IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 7, 2017, accepted February 5, 2018, date of publication February 15, 2018, date of current version March 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2806379

Parallel and Progressive Approaches for Skyline Query Over Probabilistic Incomplete Database

YIFU ZENG[1], KENLI LI[2], (Senior Member, IEEE), SHUI YU[3], (Senior Member, IEEE), YANTAO ZHOU[1], AND KEQIN LI[2,4], (Fellow, IEEE)

[1]College of Electrical and Information Engineering, Hunan University, Changsha 410082, China
[2]College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
[3]School of Information Technology, Deakin University, Burwood, VIC 3125, Australia
[4]Department of Computer Science, The State University of New York, New Paltz, NY 12561, USA

Corresponding author: Yifu Zeng (zengyifu@hnu.edu.cn)

This work was supported in part by the Key Program of National Natural Science Foundation of China under Grant 61472126 and Grant 61432005, in part by the International (Regional) Cooperation and Exchange Program of National Natural Science Foundation of China under Grant 61661146006, in part by the National Natural Science Foundation of China under Grant 61370095, and in part by the National Key Research and Development Program of China under Grant 2016YFB0200201.

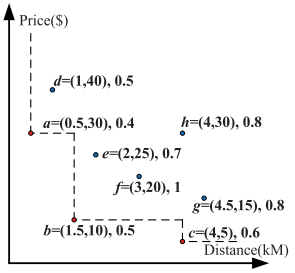
ABSTRACT The advanced productivity of the modern society has created a wide range of similar commodities. However, the descriptions of commodities are always incomplete. Therefore, it is difficult for consumers to make choices. In the face of this problem, skyline query is a useful tool. However, the existing algorithms are unable to address incomplete probabilistic databases. In addition, it is necessary to wait for query completion to obtain even partial results. Furthermore, traditional skyline algorithms are usually serial. Thus, they cannot utilize multi-core processors effectively. Therefore, a parallel progressive skyline query algorithm for incomplete databases is imperative, which provides answers gradually and much faster. To address these problems, we design a new algorithm that uses multi-level grouping, pruning strategies, and pruning tuple transferring, which significantly decreases the computational costs. Experimental results demonstrate that the skyline results can be obtained in a short time. The parallel efficiency for an Octa-core processor reaches 90% on high-dimensional, large databases.

INDEX TERMS Data management, incomplete data, parallel processing, progressive processing, probabilistic products, skyline query.

I. INTRODUCTION

The widespread use of portable Internet devices such as mobile phones and tablets, allows people to share comments on all kinds of commodities, whenever and wherever they want. In addition, people tend to rely on item descriptions and comments on the Internet, which are provided by merchants or other customers, in selecting their favorite items. Therefore, the growing numbers of comments on review sites, such as TripAdvisor and Yelp, are playing an increasingly important role in our daily lives. However, due to many issues, such as limitations of measuring equipment, deliberate withholding of disadvantages, and transmission errors, most of the information is incomplete. Additionally, the randomness of the commodities and the reliability of the data increases the probabilistic uncertainty of the data. The positive review rate and the number of comments also influence the reliability of the ratings, since they reflect the gap between the description of commodities and the subjective feelings of clients. Consequently, the information of a commodity usually carries uncertainty of not only probability but also completeness. Therefore, processing these probabilistic incomplete data efficiently and accurately would be an important achievement.

We give an example to illustrate a practical application scenario. Suppose Mr. Smith wants to enjoy his meal nearby, and the restaurant information in terms of distance, price and reliable rate is as shown in Fig. 1(a). For two restaurants with similar distance, price, or other parameter such as facilities or environment, the one with more comments and a higher positive rate is more reliable. The number of comments indicates the popularity of the restaurant. Apparently, a popular restaurant is more likely to be truly and accurately described. A negative review rate indicates the probability that a customer's dining experience will not match that shown in the advertising campaigns. Therefore, the restaurant with higher positive rate is more reliable. We use P-skyline



(a)

Restaurant	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
Distance(kM)	0.5	1.5	4	1	2	3	4.5	4
Price($)	-	10	-	40	25	20	15	-
Good review rate	0.4	0.5	0.6	0.5	0.7	1	0.7	0.8

(b)

FIGURE 1. Two forms of uncertain datasets. (a) Probabilistic tuples. (b) Probabilistic incomplete tuples.

query [1] to extract the best candidates from this database. Restaurants b, c and f are the best choices if we set the threshold to 0.5. Therefore, Mr. Smith should choose his dining place from the subset $\{b, c, f\}$ as these restaurants are better than other candidate restaurants.

However, a more realistic situation is that the data of each tuple are incomplete. As shown in Fig. 1(b), the average price information of some restaurants is lost. This is a very common phenomenon on most shopping websites and recommendation systems. Faced with this situation, Mr. Smith would be puzzled again because the previous skyline set $\{a, b, c\}$ may not contain the best choices now.

Skyline query is a powerful tool for data mining and decision making and is highly suitable for this case. It has received significant attention from the database community [2]–[5]. In addition to skyline query, some similar algorithms have been proposed for obtaining suitable candidates [6], [7]. There are many sophisticated solutions for complete, certain databases, which means the skyline query results can be obtained in a short time [8]. Additionally, for probabilistic databases, such as that shown in Fig. 1(a), there have been many recent developments of skyline queries [1], [9]–[13]. These developments are important, as this research has many applications. A famous example is the selling of airline tickets at http://www.bing.com/travel. The availability and price of airline tickets show substantial uncertainty. To help their users make decisions, bing.com provides a service that predicts changes in tickets prices. This problem is based on the possible world semantics [14], which is widely adopted in probabilistic databases [15]–[19].

However, the progressive skyline query is unwarranted with these methods, but is very important in practical applications. People may not need to choose from all possible answers. Providing several candidates for customers in a short time is very helpful. For example, Mr. Smith wants to obtain recommendations for nearby restaurants. With the the non-progressive algorithm, he obtains 40 options after 30 seconds. With a progressive algorithm, he obtains five options in the first two seconds, and the other options are given soon afterwards. In most cases, five candidates are sufficient for making a choice. Therefore, a progressive algorithm is a new challenge with many promising applications, which better matches practical needs. To address this problem, we need to evolve the algorithm and make it supportive of progressive query. Moreover, with the increasing popularity of multi-core portable devices, parallelizable algorithms are becoming more popular. An efficient parallel algorithm can be run in a smart phone and tablet, which makes it feasible to obtain all kinds of recommendations based on local databases with greatly decreased waiting time.

Motivated by these issues, we make the following contributions:

- We define tuple grouping, strict group encoding and an inclusion relation as the basis of the algorithm, which enhance the parallel efficiency of the algorithm.
- We provide an optimized definition of the incomplete data mode, which provides more reasonable answers.
- We formulate the algorithm with progresion, which reduces the waiting time for most users who do not need a complete candidate set to make a choice.
- We propose several pruning approaches in both the pretreatment stage and the comparsion stage, which reduce the response time of the system.
- We perform an extensive experimental study on both synthetic and real datasets to evaluate the efficiency and effectiveness of our proposed algorithm, especially on massive and high-dimensional datasets.

The rest of the paper is organized as follows: In Section III, we propose the incomplete data model and the P-skyline model. The related works are also reviewed. In Section IV, we design some effective pruning strategies and algorithms for overcoming the problems in skyline queries over probabilistic incomplete data. In Section V, we evaluate the performance of the proposed algorithm by extensive experiments. In Section VI, we present the conclusions of the paper and discuss directions for future work.

II. RELATED WORK

Skyline query is a popular paradigm for extracting interesting objects from multi-dimensional databases. Skyline query also garnered considerable research attention over uncertain data. Pei *et al.* [1] first proposed probabilistic skyline query over uncertain databases. Kin et al. introduced a probabilistic skyline algorithm called P-skyline, which computes the exact skyline probabilities of all objects. Its performance is scalable with the dataset size or the dimensionality. Lian and Chen focused probabilistic reverse skylines, and considered both monochromatic and dichromatic cases [20]. Zhang *et al.* [21] studied the problem of efficiently computing skylines against sliding windows over an uncertain data stream. Recently, Lian and Chen [22] proposed a novel and important query for uncertain databases, namely, probabilistic group subspace skyline query (PGSS), and presented an efficient query procedure. Ding and Jin [23] proposed the notation of distributed

skyline queries over uncertain data and designed two computationally and communication-efficient algorithms. Most of these previous studies are based on P-skyline. Specification by the user of an appropriate probability threshold is one of the challenges for P-skyline. Moreover, in P-skyline, tuples are considered individually, and the dominance relationship between them is not taken into account.

