set of tuples from the uncertain relation, that is, each tuple selects its existential states independent of other tuples. Generally speaking, the probability of W appears is

$$P(W) = \prod_{t \in W} P(t) \times \prod_{t \notin W} (1 - P(t)). \quad (1)$$

If we denote the set of all possible worlds be Ω, then we always have $\sum_{W \in \Omega} P(W) = 1$. The example in Fig. 3 illustrates the possible worlds for the uncertain database in two dimensional spaces based on the uncertainty data model described in Fig. 2.

3.1 Problem Definition

Traditionally, for two tuples t and s, we say t dominates s, denoted by $t \prec s$, if the values of t are not larger than that of s in all dimensions, and there exists at least one dimension where the value of t is smaller than s. Given a set of tuples, the skyline is composed of those tuples which are not dominated by any other tuples.

In the context of uncertain data, the skyline definition is quite different from the conventional one, that is to say, instead of clearly assigning a tuple with YES or NO to be in the skyline, whether a tuple t is in the skyline set or not is measured by its skyline probability $P_{sky}(t, D)$. For a possible world W, let *sky* (W) be the set of elements in W that compose the skyline of W, then the skyline probability $P_{sky}(t, D)$ of a tuple t equals to the summation over all existential probabilities of those possible worlds where t is contained in their skyline set, that is

$$P_{sky}(t, D) = \sum\nolimits_{t \in sky(w), w \in \Omega} P(W). \quad (2)$$

For example in Fig. 2, the skyline probability for t_1 is $P_{sky}(t_1) = P(W_2) + P(W_5) = 0.064 + 0.096 = 0.16$. Similarly, $P_{sky}(t_2) = P(W_3) + P(W_5) + P(W_7) + P(W_8) = 0.6$, and $P_{sky}(t_3) = P(W_4) + P(W_6) + P(W_7) + P(W_8) = 0.8$. Furthermore, for an uncertain tuple t with existential probability $P(t)$, the probability that t appears in uncertain database D equals to $P(t)$. Considering another uncertain tuple t' with existential probability $P(t')$, if t' dominates t, then the probability that t is dominated by t' equals to $P(t')$. Thus, the probability that t is not dominated by any other tuples is $\prod_{t' \in D, t' \prec t}(1 - P(t'))$. We conclude that the skyline probability of tuple t in (2) can be rewritten as

$$P_{sky}(t, D) = \prod_{t' \in D, t' \prec t} (1 - P(t')) \times P(t). \tag{3}$$

Based on the above definition, the problem of distributed skyline retrieval over uncertain data (DSUD) can be defined as follows:

Definition 1 (Distributed Skyline in Uncertain Data). *Given a set of m distributed sites $S = \{s_1, s_2, \ldots, s_m\}$, each possessing an uncertain database $D_i (1 \leq i \leq m)$ with size n_i, and a centralized server H where the query is processed. A distributed skyline over uncertain data would like to report at H those tuples with their global skyline probabilities not smaller than a given threshold q $(0 < q \leq 1)$. Based on the semantics of skyline definition over a centralized database and the independence of those local databases D_i, the global skyline probability of tuple t, denoted as $P_{g\text{-}sky}(t)$, is defined as follows:*

$$P_{g\text{-}sky}(t) = P(t) \times \prod_{t' \in D_1, t' \prec t} (1 - P(t')) \times \prod_{t' \in D_2, t' \prec t} (1 - P(t')) \times \cdots \times \prod_{t' \in D_m, t' \prec t} (1 - P(t')). \tag{4}$$

If we unify all the distributed databases D_i into one database D, that is $D = D_1 \cup D_2 \cup \cdots \cup D_m$, then (4) can be rewritten as

$$P_{g\text{-}sky}(t) = \prod_{t' \in D, t' \prec t} (1 - P(t')) \times P(t). \tag{5}$$

Note that, without loss of general situations as studied for many problems over distributed data, a critical requirement in distributed computation is to have low-bandwidth consumption, thus, in this paper our main objective is to minimize the total communication cost in computing the global skyline result. For simplicity, each tuple in the unified database D is assumed to be unique, thus we will not consider the situation where the local databases have overlapping tuples. The commonly used symbols are summarized in Table 1.

TABLE 1
Frequently Used Symbols

Symbol	Meaning
H	The central server
m	The number of local sites
S_i	The i-th local site
D_i	The i-th uncertain database
D	The global uncertain database
$P_{g\text{-}sky}(t)$	The global skyline probability of tuple t
$SKY(H)$	The set of global skyline tuples obtained by H
$SKY(D_i)$	The set of local skyline tuples from D_i

3.2 The Baseline Approach

A straightforward solution for our problem can always ask all local sites to transmit their uncertain databases to the server H, and then process the skyline query at H locally. Suppose we have a centralized uncertain database $D = \{t_1, t_2, \ldots, t_N\}$ at server H, according to the calculation semantic in (3), a brute-force algorithm needs $O(N)$ time to compute the skyline probability of a tuple t_i, and $O(N^2)$ time to compute all tuple's skyline probabilities and select these qualified tuples.

However, the baseline approach is both communication- and computation-expensive, the total communication cost is $|D| = \sum_{i=1}^{m} |D_i|$. As mentioned in [22], a good distributed skyline algorithm should achieve the following goals at least:

1. **Minimization of bandwidth consumption.** We measure bandwidth in the total number of tuples transmitted over the network. Actually speaking, extra bandwidth is needed for sending synchronizing messages and the packet headers in order to enforce the used network protocol. Since these messages and headers are very small compared to transmitted tuples, they are not considered when measuring the bandwidth consumption.
2. **Progressiveness.** Since the skyline query is more complex than the range or top-k queries, the entire retrieval time may be very long, especially in the distributed environment where network delay has to be considered, thus a good distributed skyline algorithm should satisfy the progressiveness property, that is, report some early discovered skyline results soon after the query beginning, and produce the remaining results in a great measure before the end of query execution.

The baseline approach is clearly very expensive and hard to achieve the above goals, thus we have an even more urgent need for effective, efficient, and progressive processing techniques for skyline queries in the distributed uncertain environment, which are the main contributions of this paper.

4 The General Framework

A general strategy in distributed query processing is to first answer the query within each local site individually, and then combine the results to get the final answer set. In this section, we present a general framework for processing skyline queries following this strategy. Here, we argue that if the users want to retrieve the subspace skyline set, our framework can be easily extend to any prespecified subset attributes of size k $(k \leq d)$ among all attributes, that is, simply by checking the dominant relation between tuples only on the user specified k dimensions.

Fig. 4 illustrates the general framework for answering distributed skyline queries over uncertain data. In particular,

```
Procedure DSUD_framework
Input : A number of m local uncertain databases D_i, a threshold q
Output : a set SKY(H) containing those qualified skyline tuples
BEGIN
(1) compute the local skyline SKY(D_i) over D_i; //local computing phase
(2) sort each SKY(D_i) in descending order of local skyline probability;
(3) while SKY(D_i) is not empty  // iteration
(4)   each site sends its representative tuple in SKY(D_i) to H;
(5)   compute each tuple's local skyline probability within H;
(6)   select representative tuples from H and deliver them to local sites;
(7)   obtain the global skyline probability, and add to SKY(H);
(8)   prune unqualified tuples in local site;
(9) end while;
END
```

Fig. 4. The framework for DSUD processing.

at the beginning of DSUD, each participant S_i computes its local skyline using a centralized skyline algorithm (the detailed implementation is introduced in Section 6) and sorts the skyline points in descending order of their skyline probabilities. At the same time, the centralized coordinator H initializes an empty set $SKY(H)$, where all the global skyline tuples will be contained eventually. Then, the rest of the algorithm is carried out in iterations, until all of local skyline sets $SKY(D_i)$ $(1 \leq i \leq m)$ have become empty or the largest local skyline probability of any tuple within D_i is smaller than the user specified threshold q. Note that, each iteration consists of four phases:

1. **To-Server Phase.** In the initial phase, each local site S_i selects its representative tuple in $SKY(D_i)$ with the largest skyline probability to the coordinator H. After completing the first iteration, which site should be selected again to transfer its representative tuple depends on the results of the Server-Delivery phase.
2. **Server-Calculation Phase.** In this phase, H unions all the representative skylines from different sites into a small data set D_0, and retrieves the skyline set $SKY(D_0)$ of D_0 again using a centralized skyline algorithm. Note that, after all the skyline points are selected from D_0, they are arranged in descending order of their partial skyline probabilities over all data in D_0.
3. **Server-Delivery Phase.** Central server H selects a tuple from $SKY(D_0)$ with the largest partial skyline probability and broadcasts it to the other local sites (except the site where the tuple comes from) to obtain its global skyline probability. Note that, suppose that H broadcasts tuple t_{ij} belonging to site S_i, then the representative tuple will be selected in S_i in the next To-Server phase.
4. **Local-Pruning Phase.** When the candidate tuple from sever H is delivered to the local sites, a local pruning method is invoked to prune all those unqualified skyline tuples from $SKY(D_i)$. It can be verified that the pruned skyline is guaranteed not to be the global skyline of the global data set D.