Although incomplete data exist widely in practice, the amount of research that considers incomplete data, let alone probabilistic incomplete data, is very limited. A closely related work to incomplete data is the k-dominant skyline problem [24]. The k-dominant skyline algorithm overcomes the non-transitivity property by discarding points that are dominated in all dimensions, while keeping points that are only dominated in k dimensions. The k-dominant skyline algorithm cannot applied to incomplete data directly and incurs prohibitive costs, which can be avoided with the knowledge of incomplete dimensions. Khalefa *et al.* [25] first defined a dominance relation for incomplete data, which is different from ours. They also introduced the ISkyline algorithm for skyline computation over incomplete data. Haghani *et al.* [26] addressed the problem of processing continuous top-k queries over incomplete data streams. Wolf *et al.* [27] introduced QPIAD, which is a novel query rewriting and optimization framework that tackles the challenges of incomplete autonomous databases. Soliman *et al.* [28] explored a novel probabilistic model that extends partial orders to represent the uncertainty in the score of database records, and formulated several types of ranking queries on this model. Gao *et al.* [29] proposed the k-skyband skyline for incomplete databases in 2014. Cheng *et al.* [30] studied similarity search on dimension-incomplete data. Lofi *et al.* [31] proposed an approach for the challenge of dealing with missing information in datasets in connection with skyline query processing, by using crowd-enabled databases.

However, none of them studied the database with uncertainties of both incomplete and probabilistic characteristics. In this paper, we formalize a progressive skyline query over probabilistic incomplete data. In addition, this query is realized in a parallel way.

III. PRELIMINARIES

In this section, we first present the uncertain data model that we use in this paper. Afterwards, an innovative incomplete data model is proposed. To avoid misunderstanding, we assume that the smaller value is better. For reference, Table 1 summarizes the symbols that are used frequently in the rest of this paper.

A. UNCERTAIN DATA MODEL

There have been many research works about query processing on the locationally uncertain data model, which emphasizes processing the uncertainty of the location [1], [32], [33]. All the objects in this model are known to exist. Admittedly, this uncertain model is useful in many cases. However, the uncertain model is not applicable to our problem about incomplete probabilistic products,in which it is possible for each product to be unavailable. Therefore, we conduct our research on the basis of another existing uncertain data model. Different from the previous model, this one has locationally certain objects with existence probabilities (see Table 2). This model is widely used in a variety of online trading sites and previous papers [5], [19], [22].

TABLE 1. Symbols and description.

Notation	Description
$t_p \prec t_q$	t_p dominates t_q
$t_p \not\prec t_q$	t_p does not dominate t_q
$P(t_p)$	the existing (reliable) probability of t_p
$t.[i]$	the i-th dimensional value of t
$t.b$	tuplue t's bitmap
B_m	a bitmap bucket contains tuples of $t.b = m$
$P_{non-dom}(S)$	S's non-dominance probability
$P_P(S)$	S's P-skyline probability
$C(t)$	tuple t's integrity
B_m^s	monotonic sorted B_m
B_m^R	reduced B_m
$PS(B_m)$	pruning set of B_m
$©t_p$	complete pruning tuple t_p
$PQ(S)$	P-skyline answer set of S
G_j	an independent partition group

TABLE 2. Distinctions between two models [32].

Property	Locationally Uncertain Object	Existentially Uncertain Object
Location	uncertain	certain
Existence	certain	uncertain
Storage	a spatial histogram	a point with existence probability

The research works about skyline query over probabilistic data are mostly based on the P-skyline Model, which was first proposed in [1]. Here, we give an example to illustrate P-skyline. According to Fig. 1(a), the non-dominated probability of tuple d is calculated by $P_{non-dom}(d) = \overline{P(a)} \times P(d) = 0.3$, because d is not dominated only if a is unavailable. Similarly, $P_{non-dom}(b) = P(b) = 0.5$, no tuple can dominate b. In summary, P-skyline returns individual data tuples with non-dominance probabilities that are greater than or equal to a specified threshold. In this example, for a threshold of 0.5, the P-skyline answer set is $\{b, c, f\}$.

B. INCOMPLETE DATA MODEL

Due to the existence of missing dimensional values, the traditional definition of a dominance relationship under complete data is no longer applicable. The existing research works on skyline query over incomplete data use an incomplete data models, as in Definition 1 [25], [29]. Admittedly, Definition 1 provides a feasible model for tackling incomplete data. However, this model is unreasonable in some cases. As shown in Fig. 1, tuple a dominates all the other tuples when its price information is missing. Restaurant a has a relatively high price in this database. A customer may find the prices unaffordable when he arrives at restaurant a.

TABLE 3. Four-dimensions incomplete dataset.

Restaurant	Price (5$)	Distance (kM)	Taste (review scores)	Service (review scores)
a	5	3	2	1
b	1	3	2	2
c	2	2	1	5
d	2	3	2	3
e	2	-	1	5
f	1	-	3	2
g	-	1	4	1
h	-	3	2	1
i	1	3	-	-
j	3	1	-	-
k	4	-	-	1
m	-	-	2	2
n	1	-	-	-
p	-	1	-	-
q	-	-	1	-
r	-	-	-	1

Therefore, we need a more reasonable incomplete data model for performing a more reasonable skyline query over incomplete data. The definition and further comparisons are given below.

Definition 1: Assume that $t.[i]$ is the i-th dimensional value of t. Tuple t dominates another tuple t' (denoted as $t \prec t'$) if the following two conditions hold:

- For each dimension i, either $t.[i]$ and/or $t'.[i]$ are/is unknown, or $t.[i] \leqslant t'.[i]$.
- There is at least one dimension j in which both $t.[j]$ and $t'.[j]$ are known, and $t.[j] < t'.[j]$.

We define a new dominance relationship over incomplete data in Definition 2. The detail comparisons of Definition 1 and Definition 2 are given under Definition 3.

Definition 2 (Dominance Relationship Over Incomplete Data): Assume that $t.[i]$ is the i-th-dimensional value of t. Tuple t dominates another tuple t' (denoted as $t \prec t'$) if the following four conditions hold:

- There is at least one dimension i in which both $t.[i]$ and $t'.[i]$ are known.
- For each dimension j in which both $t.[j]$ and $t'.[j]$ are known, $t.[j] \leq t'.[j]$.
- There is at least one dimension k in which $t.[k] < t'.[k]$ or $t.[k]$ exists while $t'.[k]$ is unknown.
- There does not exist any dimension m in which $t.[m]$ is unknown while $t'.[m]$ is known.

The skyline set for an incomplete data set S is defined as follows:

Definition 3 (Skyline Set): The skyline set of an incomplete multidimensional data set S is a set of tuples that are not dominated by any other tuple in S.

A relatively more complete database (see Table 3) is introduced for analyzing the differences between Definition 1 and Definition 2. For convenience, we adjust the measurement unit to simplify the data and assume that a lower measurement value is better. Additionally, we use different background colors to differentiate the lost dimensions.

Under Definition 1, information lose seems to be advantageout for the merchants, which is unreasonable. According to Table 3, restaurant n dominates nearly all the other restaurants, with only the value of the first dimension known, as do restaurants p, q and r. Consequently, the skyline set of this dataset under Definition 1 is $\{n, p, q, r\}$, which is apparently unacceptable for customers. This set may encourage the merchants to lose some information intentionally.

The same patterns under Definition 2 will yield more reasonable results. We give priority to tuples with higher completeness to prune the relatively incomplete tuples. According to Table 3, tuple a dominates tuples r and h; tuple b dominates tuples$\{n, m, i, f, d\}$; tuple c dominates tuples q and e; and tuple g dominates tuple p. Consequently, the skyline set of this dataset, according to Definition 2, is $\{a, b, c, g, j, k\}$, which is obviously more applicable and helpful for customers. People who are sensitive to price could choose b, which is the cheapest option with good taste and service. People who do not care about money but have limited time, can choose g. Since this nearby restaurant has the best service, it could also be chosen to ensure a positive dining experience.

Moreover, if a tuple has a great value in any one dimension, but lose values of all the other dimensions, it still can be a skyline answer under Definition 2. For example, if the distance of tuple p is 0.5, it will still be a skyline answer. Therefore, for our skyline query over incomplete data, Definition 2 is more suitable.

Definition 4: P-skyline query over a probabilistic incomplete database S outputs a subset of S that has a P-skyline probability that is larger than a threshold α.

The answer set of the database in Fig. 1(b) according to Definition 4 is $\{b, d, f\}$ if we set the threshold to 0.5. This paper studies the problem of apply P-skyline query over a probabilistic incomplete database, which is given in Definition 4.

IV. SKYLINE QUERIES PROCESSING

In this section, we first put forward a baseline algorithm for probabilistic incomplete data. Then, several methods are introduced for enhancing the efficiency of the algorithm.

A. BASELINE ALGORITHM

The baseline algorithm is designed for the control test. Since there is no existing any algorithm that is designed for P-skyline query over a probabilistic incomplete database, the baseline algorithm is simply designed according to Definition 2 and the definition of P-skyline [1]. As shown in Algorithm 1, the baseline algorithm is a brute-force algorithm; it is time-consuming and inapplicable to big databases due to the high computational burden of calculating P-skyline probabilities. Moreover, the baseline algorithm is a serial algorithm.

B. GROUPING AND SORTING

Most existing skyline query algorithms are serial because of the dependence among tuples. The computation of a tuple

Algorithm 1 Baseline Algorithm

Input: d-dimensional probabilistic incomplete data set S.
Output: $PQ(S)$.

```
Initialize PQ(S)
for each tuple t_p∈S do
    Calculate P_P(t_p);
    if P_P{t_p}≥α then
        Put t_p into PQ(S);
Return PQ(S).
```

often relies on the results for other tuples. Therefore, the key to parallelization is to determine which tuples are independent of one another.

Bitmap is an important concept in the processing of incomplete data [25], [29]. For ease of representation and computation, we represent a d-dimensional incomplete tuple t by a d-bit bitmap vector $t.b$ whose entries are 1 for all complete dimensions and 0 for all incomplete dimensions. For example, the bitmaps of tuples $t = (3, -, 2, -)$ and $t' = (-, 2, 2, 3)$ are $t.b = 1010$ and $t'.b = 0111$, respectively.

For the convenience of describing the incompleteness of a tuple or a bitmap group, integrity is defined in Definition 6. For example, the integrity of tuples $t = (3, 1, 2, -)$ and $t' = (-, -, -, 3)$ are $C(t) = 3$ and $C(t') = 1$, respectively.

Definition 5 (Independent Tuples): Tuples a and b are independent tuples if $a \not\prec b$ and $b \not\prec a$.

Definition 6 (Integrity): The integrity $C(t_i)$ of any tuple t_i is the sum of the entries of its bitmap vector. The integrity $C(B_m)$ of any bitmap group B_m is the integrity of the tuple in this group. The maximum integrity value of a tuple is its dimension d and the minimum integrity value of a tuple is 1.

Assume that tuple a and tuple b have the same integrity value and different bitmaps. It can be demonstrated that they are independent tuples. Therefore, we can allocate them to different buckets according to their bitmaps and integrity value for convenience of parallelization.

Lemma 1: Two tuples are independent tuples if they have the same integrity value and different bitmaps.

Proof: Assume that tuple a and tuple b have the same integrity value and different bitmaps. There is at least one dimension, i, in which $a.[i]$ is known and $b.[i]$ is unknown. Moreover, there is at least one dimension, j, in which $b.[j]$ is known and $a.[j]$ is unknown. According to Definition 2, $a \not\prec b$ and $b \not\prec a$. Therefore, a and b are independent tuples. □

As shown in Fig. 2, the buckets with the same integrity value have the same priority level. Thus, they can be processed at the same time. For tuples with the same bitmap, the dominance relationship is the same as that for complete data. We can process the skyline query by each bitmap in a particular order. Besides, some of the settled answers can be used to prune tuples before processing.

Lemma 2: All tuples in B_m, where $C(B_m) = d$, will not be dominated by any tuple in B_n, where $C(B_n) = d - 1$.

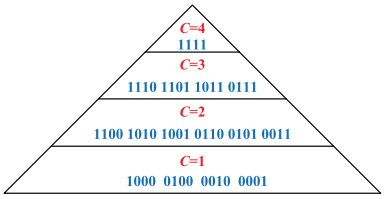


FIGURE 2. Integrity pyramid.

Proof: According to Definition 2, the number of missing dimensions of tuples in B_n is greater than that of B_m, so that Lemma 2 holds. □

According to Lemma 2, it is impossible for all of the local P-skyline answers to be dominated by tuples in the lower layers. Moreover, it is impossible for tuples that belong to different buckets of the same layer to have a dominance relationship. Therefore, once we obtain a local P-skyline answer, we can output it immediately because it is the final answer. Progression can be realized by this way.

Before the processing of each bucket, it is necessary to sort it according to a set of rules. For B_m with integrity d, sorting should be carried out according to the following rules:

- Prepose the tuple with smaller value in any dimension.
- For tuples with the same minimum value, prepose the tuple with the smaller sum value over all dimensions.

For example, subset $\{a, b, c, d\}$ in Table 3 will be sorted as $\{b, c, a, d\}$ because tuples b, c, and a have a minimum value of 1, while the minimum value of d is 2. In addition, the sum value of b over all dimensions is 8 while those for c and a are 10 and 11 respectively.

Lemma 3: For a sorted group, it is impossible for a candidate tuple to be dominated by a postpositional tuple.

Proof: Assume that tuple a is in front of tuple b. In the dimension in which the minimum value of a is attained, if b is larger than a, it cannot dominate a. If b is equal to a in that dimension, there is at least one dimension in which b is larger than a because the sum of b over all dimensions is larger than that of a. Therefore, according to Definition 2, it is impossible for b to dominate a. □

Algorithm 2 Grouping and Sorting

Input: d-dimensional probabilistic incomplete data set S.
Output: $2^d - 1$ numbers of sorted bucket B_m^s.

```
Initialize 2^d − 1 numbers of bucket B_m;
while S is not empty do
    Read an object s from S;
    Put s into a bucket B_m based on its bitmap;
    Remove s from S;
Monotonic sort each bucket B_m;
Return B_m^s
```

The sorting in descending order can be performed in each group in a parallel way. Meanwhile, it can be processed by the grouping operation. Algorithm 2 illustrates this procedure.

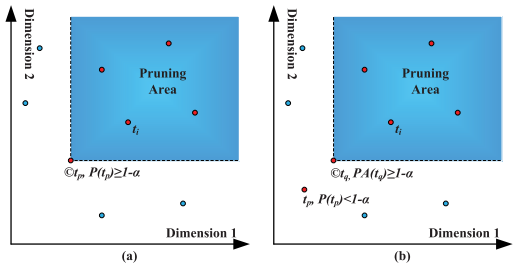


FIGURE 3. Complete and incomplete pruning tuples.

C. INDIVIDUAL BUCKET PROCESSING

The processing of each bucket begins from the top of the integrity pyramid, which is shown in Fig. 2. It is impossible for tuples in the top bucket to be dominated by tuples in lower layers so the answers can be output progressively. Moreover, we can use some tuples with higher dominance ability to prune other tuples, which accelerates the processing speed. In order to clarify the dominance ability of each tuple, Definition 7 is given as follows. A higher value of pruning ability of a tuple indicates the greater chance of this tuple to dominate other tuples.

Definition 7 (Pruning Ability): The pruning ability for tuple t_p is denoted as $PA(t_p)$ and is computed by

$$PA(t_p) = 1 - (1 - P(t_p)) \times (1 - P(t_q)) \times \prod_{\forall t_x, t_q \prec t_x \prec t_p} (1 - P(t_x))$$

Definition 8 (Completed Pruning Tuple): A tuple t_p is a complete pruning tuple if its pruning ability is larger than or equal to $1-\alpha$. the tuple is marked with ©.