Note that, at the end of an iteration, the tuple selected from $SKY(D_0)$ is definitely one of the most probable final skyline points that have been sent to H in all the iterations so far, this is guaranteed by the centralized skyline computation algorithm used in both the local sites and the central server. Therefore, every global skyline tuple must have been transmitted to H before dissemination to get its global skyline probability. Obviously, to guarantee the progressiveness of our approach, those qualified skyline tuples after each iteration will be reported as part of the final skyline results on the fly.

Moreover, the efficiency of our processing framework relies on the Server-Delivery phase where feedback tuples are used to get the global skyline results and prune those local skyline points. Indeed, the feedback from central server has no influence on the algorithm correctness, however, as analyzed below they affect the bandwidth consumption significantly.

Given the database cardinality N, and made three assumptions: 1) tuples are uniformly distributed in each dimension, and there are no two tuples share the same value along any dimension; 2) all dimensions are reciprocally independent; 3) all tuples appear with probabilities uniformly distributed in [0,1]. Based on the skyline cardinality estimation method for precise data proposed in [22] and [35], we have the following equation for the expect number of skyline tuples in a d-dimensional uncertain database with cardinality N:

$$H(d,N) \approx \sum_{n=0}^{N} \frac{\ln^{d-1} n}{d!} \times P(n). \tag{6}$$

Here, $P(n)$ denotes the probability of having exactly n tuples. This is because the distribution of the tuples remains uniform in the d-dimensional space, no mater how many tuples truly show up.

Since the central coordinator H needs to send each skyline tuple back to $m-1$ local sites, thus the total number of tuples have to feedback is at least

$$N_{back} = (m-1) \times H(d,N) \approx (m-1) \times \sum_{n=0}^{N} \frac{\ln^{d-1} n}{d!} \times P(n). \tag{7}$$

Furthermore, assume each local database stores the same number of tuples, we can obtain that the local skyline cardinality of each participant equals $H(d,\ N/m)$ tuples. Thus, the total number of local skyline tuples over all local sites is expected to be

$$N_{local} = (m-1) \times H(d,N/m) \approx (m-1) \times \sum_{n=0}^{N/m} \frac{\ln^{d-1} n}{d!} \times P(n). \tag{8}$$

We can see that, N_{back} is larger than N_{local} when $m > 1$, that is to say, the feedback mechanism costs even higher bandwidth than simply transmit all local skyline tuples to central server H.

Thus, the feedback to local sites should be chosen in a manner that it can be used to prune as many local skyline points as possible, instead of those tuples that are not

dominating any other data points, this observation motivates the core feedback mechanism used in our algorithms.

In the following section, we prove the correctness of restricting the solution to this framework and show the detail of DSUD algorithm, as well as its enhanced version e-DSUD.

5 The Algorithms

Based on the data model and the general processing framework given above, the critical components of the DSUD algorithm are first studied in detail in Section 5.1. Then, we show an enhanced version of DSUD processing algorithm in Section 5.2. The objective of our algorithms is to minimize the number of transmitted tuples and report the skyline results progressively. Finally, Section 5.3 demonstrates the e-DSUD algorithm with a concrete example.

5.1 Details about DSUD Algorithm

Although the baseline approach is correct, it violates all the performance targets as described above. An important improvement to this approach is as follows: First, each local database computes its local skyline and sends them to the central server H. Then, H unions all these skylines into another data set, and broadcasts every tuples within this data set to all the other local sites to obtain the global skyline probability. However, the union of all the local skylines may have a very large size, which still incurs expensive bandwidth consumption. In fact, it is easy to observe that not all the local skyline tuples are eventually in the final global skyline set, and thus would not need to be transferred to the server H. This observation indicates that an optimistic algorithm should be able to transfer these unqualified tuples as less as possible, which makes an important rationale behind the DSUD algorithm.

Another important observation is that the global skyline probability of any tuple t_{ij} could be calculated accumulatively from all the local sites. In particularly, we have the following Observation and Lemma:

Observation 1. For any tuple t_{ij} from uncertain database D_i at the local site S_i, the local skyline probability of t_{ij} to another uncertain database D_x at local site $S_x (x \neq i)$ can be defined as follows:

$$P_{sky}(t_{ij}, D_x) = \prod_{t' \in D_x, t' \prec t_{ij}} (1 - P(t')). \quad (9)$$

Note that, since tuple t_{ij} is not belonging to database D_x, thus its local skyline probability in D_x is not affected by its existential probability $P(t_{ij})$.

Based on the above observation, we have the following lemma to calculate the global skyline probability of t_{ij} from database D_i.

Lemma 1. *Given a number m of an uncertain databases $D_i (1 \leq i \leq m)$ with size n_i, the global skyline probability of t_{ij} from D_i, denoted as $P_{g\text{-}sky}(t_{ij})$, equals to*

$$P_{g\text{-}sky}(t_{ij}) = \prod_{x=1}^{m} P_{sky}(t_{ij}, D_x). \quad (10)$$

Proof. The proof of this lemma is easy, according to (5), we have

$$\begin{aligned} P_{g\text{-}sky}(t_{ij}) &= P(t_{ij}) \times \prod_{t' \in D_1, t' \prec t_{ij}} (1 - P(t')) \\ &\quad \times \prod_{t' \in D_2, t' \prec t_{ij}} (1 - P(t')) \times \cdots \times \prod_{t' \in D_m, t' \prec t_{ij}} (1 - P(t')) \\ &= P(t_{ij}) \times \prod_{t' \in D_i, t' \prec t_{ij}} (1 - P(t')) \\ &\quad \times \prod_{s=1, s \neq i}^{m} t' \in D_s, t' \prec t_{ij} (1 - P(t')) \\ &= P_{sky}(t_{ij}, D_i) \times \prod_{s=1, s \neq i}^{m} P_{sky}(t_{ij}, D_s)) \ //\text{by (3) and} \quad (9) \\ &= \prod_{i=1}^{m} P_{sky}(t_{ij}, D_i). \end{aligned}$$

□

It is easy to observe that by sending only the most promising tuple t_{ij} from site S_i to the other local sites, the global skyline probability of t_{ij} can be calculated accumulatively. In order to find the most promising tuples from each locate site, our proposed DSUD algorithm sorts all the tuples at site S_i based on their local skyline probabilities $P_{sky}(t_{ij}, D_i)$. Without loss of generality, we assume $P_{sky}(t_{i1}, D_i) \geq P_{sky}(t_{i2}, D_i) \geq \cdots \geq P_{sky}(t_{in}, D_i)$ for any local site s_i. Thus, these m local sites S_i transmit their most promising tuples to the central server H in descending order of their local skyline probabilities. More precisely, a priority queue L of size m is maintained at the server H, and each site S_i sends its representative tuple with its id, existential probability and $P_{sky}(t_{ij}, D_i)$ that corresponds to the local skyline probability, i.e., a quaternion $<i, j, P(t_{ij}), P_{sky}(t_{ij}, D_i)>$ to L. The quaternion in the priority queue is sorted by the local skyline probability in descending order. Clearly, the priority queue L is initialized by retrieving the first tuple's id, existential probability, and local skyline probability from each local site S_i.

In the next sever-calculation step, H retrieves the first element from priority queue L, i.e., $<i, j, P(t_{ij}), P_{sky}(t_{ij}, D_i)>$, which is considered as one of the most competing global skyline tuples. In order to compute the exact global skyline probability of t_{ij} that H has just obtained, H broadcasts t_{ij} to all the other local sites except S_i and asks each site D_x to report back the value $P_{sky}(t_{ij}, D_x)$ (based on (9)). By Lemma 1, H obtains the exact global skyline probability $P_{g\text{-}sky}(t_{ij})$ of tuple t_{ij} according to the returned values. Meanwhile, the feedback t_{ij} from server H will be used to prune these unqualified skyline tuples from $SKY(D_x)$. Then, after tuple t_{ij} is fetched from L, H asks for another tuple $t_{i(j+1)}$ from site S_i, along with its id, existential probability, and the local skyline probability $P_{sky}(t_{ij}, D_i)$ which consists a quaternion $<i, j+1, P(t_{i(j+1)}), P_{sky}(t_{i(j+1)}, D_i)>$ to be inserted into the priority queue L. Up to here, one iteration is completed.