Lemma 4: All tuples that are dominated by a complete pruning tuple ©t_p can be safely discarded without affecting the answer set.

Proof: Assuming that t_i is dominated by ©t_p. $P_p\{t_i\} = P(t_i) \times \prod_{\forall t_x, t_x \prec t_i}(1 - P(t_x))$. Since $PA(t_p) \geq 1-\alpha$, so that $(1 - P(t_p)) \times \prod_{\forall t_x, t_x \prec t_p} \leq \alpha$. Since $t_p \prec t_i$, we have $(1 - P(t_i)) \times \prod_{\forall t_x, t_x \prec t_i} < (1 - P(t_p)) \times \prod_{\forall t_x, t_x \prec t_p} < \alpha$. So that $P_p\{t_i\} < P(t_i) \times \alpha$, which is less than α. Therefore, t_i can be safely discarded and this lemma holds. □

Some tuples that are not marked as © are also useful in pruning strategies. These incomplete pruning tuples can help produce complete pruning tuples. According to Fig. 3(a), tuple t_p is a complete pruning tuple when its existence probability is larger than $1-\alpha$. When $P(t_q) < 1-\alpha$, it is possible to help t_p become a complete pruning tuple. With assistant tuples, it is still possible for a tuple with low existence probability to become a complete pruning tuple. For example, as shown in Fig. 3(b), suppose we have pruning tuples t_p and t_q. $P(t_q) = 0.3$, $P(t_p) = 0.5$ and $t_q \prec t_p$. Since $PA(t_q) < 0.6$ and $PA(t_p) > 0.6$, t_p can be marked as © while t_q is not. Further, if we set $P(t_p) = 0.4$ and keep other conditions invariably, $PA(t_p)$ will be 0.58, which is less than 0.6. If we have another pruning tuple t_r that dominates t_p, even if its existance probability is only 0.1, $PA(t_p)$ will be larger than $1-\alpha$ and t_p will become a complete pruning tuple.

TABLE 4. Three-dimensions probabilistic incomplete dataset.

Restaurant	Price (5$)	Distance (kM)	Taste (review scores)	Positive Rate (rates)
a	7	3	1	0.9
b	5	7	1	0.4
c	5	9	1	0.5
d	2	5	6	0.7
e	8	9	2	0.6
f	3	-	1	0.9
g	2	-	8	0.8
h	2	3	-	0.6
i	6	9	-	1
j	-	-	1	0.8
k	-	3	-	0.5
m	4	-	-	0.9

Therefore, a tuple set that contains complete pruning tuples and incomplete pruning tuples should be created in the processing procedure. We can directly discard tuples that are dominated by a complete pruning tuple.

With Lemmas 2, 3 and 4, Algorithm 3 is designed.

Algorithm 3 Single Bucket Processing

Input: B_m.
Output: Answer set $PQ(B_m)$.

```
1:  Initialize pruning set PS(B_m)
2:  Initialize answer set PQ(B_m)
3:  while B_m is not empty do
4:      Read a tuple t_p from B_m;
5:      if PS(B_m) ⊀ t_p then
6:          if P(t_p) ≥ α then
7:              PQ(B_m) = PQ(B_m) ∪ t_p;
8:              Output t_p;
9:          PS(B_m) = PS(B_m) ∪ t_p;
10:         if P(t_p) ≥ (1 − α) then
11:             Mark t_p as ©;
12:     else if ∃©t_q ≺ t_p, t_q ∈ PS(B_m) then
13:         Discard t_p;
14:         Break;
15:     else if ∃t_q ≺ t_p, t_q ∈ PS(B_m), ∧t_q is not © then
16:         Calculate PA(t_p);
17:         if PA(t_p) ≥ (1 − α) then
18:             Mark t_p as ©;
19:         Calculate P_p(t_p);
20:         if P_p{t_p} ≥ α then
21:             PQ(B_m) = PQ(B_m) ∪ t_p;
22:             Output t_p;
```

An example is presented to explain this algorithm. Table 4 shows a sorted dataset. Assume that the threshold is 0.4. Bucket B_{111} consists of tuples a, b, c, d, e. Tuples a and b are put into $PQ(B_{111})$ and $PS(B_{111})$ in lines 5 to 8 of Algorithm 3 where a is marked as ©, while b is not. Then, tuple c is dominated by an incomplete pruning tuple b. Tuple c is put into $PS(B_{111})$, but not $PQ(B_{111})$, after the calculation of $PA(c)$ and $P_p\{c\}$. Afterwards, d is put into $PQ(B_{111})$ and $PS(B_{111})$, while e is pruned by completed pruning tuple a.

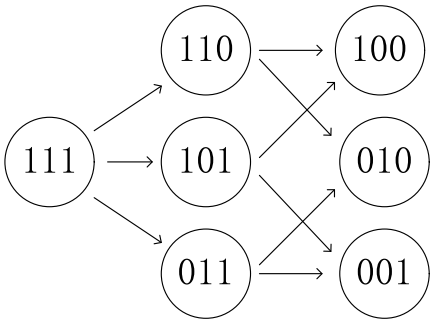


FIGURE 4. ***PS*(B_m) transfer to its subsets.**

From this procedure, we obtain $PQ(B_{111})$ of a, b, d and $PS(B_{111})$ of ©a, b, ©c, ©d.

D. LOWER-LAYER PROCESSING

The pruning set can be transferred to the subsets after we obtain the pruning set of a higher-layer bucket. The pruning efficiency can be greatly enhanced by this approach.

Definition 9: Tuples in Pruning Set $PS(B_m)$ of bitmap group B_m will be transferred to subsets $PS(B_n)$, which is the Pruning Set of bitmap group B_n if the following two conditions hold:

- Exactly on digit of $B_m.b$ is 1, while $B_n.b$ is 0.
- For each digit of $B_m.b$ that is 0, the same digit of $B_n.b$ must be 0 as well.

Lemma 5: Tuples that are supposed to be in the final P-skyline results will not be pruned by the transferred pruning tuples.

Proof: t_i is a tuple that is supposed to be in the final result set $PQ(S)$ without the use of the transferred pruning tuples. Assume that t_i will be pruned by transferred pruning tuple t_j^t. Since t_j^t is a tuple that was copied from t_j with one or more missing dimensions, t_j can also prune t_i. Thus, t_i cannot be a final skyline tuple and the assumption is invalid. Therefore, t_i will not be pruned by any transferred pruning tuple and Lemma. 5 holds. □

The notation of pruning set transfer is introduced to reduce the calculation workload and enhance the pruning efficiency. After the processing of the sorted bitmap group B_m^s, which has the maximum value of integrity $C(B_m) = d$, the $PS(B_m)$ can prune not only tuples in B_m^s but also tuples in any other bitmap group B_n^s that has an integrity value of $C(B_n) = d-1$. As illustrated in Fig. 4, $PS(B_m)$ from the upper level should be transferred to the lower level to enhance the efficiency. For example, assume that tuple $t = (1, 3, 1)$ is in $PS(B_m)$ with probability 0.7. To fully realize its pruning potential and improve the overall computational efficiency, we can transfer this tuple to the Pruning Set of bitmap groups with a bitmaps of 110,101,and 011. Similarly, the tuples in each bitmap group's pruning set could also be transferred. To streamline and optimize this transfer characteristic, we give a definition that describes the conditions of the tuple sender and the tuple receiver (see Definition 9). This also means that the receiver bucket is independent of the other higher-level buckets. The receiver bucket only needs to wait for the processes in its sender buckets to be completed, but not the processes in other buckets. The parallel efficiency is improved. For example, the transfer will occur from $B_m.b = 110$ to $B(n).b = 100$, but not $B_p.b = 001$. In this way, the initial Pruning Set of these groups is not empty, which provides a considerable reduction in the number of calculations in the overall process of P-skyline query over uncertain incomplete data.

Algorithm 4 Lower Layers Processing

Input: B_m.
Output: Answer set $PQ(B_m)$.
1: Initialize pruning set $PS(B_m)$
2: Get pruning tuples from related upper layer;
3: Initialize answer set $PQ(B_m)$
4: $SBP(B_m)$;

With the use of transferred pruning tuples, Algorithm 4 is designed as follows: *SBP*() is moved from line 3 to line 22 in Algorithm 3. We give an example in Table 4 to illustrate the transfer mechanism. We obtain $PS(B_{111})$ after applying Algorithm 3. Then, $PS(B_{111})$ can be transferred to $PS(B_{101})$ and $PS(B_{110})$. Buckets B_{101} and B_{110} can be processed in parallel with these pruning sets at this stage. Tuple f is put into $PS(B_{101})$ and output as an answer, while tuple g is pruned by transferred pruning tuple d. The other buckets are processed in a similar way. The final answers of P-skyline in this database under threshold of 0.4 are tuples a, b, d, f and h.

E. INDIVIDUAL BUCKET EFFICIENCY ENHANCEMENT

The processing of individual buckets is serial in Algorithm 3. The parallel efficiency decreases when the processing of some buckets finishes before the processing of others. A typical example is that only one thread is loaded when processing the first bucket. The processing of all the other buckets relies on the result for the first bucket. Therefore, parallelizing the processing of individual buckets is very important.

Definition 10 (Tuple Bus): Tuples in one tuple bus cannot dominate each other, which means they are independent tuples. The capacity of the tuple bus is the number of processor threads.

Lemma 6: Any number of independent tuples can be processed at the same time without changing the final results.

Proof: Assume that tuples a and b are independent tuples. According to Algorithm 3, the sequence of processing influences the generation of the pruning set. Since a and b do not have a dominance relationship, putting one into the pruning set does not affect the other. Therefore, a and b can be processed in any order and Lemma 6 holds. □

Lemma 7: The use of a tuple bus does not affect the final results.

Proof: Tuples in the same tuple bus do not have a dominance relationship. Therefore, according to Lemma 6, Lemma 7 holds. □

TABLE 5. The database parameter settings.

Parameter	Default Value	Variation Range
Database Size	10K	1K∼1M
Dimensionality	3	2∼7
Incomplete Rate	0.3	0.2∼0.6

Before the processing in Algorithm 3, we can obtain individual tuples from the sorted bucket before the full load of the tuple bus. Then, they can be read by each thread and processed in parallel. We give an example to illustrate the operation. Assume that the number of thread is two. The processing of B_{111} in Table 4 involves the processing of buses $\{a, b\}$, $\{d\}$. The tuples in each bus are processed in parallel.

The final algorithm is shown in Algorithm 5. Line 4 ensures the parallel processing according to different platforms. Line 5 to line 8 are the building of bitmap index. Line 10 to line 36 are the main processing flow that are illustrated above. With Algorithm 5, a probabilistic incomplete database can be processed to a set of skyline answers efficiently.

Algorithm 5 Final Algorithm

Input: Probabilistic incomplete database S.
Output: Answer set $PQ(S)$.

1: Initialize $PQ(S)$;
2: Initialize $2^d - 1$ numbers of B_m;
3: Initialize $2^d - 1$ numbers of $PS(B_m)$;
4: Set capacity of a tuple bus to thread count;
5: **while** S is not empty **do**
6: Read an object s from S;
7: Put s into a bucket B_m based on its bitmap;
8: Remove s from S;
9: Monotonic sort each B_m;
10: **for** each B_m^s from higher layer to lower layer **do**
11: **while** tuple bus TB is not full **do**
12: Read a tuple t_p from B_m^s;
13: **if** $TB \not\prec t_p$ **then**
14: $TB = TB \cup t_p$;
15: Discard t_p from B_m^s;
16: **else**
17: Skip t_p;
18: Capture a tuple t_p from TB;
19: **if** $PS(B_m) \not\prec t_p$ **then**
20: **if** $P(t_p) \geq \alpha$ **then**
21: $PQ(B_m) = PQ(B_m) \cup t_p$;
22: Output t_p;
23: $PS(B_m) = PS(B_m) \cup t_p$;
24: **if** $P(t_p) \geq (1 - \alpha)$ **then**
25: Mark t_p as ©;
26: **else if** $\exists$©$t_q \prec t_p$, $t_q \in PS(B_m)$ **then**
27: Discard t_p;
28: Break;
29: **else if** $\exists t_q \prec t_p$, $t_q \in PS(B_m)$, $\wedge t_q$ is not © **then**
30: Calculate $PA(t_p)$;
31: **if** $PA(t_p) \geq (1 - \alpha)$ **then**
32: Mark t_p as ©;
33: Calculate $P_p(t_p)$;
34: **if** $P_p(t_p) \geq \alpha$ **then**
35: $PQ(B_m) = PQ(B_m) \cup t_p$;
36: Output t_p;
37: Transfer $PS(B_m)$ to its subsets;
38: **return** 0

V. EXPERIMENTAL EVALUATION

To evaluate our proposed algorithms, we implement them in C++. The experiments are performed on a PC with an Intel Xeon^{TM} E5-2690 2.9GHz CPU (with 8 cores) and 8GB main memory, under the Ubuntu 14.04 operation system.

A. EXPERIMENTAL SETUP

To generate the synthetic datasets that are used in the experiments, we compile a random database generator program in C++. The generator provides the database with three parameters: database size, dimensionality and incomplete rate. Similar to [5], [23], we use uniform distribution to randomly generate an reliable probability of each tuple to make them be probabilistic uncertain. The reliable probability of each tuple takes a random value between 0 and 1.The parameters settings of the synthetic database are summarized in Table 5.

We also evaluated our algorithms on four real world datasets: *CCarDB*, *HotDB*, *NBA* and *UCarDB*. *CCarDB* is a 6-dimensional database of size 41,424. In our experiments, we consider three numerical attributes of each car: Price, Mileage and Age. *HotDB* contains 10,120 5-dimensional values, which represent the comments and positive review ratios of hotels in Beijing. The attributes contain location, facilities, service, sanitary condition and price inforamtion. These two databases are probabilistic incomplete databases and were obtained by us from two famous e-commerce websites in China. *NBA* contains 17,266 5-dimensional values, which represent the box scores of the basketball players in the National Basketball Association. *UCarDB* is a 2-dimensional dataset of size 1,048,575, which represents used car information in U.S. These two databases have been widely used in many previous works on dominance problems [34]–[40].

The experiments are divided into two parts: In the first part, we compare the performances of four algorithms: *BF*, *IP*, *TS* and *EP*. *BF* is the baseline algorithm, which is described in Section 1. *IP* applies Algorithm 3 to all buckets after grouping and sorting. *TS* applies pruning tuple transfer, as described in Section 4, on *IP*. *EP* is the algorithm that is presented as Algorithm 5. We will find that *EP* has obvious advantages over the other algorithms through comparison. In the second part, examine the performance of *EP* under other conditions.

In the following experiments, we consider the following three aspects as our performance metrics:

- Processing time: the time spent processing the database;
- Progression: the number of output answer tuples in the timer shaft;