Based on Lemma 1, we have the following corollary which could be used to guarantee the iteration halt timely and correctly.

Corollary 1. *The global skyline probability $P_{g\text{-}sky}(t_{ij})$ of a tuple is upper bounded by its local skyline probability $P_{sky}(t_{ij}, D_i)$, that is to say: $P_{g\text{-}sky}(t_{ij}) \leq P_{sky}(t_{ij}, D_i)$.*

Note that, the local skyline probability of any unfetched tuples by H from all local sites is upper bounded by the head element from the priority queue L. Thus, it is easy to induce that, the global skyline probability of any unfetched tuples in all the local sites is upper bounded by the local skyline probability of the first element within L. Let $P_{sky}(t_{ij}, D_i)$ be the local skyline probability of the first element in L. It is safe for the iteration to terminate if $P_{sky}(t_{ij}, D_i) < q$ at a certain round. There is no missing elements in L when determine the final skyline tuples. We denote this algorithm as DSUD.

Up to now, we have introduced the DSUD algorithm. Clearly, the efficiency of DSUD algorithm is affected by the feedback from central server H. In particular, as the feedback tuples can be used to prune those unqualified skyline tuples in $SKY(D_i)$, they would reduce the to-server phase cost, thus affect the bandwidth consumption significantly. In the following, we introduce an enhanced DSUD algorithm, which selects the feedback in an effective way and can make the iteration terminate early.

5.2 The Enhanced DSUD

In this section, we introduce an enhanced DSUD algorithm, which adopts a different feedback mechanism from DSUD algorithm. In general, the sorted access on local skyline probability in the DSUD algorithm has limited pruning power as it simply relies on the next tuple's existential probability from each site to estimate the upper global skyline probability bound of any unseen tuple. However, compared to the global skyline probability, the local skyline probability of a tuple within server H is usually too large to be used for estimation appropriately, thus a more precise probability should be used to tight the bound. In the sequel, we introduce the core problem of feedback selection within central server H, which incurs much less feedbacks and communication cost than the basic local skyline probability ranking approach.

Rationale. Given a set of local skyline tuples discovered at central server H so far, the best selection criterion is to select one tuple with the most powerful dominant ability as the feedback for every participant S_i. In particular, the criterion adopts the following principle: a tuple t in server H is currently the most promising feedback to local site S_i, only if it has the largest global skyline probability upper bound, that is to say, tuple t can prevent local sites from transmitting more tuples to server H. Note that, if no feedback was used, each local site S_i would need to transfer the entire local skyline set $SKY(D_i)$ to central server H. In order to select the most dominant tuple from priority queue L, H must obtain some knowledge about the global skyline probabilities of these tuples gathered in L or the content of local skyline set $SKY(S_i)$. Otherwise, H cannot figure out how many tuples would be pruned by a feedback tuple. A naïve approach for H to obtain such information would be to ask all the local sites S_i to transfer some *data synopses* which retaining the key statistical traints of the original data distribution over uncertain database D_i. However, this approach itself is communication expensive for the transmitting of such *data synopses* may occupy too much network bandwidth. Thus, a better solution is to calculate the approximation of the global skyline probability of the tuple within L without further bandwidth consumption. We have the following observation:

Observation 2. For tuple t from uncertain database D_i at local site S_i, and another tuple s from uncertain database D_j at site S_j. Suppose both of these two tuples are fetched into priority queue L, and tuple s is dominated by t, then the upper bound of the local skyline probability of s to uncertain database D_i at local site S_i can be defined as follows:

$$\begin{aligned} P_{sky}(s, D_i) &= \prod_{t' \in D_i, t' \prec t} (1 - P(t')) \times (1 - P(t)) \\ &\quad \times \prod_{s' \in D_i, s' \prec s, s' \not\prec t} (1 - P(s')) \\ &= \frac{P_{sky}(t, D_i)}{P(t)} \times (1 - P(t)) \times \prod_{s' \in D_i, s' \prec s, s' \not\prec t} (1 - P(s')) \\ &\leq \frac{P_{sky}(t, D_i)}{P(t)} \times (1 - P(t)). \end{aligned} \tag{11}$$

From the above observation, we can conclude that the upper bound of the local skyline probability of tuple s to another uncertain database D_i would be easily obtained by only using the information in priority queue L. Furthermore, the following corollary can be used to deduce a more precise upper bound of the global skyline probability of a tuple s within the priority queue L.

Corollary 2. *The global skyline probability $P_{g\text{-}sky}(s)$ of a tuple s from site S_j is upper bounded by its local skyline probability $P_{sky}(s, D_j)$ multiplies the upper bound of its local skyline probabilities to other uncertain databases D_x, from where its representative tuple t transferred to priority queue L is in dominant of tuple s, that is to say*

$$\begin{aligned} P_{g\text{-}sky}(s) &= \prod_{x=1}^{m} P_{sky}(s, D_x) \\ &= P_{sky}(s, D_j) \times \prod_{x=1, x \neq j}^{m} P_{sky}(s, D_x) \\ &\leq P_{sky}(s, D_j) \times \prod_{x=1, x \neq j, t \in D_x, t \in L, t \prec s}^{m} P_{sky}(s, D_x) \\ &\leq P_{sky}(s, D_j) \times \prod_{x=1, x \neq j, t \in D_x, t \in L, t \prec s}^{m} \frac{P_{sky}(t, D_x)}{P(t)} \\ &\quad \times (1 - P(t)) \\ &= P_{g\text{-}sky}(s)^*. \end{aligned} \tag{12}$$

The proof of this corollary is obvious, we will not show the details here.

The approximate value of the global skyline probability $P_{g\text{-}sky}(s)$, denoted as $P_{g\text{-}sky}(s)^*$, can be obtained without more bandwidth consumption. Up to now, we are ready to elaborate how to choose optimal feedback from priority queue L to all the local sites. For this purpose, the

coordinator H needs to compute $P_{g\text{-}sky}(s)^*$ of those tuples obtained in priority queue L so far and then choose the tuple with largest approximate value as the feedback. In particular, these tuples with its id, existential probability and the approximate value, i.e., a quaternion $<i, j, P(t_{ij}), P_{g\text{-}sky}(t_{ij})^*>$ will be inserted into another priority queue G sorted in descending order of their global skyline approximate values. Then, H selects the first element in the sorted queue and broadcasts it to other local sites to obtain the exact global skyline probability $P_{g\text{-}sky}(t_{ij})$. If $P_{g\text{-}sky}(t_{ij})$ is not smaller than threshold q, then t_{ij} will be added to the global skyline set $SKY(H)$. In the sequel, the next tuple from $SKY(D_i)$ at site S_i will be sent to coordinator H, and then the next iteration starts.

Note that, if the approximate value $P_{g\text{-}sky}(s)^*$ of a global skyline probability $P_{g\text{-}sky}(s)$ is smaller than threshold q, then tuple s will be expunged from the system immediately, and a tuple from the same local source in $SKY(D_i)$ will be retrieved to coordinator H in the next round of iteration.

5.3 An Illustrative Example

In this section, we illustrate our proposed e-DSUD algorithm with a concrete example. Suppose a hotel booking system that has three local sites: S_1 in Qingdao, S_2 in Shanghai, and S_3 in Xiamen. In particular, two dimensional attributes, price of the hotel and distance to the beach, of a tuple are contained in each local database at the local site. Besides, each tuple is also associated with a confidence (existential) probability. For example, a triple $<340, 66, 0.8>$ means the price of a hotel is 340, its distance to the beach is 66, and its confidence probability is 0.8. Now, a customer wants to query the skyline set over these three cities, and quality guarantee threshold q is set to 0.3. The query will be processed as follows:

At the beginning of e-DSUD, each local site computes its own local skyline set, and put all these qualified skyline tuples in $SKY(D_i)$. Table 2a illustrates these skyline tuples at each site $D_i (1 \le i \le 3)$, sorted in descending order of their skyline probabilities. More precisely, the quaternion $<3, 8, 0.8, 0.5>$ means the existential probability of tuple $<3, 8>$ is 0.8, and its local skyline probability within uncertain database D_1 is 0.5.

In the To-Server phase, central server H is initialized with each local site's representative tuple at the first iteration, that is, each participant S_i sends its first tuple in $SKY(D_i)$ to server H. In particular, H retrieves (6, 6, 0.7, 0.65) from $SKY(D_1)$, (6.5, 7, 0.8, 0.65) from $SKY(D_2)$, and (6.4, 7.5, 0.9, 0.8) from $SKY(D_3)$. These items are stored according to their local skyline probabilities in a priority queue L at server H, shown in Table 2b.