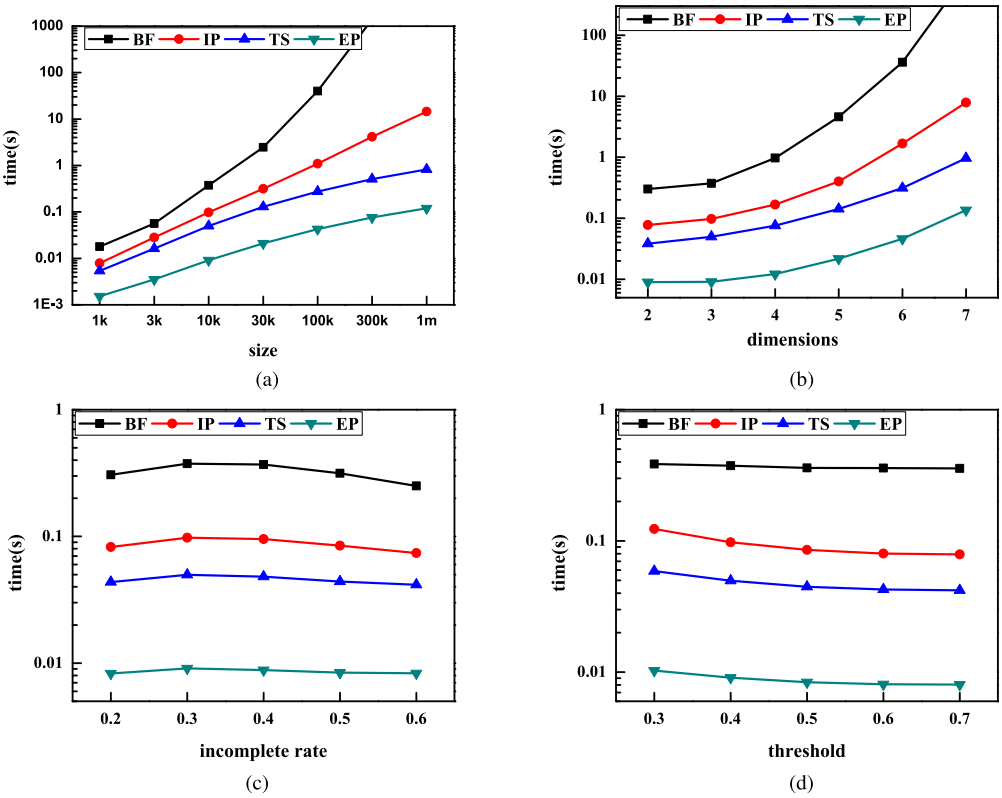


FIGURE 5. Experiment results for synthetic data. (a) Processing time with different database sizes. (b) Processing time with different data dimensions. (c) Processing time with different incomplete rates. (d) Processing time with different threshold values.

- Parallel efficiency: the parallel efficiency of our parallelized algorithm.

B. COMPARISON EXPERIMENTS

To evaluate the performance of our algorithm, a comparison is necessary. However, our algorithm is the first parallel algorithm to tackle P-skyline query over incomplete uncertain databases with progression. Therefore, we design a comparison with the four algorithms that are presented in this paper: BF, *IP*, *TS* and *EP*, which were introduced in the previous subsection.

The performance comparison is the first experiment on a synthetic database. Three parameters are varied in building the synthetic database: the number of tuples in the database, the dimensionality d and the incomplete rate. We change the dimensionality, database size, incomplete rate, and threshold value from $d = 2$ to $d = 7$, $1k$ to $1M$, 0.2 to 0.6, and $\alpha = 0.3$ to $\alpha = 0.7$, respectively. The results are shown in Fig. 5.

As shown in Fig. 5(a), the processing time of each algorithm keeps growing when the database size increases. However, the processing time of *BF* is obviously much larger compared to the other algorithms, regardless of whether the database size is large or small. We find that the time cost of *BF* becomes unacceptable when the database size reaches $300K$. In addition, *IP* has a barely acceptable processing time, according to the figure. For a small database, its processing time is only slightly longer than those of *TS* and *EP*. However, the time cost of *IP* is more than ten times that of *EP* when the database size reaches $100K$. Additionally, the time gap between *IP* and *EP* widens as the size of the database increases. Therefore, both of *BF* and *IP* are unsuitable for processing a large database.

Fig. 5(b) indicates that for all algorithms, higher dimensionality databases require much more processing time than lower dimensional databases. The cost for $d = 7$ is more than ten times that for $d = 2$ in all algorithms. However, *EP* still outperforms the others algorithms. The curve of *EP* is flatter than those of the others, which means it is suitable for high-dimensional databases.

Fig. 5(c) reveals that the incomplete rate has little effect on the processing time of each algorithm. The processing time of each algorithm rises inconspicuously when the incomplete rate is 0.3. All of the four lines are nearly straight.

Similarly, the threshold value has little influence on the processing time of each algorithm. Fig. 5(c) shows that the larger threshold value is, the lower the time cost.

Next, we try to implement all four algorithms on real databases *CCarDB*, *HotDB*, *NBA* and *UCarDB*. The experimental results are shown in Fig. 6. *BF* is unable to handle *UCarDB*. *BF* requires more than one thousand seconds, which is unacceptable. *EP* remarkably outperforms all the other algorithms. *EP* is almost twenty times faster than *IP*

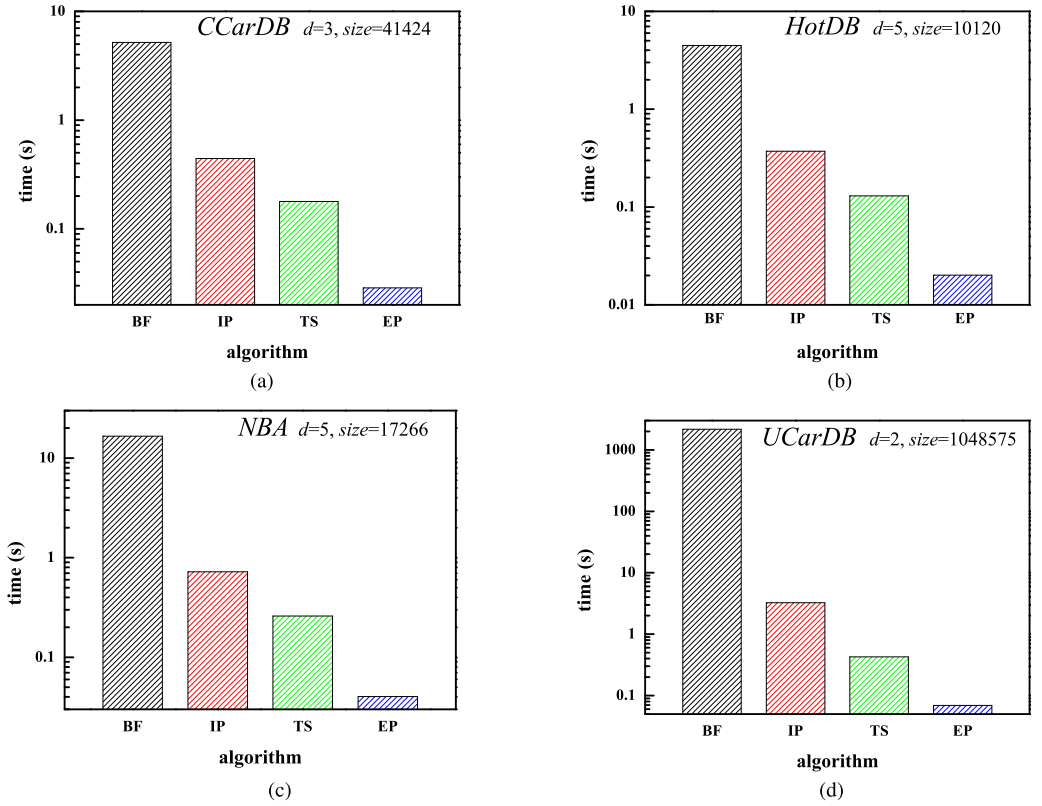


FIGURE 6. Experiment results for real data. (a) Processing time of *CCarDB*. (b) Processing time of *HotDB*. (c) Processing time of *NBA*. (d) Processing time of *UCarDB*.

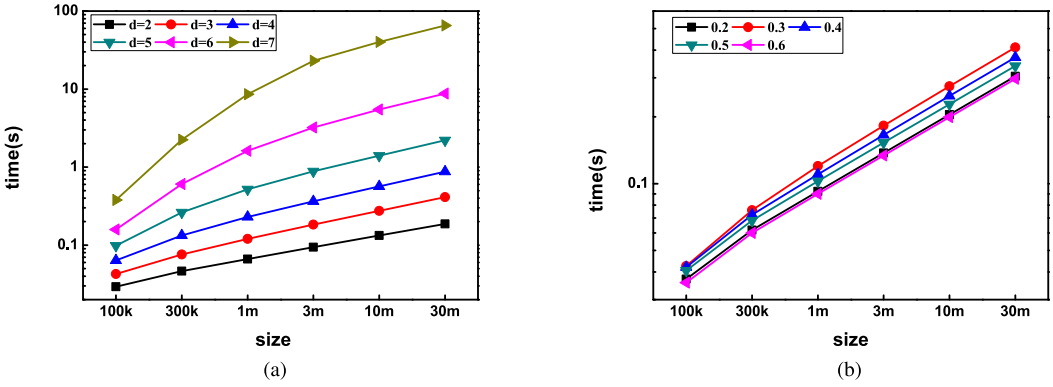


FIGURE 7. Processing time of *EP*. (a) Processing time by dimensions. (b) Processing time by incomplete rates.

on *HotDB* and *NBA*, and it is more than forty times faster that *UCarDB*. Therefore, our algorithm *EP* is also very efficient on real data.