In order to select the most dominant tuple from L, in the server-computation phase, we use (9) to calculate the approximate values of the global skyline probabilities for these tuples in L. In particular, since tuple (6.4, 7.5) is dominated by tuple (6, 6), its approximate value equals to

$$\begin{aligned} P_{g\text{-}sky}((6.4, 7.5))^* &= P_{sky}((6.4, 7.5), D3) \\ &\quad \times P_{sky}((6, 6), D_1)/P(6, 6) \times (1 - P(6, 6)) \\ &= 0.8 \times (0.65/0.7) \times 0.3 = 0.22, \end{aligned}$$

TABLE 2
Illustration of E-DUSD Algorithm

Set	Content
$SKY(D_1)$	(6, 6, 0.7, 0.65), (8, 4, 0.8, 0.6), (3, 8, 0.8, 0.5)
$SKY(D_2)$	(6.5, 7, 0.8, 0.65), (4, 9, 0.6, 0.6), (9, 5, 0.7, 0.6)
$SKY(D_3)$	(6.4, 7.5, 0.9, 0.8), (3.5, 11, 0.7, 0.7), (10, 4.5, 0.7, 0.7)

(a)

Server	Content
L	(6.4, 7.5, 0.9, 0.8), **(6, 6, 0.7, 0.65)**, (6.5, 7, 0.8, 0.65)

(b)

Set	Content
$SKY(D_1)$	(8, 4, 0.8, 0.6), (3, 8, 0.8, 0.5)
$SKY(D_2)$	(4, 9, 0.6, 0.6), (9, 5, 0.7, 0.6)
$SKY(D_3)$	(3.5, 11, 0.7, 0.7), (10, 4.5, 0.7, 0.7)

(c)

Server	Content
L	**(8, 4, 0.8, 0.6)** , (6.4, 7.5, 0.9, 0.22), (6.5, 7, 0.8, 0.18)

(d)

Set	Content
$SKY(D_1)$	(3, 8, 0.8, 0.5)
$SKY(D_2)$	(4, 9, 0.6, 0.6)
$SKY(D_3)$	(3.5, 11, 0.7, 0.7)

(e)

Server	Content
L	**(3, 8, 0.8, 0.5)**, (6.4, 7.5, 0.9, 0.24), (6.5, 7, 0.8, 0.195)

(f)

Set	Content
$SKY(D_1)$	NULL
$SKY(D_2)$	NULL
$SKY(D_3)$	NULL

(g)

Server	Content
L	(6.4, 7.5, 0.9, 0.24), (6.5, 7, 0.8, 0.195)

(h)

similarly,

$$\begin{aligned} P_{g\text{-}sky}((6.5, 7))^* &= P_{sky}((6.5, 7), D_2) \times P_{sky}((6, 6), D_1)/P(6, 6) \\ &\quad \times (1 - P(6, 6)) = 0.18. \end{aligned}$$

Note that, tuple (6, 6) is a skyline tuple within L, thus its approximate probability $P_{g\text{-}sky}((6, 6))^*$ remains the same as $P_{sky}((6, 6), D_1) = 0.65$. Consequently, in the following feedback phase, coordinator H selects tuple (6, 6) which has the largest skyline value and sends it to the other two local sites D_2 and D_3, where the local skyline probability of tuple (6, 6) is obtained to get its final global skyline probability (7). Suppose the global skyline probability $P_{g\text{-}sky}((6, 6))$ is larger than threshold 0.3, then (6, 6) will be added to $SKY(H)$. Meanwhile, in the local-pruning phase, D_2 removes (6.5, 7, 0.8, 0.65) from $SKY(D_2)$, and D_3 removes (6.4, 7.5, 0.9, 0.8)

from $SKY(D_3)$, since they are dominated by (6, 6). At the end of the first iteration, the updated list of active local skyline tuples at each local site is shown in Table 2c.

In the sequel, the second iteration starts. Different from the first iteration, only local site D_1 transmits tuple (8, 4, 0.8, 0.6) in its current local skyline list $SKY(D_1)$ to H. Table 2d demonstrates the tuples gathered by server H so far. The server computation phase discovers a new skyline tuple (8, 4, 0.8, 0.6), which has the largest approximate skyline value and is returned as the feedback to other two local sites D_2 and D_3. For the pruning power of feedback information, tuple (8, 4) will filter (9, 5, 0.7, 0.6) out of $SKY(D_2)$, and (10, 4.5, 0.7, 0.7) out of $SKY(D_3)$. Table 2e presents the current active local skyline tuples within each local site. Again, suppose $P_{g\text{-}sky}((8,4))$ is larger than threshold 0.3, then (8, 4) will be added to $SKY(H)$. Thus, $SKY(H)$ contains (6, 6) and (8, 4) up to now.

Similar to the second iteration, To-Server phase in the third iteration will transmit the one and only tuple (3, 8, 0.8, 0.5) in S_1's current local skyline list $SKY(D_1)$ to H, as shown in Table 2f. Then, server H selects tuple (3, 8) and broadcasts it to local sites D_2 and D_3 to obtain its global skyline probability. Accordingly, in the local-pruning phase, D_2 removes (4, 9, 0.6, 0.6) from $SKY(D_2)$, D_3 removes (3.5, 11, 0.7, 0.7) from $SKY(D_3)$, since they are dominated by (3, 8). Suppose the global skyline probability $P_{g\text{-}sky}((3,8))$ is larger than 0.3, then (8, 4) will be added to $SKY(H)$. Thus, $SKY(H)$ is updated to (6, 6), (8, 4), and (3, 8). Up to now, since all the local skyline set $SKY(D_i)(1 \leq i \leq 3)$ is empty as described in Table 2g, and the largest skyline probability for tuples within central server is smaller than 0.3 as shown in Table 2h, the exit condition holds and our e-DSUD algorithm terminates.

5.4 Updates

In this section, we explore how to continuously maintain the skyline results when data updates happen at the local databases D_1, $D_2, \ldots, D_m$ within every local sites S_1, $S_2, \ldots, S_m$. As mentioned above, our proposed algorithm obtains the global skyline from scratch. Thus, a straightforward solution is to restart the algorithm whenever the global skyline results need to be reported. However, since the update operation and invoked skyline query processing at server and local sites might be significantly deteriorated to become the bottleneck of the system performance, some scalable and efficient approaches, adopting incremental evaluation and sharing common information, have to be used in order to address the update performance.

Compared to the naïve solution, a much more efficient approach is to update the skyline result incrementally. That is to say, every time an insertion or a deletion operation is occurred at any local database, the central server H immediately modify priority queue L and global skyline set $SKY(H)$ if necessary. To implement the incremental strategy efficiently, we duplicate $SKY(H)$ at all local sites. In this way, when an update happens at a participant, it can decide whether L and $SKY(H)$ need to be modified without any bandwidth consumption. On the other hand, suppose that $SKY(H)$ is kept without any replica at the participants. Then, every data update at any participant must be routed to central server for inspecting the correctness of skyline results, thus causing serious network overhead.

In the sequel, we introduce the detail steps of skyline maintenance. Generally speaking, an update can be seen as a combination of a separate delete operation and a separate insert operation. First, to handle the delete operation, we suppose that a tuple t is deleted from D_m. Thus, if t is neither contained in the replica of $SKY(H)$ at S_m nor the representative tuple of S_m passed to L, the maintenance procedure does not need to rout any message to H, in this case, only a local index, i.e., PR-tree is searched according to the traditional top-down approach to locate and delete the data item t to be updated. On the other hand, if t belongs to the replica of $SKY(H)$, local site S_m sends t to H, which then transmits t back to other local sites except S_m to retrieve the skyline tuples pruned by t. If t is the representative tuple of S_m passed to L, it is deleted from L. At the same time, t is deleted from the local index constructed on D_m, and the skyline set $SKY(D_m)$ and $SKY(H)$.

Second, for an insert operation, assume that a tuple t is inserted into local site S_m. S_m checks whether t is dominated by any tuple in the replica of global skyline $SKY(H)$. If the answer is positive, the insertion of t has no influence on $SKY(H)$ and the maintenance is only processed on the local index within S_m. If the answer is negative, i.e., there is no dominating tuple from $SKY(H)$, t is added to $SKY(H)$ according to its skyline probability, and meanwhile, all the tuples in $SKY(H)$ dominated by t are checked to verify whether they still satisfy the query condition. Finally, S_m updates its own index, and informs the central server H together with other participants about the changes $SKY(H)$.