C. FURTHER PERFORMANCE EVALUATION

According to the comparison experiments, our algorithm can output the answer in a quarter of a second, even when the database size is increased to one million. In this section, we try to determine the maximum performance by applying it to a larger and higher-dimensional database. At the same time, the parallel efficiency and progression of *EP* are also tested.

The experiments show that the processing time increases steadily and continuously as the database size increases. The dimensionality of the database also influences the processing time. Experimental results are shown in Fig. 7(a). For a database with a size of 100 K, the processing time varies from 0.029 s to 0.379 s when its dimensionality is varied from 2 to 7. Even when the database size is increased to one million, our algorithm can still return the answer quickly. On a database with 30 M tuples, the processing time is still acceptable. Even for a 7-dimensional database, the result is given in approximately one minutes. Therefore, the time complexity of our algorithm is acceptable.

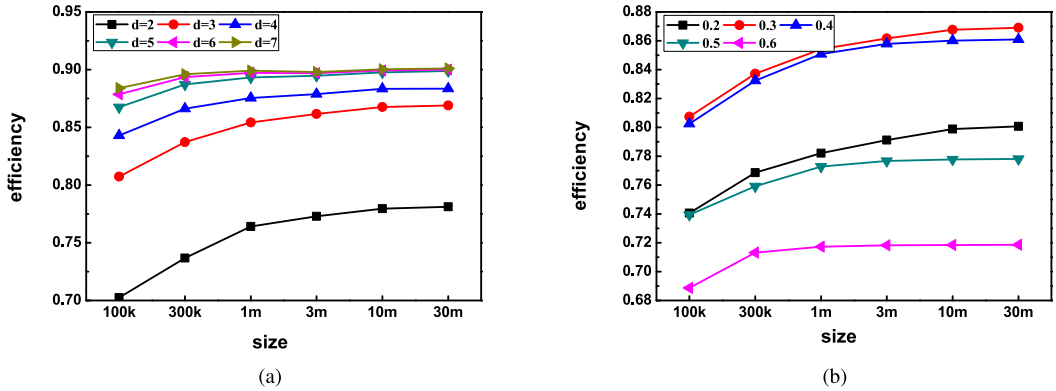


FIGURE 8. Parallel efficiency of *EP*. (a) Parallel efficiency by dimensions. (b) Parallel efficiency by incomplete rates.

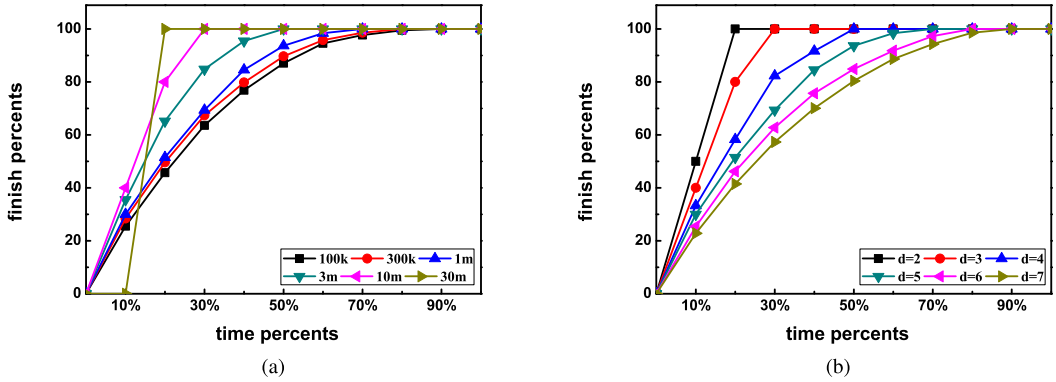


FIGURE 9. Progressive of *EP*. (a) Progressive by sizes. (b) Progressive by dimensions.

The incomplete rate of the database is correlated with the difficulty of processing. In our experiment, 0.3 seems to be the hardest incomplete rate, which is shown in Fig. 7(b). However, the influence of the incomplete rate is comparatively small.

Fig. 8(a) shows the parallel efficiency of *EP*. The y-coordinate implies the parallel efficiency of *EP*. For the Octa-core system, a parallel efficiency of 0.9 means a speedup ratio of 7.2. The parallel efficiency increases as the database size and dimensionality increase. This is very significant because larger and higher-dimensional databases are more time-consuming. Fig. 8(b) indicates that the parallel efficiency is highest when incomplete rate is 0.3, which means that *EP* enjoys better parallel performance on more difficult tasks. Progressive testing is conducted under default settings of $d = 5$ and $size = 1\ M$. The number of answers on a low-dimensional or small database is too small for determining the progression of *EP*. The experimental result is shown in Fig. 9. It illustrates that *EP* achieves better progressive on a larger database. According to Fig. 9(a), *EP* outputs all the results in the first fifth of the processing procedure when $size = 30\ M$, which is very time-saving for users. For a smaller database, users also receive sufficient skyline tuples in a shorter time. Additionally, the processing time on a small database is shorter. Fig. 9(b) implies that at least fifty percent of the final answers can be output in the first third of the processing time. This means that users reveive enough skyline candidates to make decisions before the querying is finished. Therefore, the progression of *EP* is very meaningful.

VI. CONCLUSION

For most problems in market analysis and decision making, P-skyline query over incomplete uncertain data is a very useful tool. In most cases, the number of tuples that need to be processed does not exceed ten millions, and the dimensionality is not more than six. The algorithm that we proposed in this paper has been demonstrated to be efficient and valuable in this range. The processing time is limited to a few seconds. The parallel efficiency of our algorithm is also outstanding when processing high-dimensional large databases, which greatly reduces the time-cost. Additionally, the progression of our algorithm provides sufficient candidates for users to make decision before whole querying process has finished. Therefore, we believe this algorithm has many applications. Future research should examine skyline queries with privacy protection.

REFERENCES

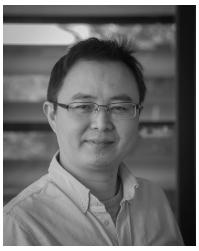
[1] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 15–26.

[2] I. Bartolini, P. Ciaccia, and M. Patella, "SaLSa: Computing the skyline without scanning the whole sky," in *Proc. 15th ACM Int. Conf. Inf. Knowl. Manage.*, 2006, pp. 405–414.