Note that, we use the central server H in the system to maintain the initial skyline results. The central server is responsible for broadcasting the information of the initial skyline results to all the local sites in network, when any update happens. Thus, each local site knows the initial skyline results of the whole network. Here, we assume that all of the local sites in network are stable, and therefore, use of the central server to maintain the initial skyline results and its alternations are applicable.

6 The Implementation Issues

Two trivial executions of our proposed algorithm are: 1) Calculate the local skyline set $SKY(D_i)$ in each local site S_i and then send the first element to coordinator H; 2) Broadcast each element in the priority queue to other local sets to obtain its global skyline probability $P_{g\text{-}sky}(a)$; then choose element with $P_{g\text{-}sky}(a) \geq \mathrm{q}$ to the final skyline set $SKY(H)$. Note that, the basic linear scan method may cause all the elements in database to be attached for computing the skyline probability. Thus, the amortized time complexity is $O(|N|)$ per element which is too pessimistic for query processing.

For managing a large number of elements, indexes are usually used to improve query efficiency. Currently, the most popular indexing technique for multidimensional data storage and access is the well-known R-tree [10]. In this section, we present novel techniques to efficiently execute algorithms based on R-tree with the aim to visit as few

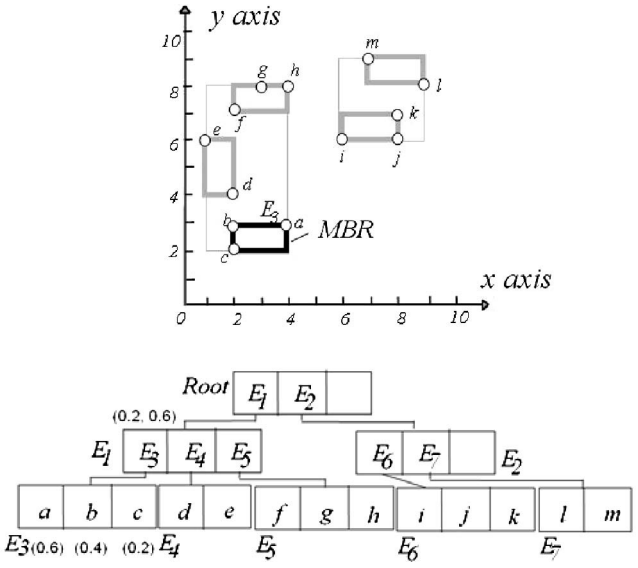

Fig. 5. The probabilistic R-tree.

elements as possible. Based on these techniques, we incrementally maintain the priority queue L and global skyline set $SKY(H)$.

The rest of this section is organized as follows: We first present index structures to be used. Then, we present our efficient techniques for computing the local skyline set within each local database regarding a given probability threshold. Finally, we present our techniques to obtain the global skyline probability of those tuples, which are broadcasted from the central coordinator to all the other local sites.

6.1 The Probabilistic R-Tree (PR-Tree)

This section explains the structure of a *Probabilistic R-tree* and its properties. A PR-tree is designed for pruning subtrees that do not contain any qualified skyline results, thus to accelerate the skyline query procedure. The general idea of PR-tree is to allow the entry of node in R-tree to store probability values. That is, each leaf entry contains not only the tuple's attributes, but also the existential probability of the tuple.

In particular, at each intermediate entry e contained in PR-tree, the following information is stored: a pointer referencing its child node, a minimum bounding rectangle (MBR) which includes all the objects in the child node of e, an existential probability P_1 which equals to the minimum occurrence probability of the elements rooted at e, and an existential probability P_2 which equals to the maximum occurrence probability of the elements rooted at e. Regarding the probabilistic R-tree with node capacity equals to 3 in Fig. 5, the occurrence probabilities of three elements a, b, and c are as depicted, therefore their upper level intermediate entry E_3 (corresponds to the minimum bound rectangle of a, b, and c) has the following existential probabilities $P_1(E_3) = Min\{0.6,\ 0.4,\ 0.2\} = 0.2,\ P_2(E_3) = Max\{0.6, 0.4, 0.2\} = 0.6$.

6.2 Local Skyline Query Procedure

Given a user specified probability threshold q, this section explains how to obtain the qualified local skyline set efficiently through the usage of PR-tree. Our proposed query algorithm is similar to the I/O optimal *Branch and Bounding Skyline* (BBS) algorithm proposed in [11]. In particular, the local skyline query procedure starts at the root node of the PR-tree and inserts all its entries into a heap sorted according to their minimum distance *mindist* to the space origin. The entry with the minimum *mindist* is selected to expand. This expansion removes the selected entry out of the heap first, and then inserts its children into the heap. The procedure continues and the next expanded entry is again the one with minimum *mindist*. When the element popped from the heap is any object a in the leaf node, and suppose the occurrence probability of a is larger than q, then object a belongs to the local skyline, and is inserted to the local skyline list $SKY(D_i)$. Otherwise, a is abandoned by the algorithm. The algorithm proceeds in the same manner until the heap becomes empty.

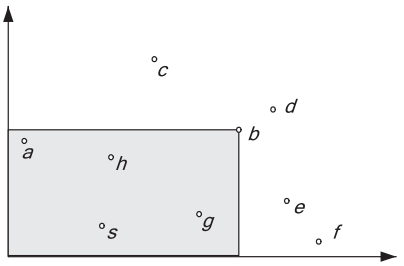

Fig. 6. The query window.

Note that, before an entry is inserted into the heap or is expanded, it has to be guaranteed that it contains or itself is a qualified local skyline. More precisely, suppose an intermediate entry b is dominated by an object a contained in local database D_i, and the largest occurrence probability P_2 of b multiplies the nonoccurrence probability of object a is smaller than threshold q, then there is no need to insert entry b into the heap, as there is no qualified skylines contained in the subtree of b.

6.3 Global Skyline Query Procedure

As explained above, to get the global skyline probability of a tuple t, the critical issue is to get its local skyline probabilities over all the local databases. We use the PR-tree to speed up the calculation of local skyline probabilities. In particular, for uncertain tuple b that broadcasted from the central server H. Suppose we calculate the local skyline probability of b against local database D_i, then we issue a window query (with space origin and the location of b as diagonal corners) on the PR-tree constructed over D_i to obtain those tuples dominate b. From these uncertain tuples returned by this query, we calculate the local skyline probability of b over D_i by multiplies their nonoccurrence probabilities. An example of this procedure is shown in Fig. 6 with one query window. Specifically, given one query tuple b, the shaded area contains all those tuples within the local database that dominate b, therefore the local skyline probability of b against this local database equals to $(1 - P(a)) \times (1 - P(g)) \times (1 - P(h)) \times (1 - P(s))$.

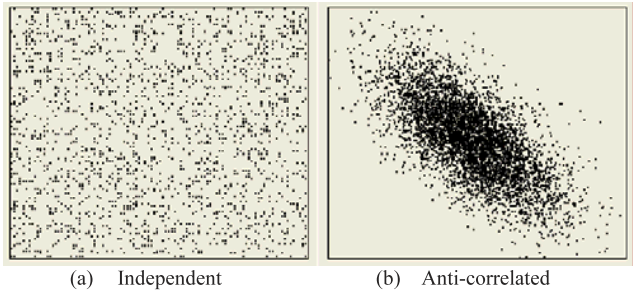

Fig. 7. Illustration of synthetic data.

7 Experimental Results

In this section, we empirically evaluate the efficiency and progressiveness of the proposed DSUD algorithm and its enhanced version e-USDU algorithm. Since the baseline approach, which simply asks all the participants to send their entire uncertain databases to the coordinator, is too communication expensive. We will consider the DSUD algorithm as the baseline algorithm, and compare e-DSUD against this baseline solution.

Data set. We use both the synthetic and real data sets for experiments. In particular, we create synthetic data sets with two popular distributions in the literature [2], [5], [22], [26]: *Independent* and *Anticorrelated*, whose generation follows the description in Fig. 7. In *Independent* distribution, all attribute values are generated independent of each other using a uniform distribution; in *Anticorrelated* distribution, if a point has a small coordinate on one dimension, it tends to have a large coordinate on one or all of the another dimensions. The dimensionality d of a synthetic data set is a parameter ranging from 2 to 5. The overall cardinality of the generated data set is 2 million tuples. Furthermore, we use *uniform* distribution to randomly assign an occurrence probability of each tuple to make them be uncertain. Clearly, in *uniform* distribution, the occurrence probability p of each tuple takes a random value between 0 and 1.

After the synthetic data set is generated, and given the number of m local sites, each tuple from the synthetic uncertain database D is assigned to site S_i chosen uniformly. Clearly, all local sites have the same data distribution. In particular, a local site server keeps a random sample set of the underlying data set, and the sample sets are mutually disjoint. In the experiments, every local server possesses an equal number of points, named the local cardinality. In case of m local sites, the local cardinality $|D_i|$ equals to $|N|/\mathrm{m}$, where $|N|$ is the cardinality of the entire data set.

User specified q is the probability threshold in evaluating the efficiency of query processing. To evaluate both the DSUD and e-DSUD algorithms, we use 0.3 as a default value of q, and evaluate its effect with 3 given probability thresholds 0.5, 0.7, and 0.9 that spread between [0.3, 1].

In the following experiments, we evaluate the efficiency of our algorithms, in terms of bandwidth consumption against dimensionality d, number of local databases m, and probabilistic threshold q under two distributions of objects' spatial locations. We also evaluate the progressiveness of our methods under different location distributions. Specifically, bandwidth consumption is measured by the number of tuples transmitted over the network. Progressiveness, on the other hand, is evaluated by measuring the bandwidth consumption cost and CPU runtime as a function of the number of qualified skyline tuples reported. It is expected that all the performance of e-DSUD algorithm is better than that of DSUD. Table 3 summarizes the parameters, as well as their values, to be used in our experiments. Unless specifically pointed out, each parameter is set to its default value as indicated in bold in Table 3. Note that, all the reported results are an average of 10 queries under the same parameter settings.

TABLE 3
System Parameters

Parameter	Values
N	2000K
m	40 **60** 80 100
d	2 **3** 4 5
q	**0.3** 0.5 0.7 0.9
P	[0, 1]

7.1 Performance with Dimensionality

In the first set of experiments, we test the effect of dimensionality d on the query performance. Specifically, the number of tuples transmitted during the query processing over two types of data sets are reported, with dimensionality varying from 2 to 5 by a step of 1, the total data size $|N| = 2{,}000$ K, the number of local sites equals to 60, and the user specified threshold $q = 0.3$.

Figs. 8a and 8b illustrate the total bandwidth consumption as a function of dimensionality d, under the *Independent* and *Anticorrelated* distributions, respectively. The numbers of transmitted tuples over two algorithms increase when d varies from 2 to 5. This is as expected, since larger the dimensionality of data tuples would make more tuples not to be dominated with each other, which makes the final skyline set become larger. Obviously, our e-DSUD algorithm requires considerably less bandwidth than its baseline DSUD algorithm, which indicates the efficiency and great importance of our proposed feedback mechanism in e-DSUD algorithm. Furthermore, we also observe that the *anticorrelated* data sets always have the larger bandwidth consumption than the *independent* data distribution under the same experimental settings. This is similar to the

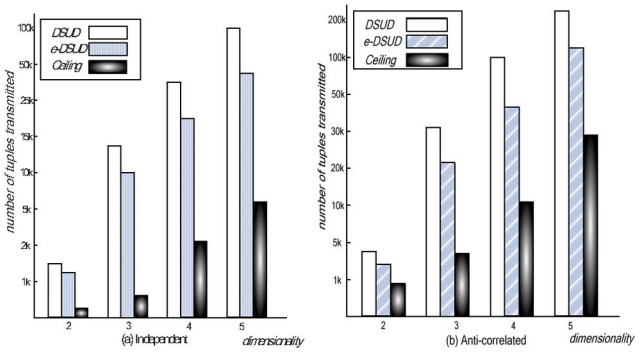

Fig. 8. Performance versus Dimensionality d.

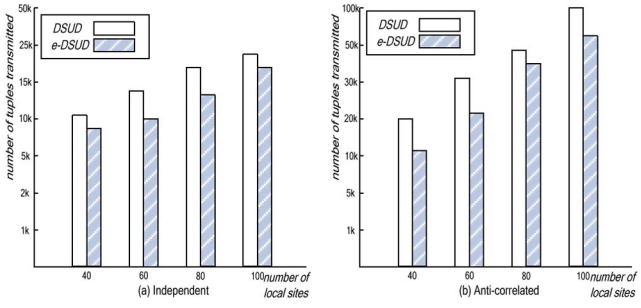

Fig. 9. Performance versus Number of local sites m.

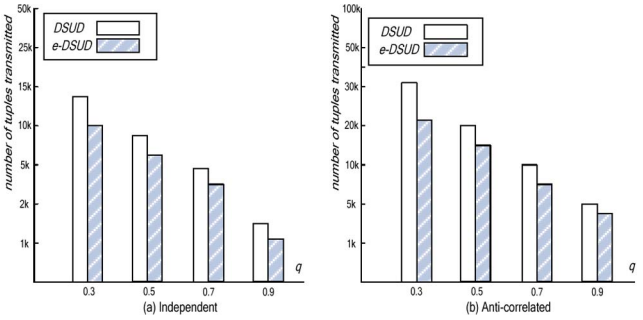

Fig. 10. Performance versus Threshold q.

situations of skylines on centralized database as the dimensionality grows.

Additionally, to get a better understanding of the relative performance of e-DSUD, we count the qualified global skyline tuples and calculate the minimum number of transmitted tuples, as depicted as *Ceiling* in Figs. 8a and 8b. We can see that performance of e-DSUD over anticorrelated data set is better than the performance over independent data set, this can be explained by the larger number of qualified skyline tuples in the anticorrelated data set. In the best case, the transmitted tuples of our e-DSUD algorithm is around 3 times larger than that of the optimal technique which could not be achieved in practice.

7.2 Performance with Number of Local Site

In the second set of experiments, our goal is to test the effect of the number of local uncertain databases m on the skyline query performance. Similarly, the number of tuples transmitted during the query processing over two types of data sets are reported, with m varying from 40 to 100 by a step of 20, the total database size $|N|$ is fixed at 2,000 K, the dimensionality of each tuple equals to 3, and the probability threshold $q = 0.3$.

Figs. 9a and 9b show the total bandwidth consumption when we vary m from 40 to 100, under the Independent and Anticorrelated distributions. The numbers of transmitted tuples by two algorithms increase when m gets larger. This is reasonable, in the feedback phase, a tuple selected from central server H is broadcasted to all the local sites S_i. Thus, m local sites mean at least a number of m tuples are transmitted over the network within one iteration. Since the total number of final skyline tuples, which should be delivered from H, is fixed according to the generated database, thus larger the number of local sites would make more network bandwidth consumption. Also, our proposed e-DSUD algorithm requires less bandwidth than baseline DSUD algorithm, which indicates the pruning efficiency of our proposed feedback mechanism in e-DSUD processing.

7.3 Performance with Threshold q

In the third set of experiments, we test the effect of the user specified threshold q on the query performance. Similarly, the number of tuples transmitted during the query processing over two types of data sets are reported, with the probability threshold q varying from 0.3 to 0.9 by a step of 0.2, the total data size $|N| = 2{,}000$ K, the dimensionality of each tuple equals to 3, and the number of local sites equals to 60.

As shown in both Figs. 10a and 10b, we find that, the numbers of transmitted tuples over two algorithms decrease when q gets larger. This is as expected, since the probability threshold q of the skyline query affects the total size of the final qualified skylines. Generally speaking, the smaller the probability threshold, the larger the skyline results. The reason is that if a tuple M belongs to p-Skyline, then it will always be in the result of p'-Skyline, where $p' \leq p$. Thus, since the total number of final skyline tuples, which should be delivered over the network, decreases according to the increasing threshold q, the larger value of q would make less network bandwidth consumption. Note that, the query performance is very sensitive to the change of probability threshold. This is because the feedback mechanism used in our framework can prune most of the unqualified skyline candidates under larger threshold.

7.4 Performance with Real Data Set

In the fourth set of experiments, we use the real data set NYSE borrowed from [2] to evaluate the query performance of our proposed algorithms. In particular, NYSE is extracted from the stock statistics from New York Stock Exchange, and it records 2 million stock transaction of Dell Incorporation from 1/12/2000 to 22/5/2001. Each transaction (tuple) has two attributes: the average price per volume and total number of volume. Similarly, in order to represent the transaction (tuple) uncertainty, we randomly assign a probability value to each transaction, following either *uniform* or *gaussian* distribution. In *uniform* distribution, the occurrences probability of each element takes a random value between 0 and 1, while in the *gaussian* distribution, the mean value μ varies from 0.3 to 0.9 and standard deviation σ is set to 0.2. The entire NYSE data set is assigned to m local sites equally.

As shown in both Figs. 11a and 11b, the bandwidth consumptions of our proposed algorithms over NYSE are reported by varying number of local sites m from 40 to 100 (the probability threshold is set to 0.3), and the probability threshold q from 0.3 to 0.9 (the number of local sites is set to 60), both with *uniform* distribution. In Fig. 11a, the numbers of transmitted tuples over two algorithms increase when m gets larger. This trend is similar to the synthetic data set in Fig. 9. Similarly, the numbers of transmitted tuples over two algorithms drop as q increases in Fig. 11b.