[3] K. C. K. Lee, B. Zheng, H. Li, and W.-C. Lee, "Approaching the skyline in z order," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 279–290.
[4] Y. Wang, Z. Shi, J. Wang, L. Sun, and B. Song, "Skyline preference query based on massive and incomplete dataset," *IEEE Access*, vol. 5, pp. 3183–3192, 2017.
[5] X. Zhou, K. Li, Y. Zhou, and K. Li, "Adaptive processing for distributed skyline queries over uncertain data," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 2, pp. 371–384, Feb. 2016.
[6] Y. M. AbdulAzeem, A. I. Eldesouky, H. A. Ali, and M. M. Salem, "Ranking distributed database in tuple-level uncertainty," *Soft Comput.*, vol. 19, no. 4, pp. 965–980, 2015.
[7] Y.-T. Tsou, Y.-L. Hu, Y. Huang, and S.-Y. Kuo, "SFTopk: Secure functional top-k query via untrusted data storage," *IEEE Access*, vol. 3, pp. 2875–2890, 2015.
[8] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. 17th Int. Conf. Data Eng.*, Apr. 2001, pp. 421–430.
[9] H. Yong, J.-H. Kim, and S.-W. Hwang, "Skyline ranking for uncertain data with maybe confidence," in *Proc. IEEE 24th Int. Conf. Data Eng. Workshop (ICDEW)*, Apr. 2008, pp. 572–579.
[10] M. J. Atallah and Y. Qi, "Computing all skyline probabilities for uncertain data," in *Proc. 28th ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2009, pp. 279–287.
[11] W. Zhang, X. Lin, Y. Zhang, W. Wang, and J. X. Yu, "Probabilistic skyline operator over sliding windows," in *Proc. 25th Int. Conf. Data Eng. (ICDE)*, Mar./Apr. 2009, pp. 1060–1071.
[12] I. Bartolini, P. Ciaccia, and M. Patella, "The skyline of a probabilistic relation," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 7, pp. 1656–1669, Jul. 2013.
[13] L. Antova, C. Koch, and D. Olteanu, "From complete to incomplete information and back," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2007, pp. 713–724.
[14] S. Abiteboul, P. Kanellakis, and G. Grahne, "On the representation and querying of sets of possible worlds," *Theor. Comput. Sci.*, vol. 78, no. 1, pp. 159–187, 1991.
[15] C. C. Aggarwal and P. S. Yu, "A survey of uncertain data algorithms and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 5, pp. 609–623, May 2009.
[16] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: A probabilistic threshold approach," in *Proc. ACM Int. Conf. Manage. Data (SIGMOD)*, 2008, pp. 673–686.
[17] E. Michelakis, R. Krishnamurthy, P. J. Haas, and S. Vaithyanathan, "Uncertainty management in rule-based information extraction systems," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 101–114.
[18] G. Xiao, K. Li, K. Li, and X. Zhou, "Efficient top-(k, l) range query processing for uncertain data based on multicore architectures," *Distrib. Parallel Databases*, vol. 33, no. 3, pp. 381–413, 2015.
[19] X. Zhou, K. Li, G. Xiao, Y. Zhou, and K. Li, "Top k favorite probabilistic products queries," *IEEE Trans, Knowl. Data Eng*, vol. 28, no. 10, pp. 2808–2821, Oct. 2016.
[20] X. Lian and L. Chen, "Reverse skyline search in uncertain databases," *ACM Trans. Database Syst.*, vol. 35, no. 1, 2010, Art. no. 3.
[21] W. Zhang, X. Lin, Y. Zhang, W. Wang, G. Zhu, and J. X. Yu, "Probabilistic skyline operator over sliding windows," *Inf. Syst.*, vol. 38, no. 8, pp. 1212–1233, 2013.
[22] X. Lian and L. Chen, "Efficient processing of probabilistic group subspace skyline queries in uncertain databases," *Inf. Syst.*, vol. 38, no. 3, pp. 265–285, 2013.
[23] X. Ding and H. Jin, "Efficient and progressive algorithms for distributed skyline queries over uncertain data," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 8, pp. 1448–1462, Aug. 2012.
[24] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang, "Finding k-dominant skylines in high dimensional space," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2006, pp. 503–514.
[25] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, "Skyline query processing for incomplete data," in *Proc. IEEE 24th Int. Conf. Data Eng. (ICDE)*, Apr. 2008, pp. 556–565.
[26] P. Haghani, S. Michel, and K. Aberer, "Evaluating top-k queries over incomplete data streams," in *Proc. 18th ACM Conf. Inf. Knowl. Manage.*, 2009, pp. 877–886.
[27] G. Wolf, H. Khatri, B. Chokshi, J. Fan, Y. Chen, and S. Kambhampati, "Query processing over incomplete autonomous databases," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 651–662.
[28] M. A. Soliman, I. F. Ilyas, and S. Ben-David, "Supporting ranking queries on uncertain and incomplete data," *Int. J. Very Large Data Bases*, vol. 19, no. 4, pp. 477–501, 2010.
[29] Y. Gao, X. Miao, H. Cui, G. Chen, and Q. Li, "Processing k-skyband, constrained skyline, and group-by skyline queries on incomplete data," *Expert Syst. Appl.*, vol. 41, no. 10, pp. 4959–4974, 2014.
[30] W. Cheng, X. Jin, J. T. Sun, X. Lin, X. Zhang, and W. Wang, "Searching dimension incomplete databases," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 3, pp. 725–738, Mar. 2014.
[31] C. Lofi, K. El Maarry, and W.-T. Balke, "Skyline queries in crowd-enabled databases," in *Proc. 16th Int. Conf. Extending Database Technol.*, 2013, pp. 465–476.
[32] M. L. Yiu, N. Mamoulis, X. Dai, Y. Tao, and M. Vaitis, "Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 1, pp. 108–122, Jan. 2009.
[33] Y. Wang, X. Li, X. Li, and Y. Wang, "A survey of queries over uncertain data," *Knowl. Inf. Syst.*, vol. 37, no. 3, pp. 485–530, 2013.
[34] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.
[35] W. Zhang, X. Lin, Y. Zhang, J. Pei, and W. Wang, "Threshold-based probabilistic top-k dominating queries," *Int. J. Very Large Data Bases*, vol. 19, no. 2, pp. 283–305, 2010.
[36] E. Tiakas, G. Valkanas, A. N. Papadopoulos, Y. Manolopoulos, and D. Gunopulos, "Metric-based top-k dominating queries," in *Proc. EDBT*, 2014, pp. 415–426.
[37] E. Tiakas, A. N. Papadopoulos, and Y. Manolopoulos, "Progressive processing of subspace dominating queries," *Int. J. Very Large Data Bases*, vol. 20, no. 6, pp. 921–948, 2011.
[38] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos, "Continuous top-k dominating queries," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 5, pp. 840–853, May 2012.
[39] B. J. Santoso and G.-M. Chiu, "Close dominance graph: An efficient framework for answering continuous top-k dominating queries," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1853–1865, Aug. 2014.
[40] Y. Tao, X. Xiao, and J. Pei, "Efficient skyline and top-k retrieval in subspaces," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 8, pp. 1072–1088, Aug. 2007.

YIFU ZENG received the master's degree from the College of Electrical and Electronic Engineering, Nanyang Technological University in 2012. He is currently pursuing the Ph.D. degree with the Department of Information Science and Engineering, Hunan University, Changsha, China. His research interests include parallel computing and data management.

KENLI LI received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, China, in 2003. He was a Visiting Scholar with the University of Illinois at UrbanaChampaign from 2004 to 2005. He is currently a Full Professor of computer science and technology with Hunan University and an Associate Director with the National Supercomputing Center, Changsha, China. He has authored over 160 papers in international conferences and journals, such as the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *Journal of Parallel and Distributed Computing*, *International Conference on Parallel Processing*, and CCGrid. His major research interests include parallel computing, grid and cloud computing, and DNA computing. He is an Outstanding Member of CCF, and is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS.



SHUI YU received the B.Eng. degree in electronic engineering, the Associate degree in mathematics, and M.Eng. degree in computer science from the University of Electronic Science and Technology of China, China, in 1993, 1993, and 1999, respectively, and the Ph.D. degree in computer science from Deakin University, Melbourne, Australia, in 2004. He was a Lecturer with the Computer College, University of Electronic Science and Technology of China. He has a good experience of industry, especially in network design and software development organization and implementation. He is currently a Senior Lecturer with the School of Information Technology, Deakin University. His research interests include big data theory and application, networking theory and application, and mathematical modeling. He dedicates himself in advance human understanding of networks and information, including their measurement, representation, analysis, and application. As a semi-mathematician, he targets on narrowing the gap between theory and application using mathematical tools.

YANTAO ZHOU received the Ph.D. degree in information and electrical engineering from the Wuhan Naval University of Engineering, China, in 2009. He is currently a Professor of electric and information engineering with Hunan University, Changsha. His major research interests include parallel computing and data management.

KEQIN LI (M'90–SM'95–F'15) is currently a Distinguished Professor of computer science with the State University of New York. He has authored or co-authored over 400 journal articles, book chapters, and refereed conference papers. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of Things, and cyber-physical systems. He has received several best paper awards. He is currently or has served on the Editorial Boards of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON CLOUD COMPUTING, and the *Journal of Parallel and Distributed Computing*.