Figs. 11c and 11d report the impact of appearance probability distribution against the bandwidth consumption and number of skyline tuples. The results before $\mu = 0.5$ show that the larger the average appearance probability of the uncertain tuples, the more bandwidth will be consumed or the more skyline tuples will be kept in result. After $\mu = 0.5$

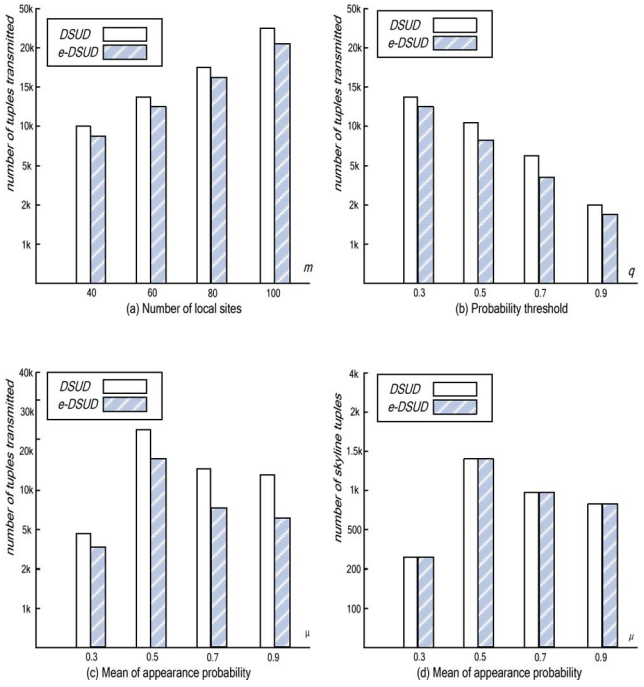

Fig. 11. Performance versus Real data set.

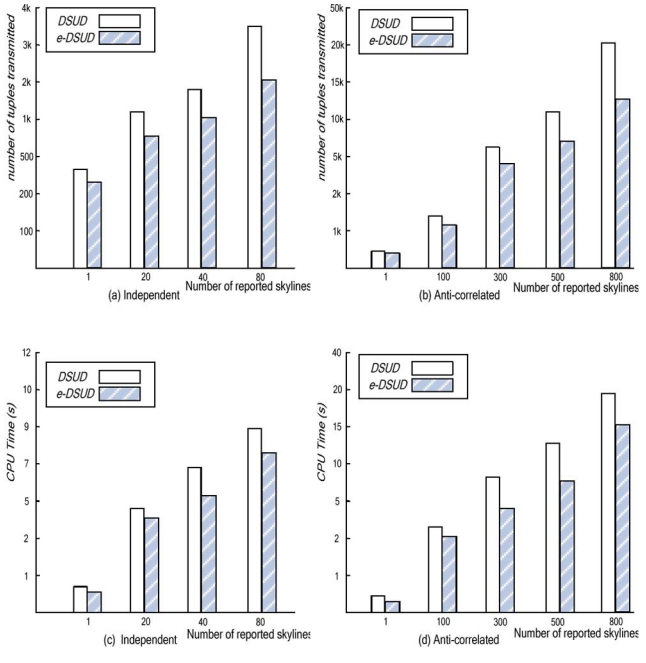

Fig. 12. Progressiveness versus Synthetic data set.

the bandwidth consumption and qualified skyline tuples are decreased when the average appearance probability increases. This is as expected, since the default user specified threshold equals to 0.3, a majority of those dominated tuples with appearance probability no larger than 0.5 cannot obtain their global skyline probabilities larger than 0.3, on the contrary, most of those free tuples with appearance probability larger than 0.5 themselves are in the final skyline result. Note that, although the bandwidth consumption of DSUD is larger than that of e-DSUD, as illustrated in Fig. 11c, the numbers of qualified skyline are the same, as illustrated in Fig. 11d. This is because the feedback mechanism used in e-DSUD prevents more unqualified tuples from being broadcasted to the local sites.

7.5 Progressiveness Performance

In the fifth set of experiments, we use both the synthetic data set and the real data set NYSE to evaluate the progressiveness of our proposed algorithms. We test the number of bandwidth consumption and the CPU runtime with the increasing number of reported skyline tuples. In particular, we generate synthetic data set under *Independent* and *Anticorrelated* distributions, the other parameter settings are as those default values in Table 3, and we use *uniform* function to assign an occurrence probability for each tuple randomly to make them uncertain. For real data set NYSE, to represent the tuple uncertainty, we also randomly assign a probability value to each tuple, following *uniform* or *gaussian* distribution as in Section 7.4.

As shown in both Figs. 12a and 12b, the accumulated bandwidth consumptions of our proposed algorithms over independent and anticorrelated distributions are reported by varying the number of reported skyline tuples. In Fig. 12a, the numbers of transmitted tuples over two algorithms act as a function of the number of returned skyline tuples. The e-DSUD exhibits a more flat changing line than DSUD, indicating relatively good progressiveness. This is expected because DSUD does not produce as many skyline tuples as e-DSUD after a certain number of iterations, that is to say, for the same increasing number of skyline tuples, DSUD algorithm needs more bandwidth consumption to get the needed skylines. The similar trend can be found in Fig. 12b for the synthetic data set with anticorrelated distribution.

To evaluate the speed of the algorithms in returning the skyline tuples incrementally, Figs. 12c and 12d report the CPU runtime of DSUD and e-DSUD as a function of the tuples returned for synthetic data set with independent and anticorrelated distributions, respectively. We can see that both algorithms return the first skyline tuple with short time

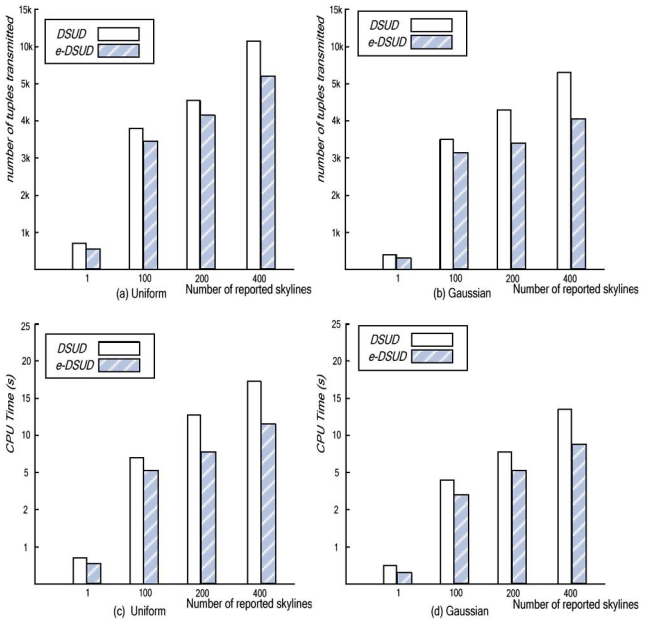

Fig. 13. Progressiveness versus Real data set.

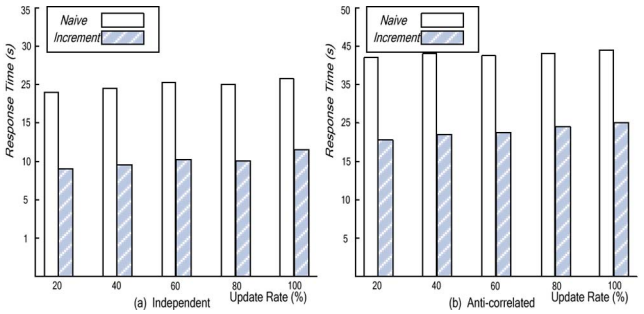

Fig. 14. Performance versus Data update.

cost after the query beginning, and e-DSUD can report a large number of skyline tuples before the query termination, which indicates their good progressiveness performance.

Figs. 13a, 13c and 13b, 13d demonstrate the results of the same experiments on real data set NYSE with tuple uncertainty following *uniform* and *gaussian* distributions, respectively, confirming the observations as discussed in above. In particular, for the gaussian distribution, the mean value is set to 0.5 and the standard deviation is set to 0.2. Generally speaking, more bandwidth and CPU time are consumed for uniform distribution, this is because, in the gaussian distribution, some local databases may have tuples that are dominated by those central tuples with high-appearance probability. Therefore, a broadcasting tuple in server may prune many local skyline points in local sites, thus reducing the bandwidth consumption and CPU time cost.

7.6 Update Performance

In the last set of experiments, we use the synthetic data set to show the performance of our proposed approaches on data update in local sites. In particular, the response time of the query processing over two types of data sets, that is with *Independent* and *Anticorrelated* distributions, are reported with the update rate varying from 20 to 100 percent by a step of 20 percent, the total database size $|N| = 2{,}000$ K, the dimensionality of each tuple equals to 3, and the number of local sites equals to 60.

We compare our e-DSUD algorithm with the Incremental update approach and the Naive approach proposed in Section 5.4. As shown in Fig. 14, the performances of e-DSUD with incremental approaches and Naive approaches are quite stable. The Incremental strategy can maintain the global skyline very efficiently for data update, and data update has negligible effect on query efficiency. The Naive mechanism has almost the same response time when the update rate increases and performs worse than the Incremental strategy because it needs more computation cost and higher communication cost. Additionally, the *Anticorrelated* date set adopting reversal values on each dimension offers more qualified skyline tuples for query processing, which costs much more response time.

8 Conclusion

The work in this paper studies skyline queries for distributed probabilistic data. We show that significant communication cost could be saved by exploring the interplay between the global skyline probability and its approximate value. Through the usage of Probabilistic R-tree, we also demonstrate how to alleviate the computation burden at each distributed site so that communication and computation efficiency are achieved simultaneously. Extensive experiments have been conducted to verify that our techniques can process skyline queries over distributed uncertain data both communication- and computation-effectively.

In this paper, we have focused on arbitrary horizontal partitioning, where a local site has all the attributes but stores only a subset of the entire tuples. As an interesting direction, vertical partitioning between distributed data still exists in the context of uncertain data. Thus, studying new algorithms to those cases is an important future work.

Acknowledgments

This work was supported by the National Natural Science foundation of China under Grant No. 61100060, the Key Project in the National Science and Technology Pillar program of China under grant No. 2008BAH29B00.

References

[1] F. Li, K. Yi, and J. Jestes, "Ranking Distributed Probabilistic Data," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '09),* June 2009.

[2] W. Zhang, X. Lin, Y. Zhang, W. Wang, and J. Yu, "Probabilistic Skyline Operator over Sliding Windows," *Proc. IEEE 25th Int'l Conf. Data Eng. (ICDE '09),* pp. 305-316, Mar. 2009.

[3] Y. Yuan, X. Lin, Q. Liu, W. Wang, J.X. Yu, and Q. Zhang, "Efficient Computation of the Skyline Cube," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05),* pp. 241-252, 2005.

[4] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting Stars: The K Most Representative Skyline Operator," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE '07),* pp. 86-95, 2007.

[5] S. Borzsonyi, D. Kossmann, and K. Stocker, "The Skyline Operator," *Proc. 17th Int'l Conf. Data Eng. (ICDE '01),* pp.421-430, 2001.

[6] X. Lian and L. Chen, "Monochromatic and Bichromatic Reverse Skyline Search over Uncertain Database," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 213-226, 2008.

[7] X. Lian and L. Chen, "Probabilistic Ranked Queries in Uncertain Databases," *Proc. Int'l Conf. Extending Database Technology (EDBT '08),* pp. 511-522, 2008.

[8] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2008.

[9] R. Jampani, F. Xu, M. Wu, L.L. Perez, C.M. Jermaine, and P.J. Haas, "MCDB: A Monte Carlo Approach to Managing Uncertain Data," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2008.

[10] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 47-57, 1984.

[11] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and Progressive Algorithm for Skyline Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 467-478, 2003.

[12] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi, "Holistic Aggregates in a Networked World: Distributed Tracking of Approximate Quantiles," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2005.

[13] R. Huebsch, M. Garofalakis, J.M. Hellerstein, and I. Stoica, "Sharing Aggregate Computation for Distributed Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2007.

[14] S. Wang, Q.H. Vu, B.C. Ooi, A.K.H. Tung, and L. Xu, "Skyframe: A Framework for Skyline Query Processing in Peer-to-Peer Systems," *The VLDB J.,* vol. 18, pp. 345-362., 2009.

[15] S. Michel, P. Triantafillou, and G. Weikum, "KLEE: A Framework for Distributed Top-k Query Algorithms," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2005.

[16] I. Sharfman, A. Schuster, and D. Keren, "A Geometric Approach to Monitoring Threshold Functions over Distributed Data Streams," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2006.
[17] B. Babcock and C. Olston, "Distributed Top-k Monitoring," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2003.
[18] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong, "Model-Driven Data Acquisition in Sensor Networks," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2004.
[19] N. Dalvi and D. Suciu, "Efficient Query Evaluation On Probabilistic Databases," *The VLDB J.,* vol. 16, no. 4, pp. 523-544, 2007.
[20] K. Deng, X. Zhou, and H.T. Shen, "Multi-Source Skyline Query Processing in Road Networks," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE '07),* 2007.
[21] A. Vlachou, C. Doulkeridis, and Y. Kotidis, "Angle-Based Space Partitioning for Efficient Parallel Skyline Computation," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2008.
[22] L. Zhu, Y. Tao, and S. Zhou, "Distributed Skyline Retrieval with Low Bandwidth Consumption," *IEEE Trans. Knowledge and Data Eng.,* vol. 21, no. 3, pp. 384-400, Mar. 2009.
[23] W.-T. Balke, U. Guntzer, and J.X. Zheng, "Efficient Distributed Skylining for Web Information Systems," *Proc. Ninth Int'l Conf. Extending Database Technology (EDBT '04),* pp.256-273, 2004.
[24] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis, "Skypeer: Efficient Subspace Skyline Computation over Distributed Data," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE '07),* pp. 416-425, 2007.
[25] P. Wu, C. Zhang, and Y. Feng, "Parallelizing Skyline Queries for Scalable Distribution," *Proc. Int'l Conf. Extending Database Technology (EDBT '05),* pp.112-130, 2005.
[26] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic Skylines on Uncertain Data," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2007.
[27] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM Int'l Conf. Data Comm.,* 2001.
[28] H.V. Jagadish, B.C. Ooi, and Q.H. Vu, "Baton: A Balanced Tree Structure for Peer-to-Peer Networks," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 661-672, 2005.
[29] A. Vlachou, C. Doulkeridis, K. Norvag, and M. Vazirgiannis, "On Efficient Top-k Query Processing in Highly Distributed Environments," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2008.
[30] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden, "Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),* pp. 174-185, 2005.
[31] L. Chen, B. Cui, H. Lu, L. Xu, and Q. Xu, "iSky: Efficient and Progressive Skyline Computing in a Structured P2P Network," *Proc. IEEE 28th Int'l Conf. Distributed Computing Systems (ICDCS),* 2008.
[32] X. Ding and H. Jin, "Efficient and Progressive Algorithms for Distributed Skyline Queries over Uncertain Data," *Proc. IEEE 30th Int'l Conf. Distributed Computing Systems (ICDCS),* pp. 149-158, 2010.
[33] B. Cui, L. Chen, L. Xu, H. Lu, G. Song, and Q. Xu, "Efficient Skyline Computation in Structured Peer-to-Peer Systems," *IEEE Trans. Knowledge and Data Eng.,* vol. 21, no. 7, pp. 1059-1072, July 2009.
[34] L. Chen, B. Cui, and H. Lu, "Constrained Skyline Query Processing against Distributed Data Sites," *IEEE Transaction on Data and Knowledge Eng.,* vol. 23, no. 2, pp. 204-217, Feb. 2011.
[35] Z. Zhang, Y. Yang, R. Cai, D. Papadias, and A. Tung, "Kernel-Based Skyline Cardinality Estimation," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2009.

Xiaofeng Ding received the PhD degree in computer science in 2009 from the Huazhong University of Science and Technology. Currently, he is an assistant professor in the School of Computer Science and Technology at Huazhong University of Science and Technology. His research interests include distributed computing, query processing, uncertain data management, peer-to-peer computing, and spatial databases.

Hai Jin received the PhD degree in computer engineering in 1994 from Huazhong University of Science and Technology (HUST), China, where he is currently the Cheung Kong professor and the dean of the School of Computer Science and Technology. In 1996, he was awarded a German Academic Exchange Service Fellowship to visit the Technical University of Chemnitz in Germany. He worked at the University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded the Distinguished Young Scholar Award from the National Science Foundation of China in 2001. He is the chief scientist of ChinaGrid, the largest grid computing project in China. He is the member of Grid Forum Steering Group (GFSG). He has coauthored 15 books and published more than 400 research papers. His research interests include distributed computing, computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage, and network security. He is a senior member of the IEEE and a member of the ACM.