Knowledge-Based Systems 163 (2019) 204–213

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

M-Skyline: Taking sunk cost and alternative recommendation in consideration for skyline query on uncertain data

Yifu Zeng [a], Guo Chen [b,*], Kenli Li [b,c], Yantao Zhou [a], Xu Zhou [b], Keqin Li [b,c,d]

[a] *College of Electrical and Information Engineering, Hunan University, Changsha 410082, Hunan, China*
[b] *College of Information Science and Engineering, Hunan University, Changsha 410082, Hunan, China*
[c] *National Supercomputing Center in Changsha, Changsha 410082, Hunan, China*
[d] *Department of Computer Science, State University of New York, New Paltz, NY 12561, USA*

ARTICLE INFO

Article history:
Received 30 March 2018
Received in revised form 14 August 2018
Accepted 19 August 2018
Available online 30 August 2018

Keywords:
Combinations
Data management
M-Skyline
Probabilistic products
Skyline query

ABSTRACT

Traditional probabilistic skyline query over uncertain data returns a tuple of individual recommendations for customers. However, the uncertainty of the dataset brings the possibility that the recommendation is not correct. Once the incorrect candidate is recommended, user needs to query the skyline again (may use a higher probability threshold) and tries to find alternatives. This greatly hurts user experience for those recommendation scenarios where finding out query results to be wrong brings non-negligible sunk cost, such as spending time to visit a recommended interest point. To address this concern, we propose a novel M-Skyline query model that takes consideration of sunk cost and offers backup recommendation. Moreover, in order to optimize the query speed for finding such M-Skyline results, we devise several fast query algorithms. Extensive experiments with both real and synthetic datasets demonstrate the effectiveness and efficiency of our proposed algorithms under various scenarios.

1. Introduction

1.1. Skyline query and P-Skyline query

The skyline query is a powerful tool for multi-criteria data analysis, data mining, and decision making [1–6]. Given a set of data points with multiple attributes, a skyline query retrieves a set of data points, called skyline tuple. The points in skyline tuple are not dominated by any other data points, *i.e.*, the best choice will come from the skyline tuple for sure in regards of any offered criteria. The following classic example illustrates the skyline query. Suppose a user Tom wants to find a hotel nearby and the price and distance of these hotels are shown in Fig. 1(a). We can see that hotel a, c, and f are candidates for best choices because no hotel has better attributes in all dimensions than any of a, c or f (e.g., b has shorter distance than c, but higher price). Therefore, the result for the skyline query upon this dataset is $\{a, c, f\}$, which forms a skyline tuple.

In reality, however, our datasets often contain uncertainty arising from various causes, such as incomplete survey results, data measure and collection methods, statistical and data mining techniques. To handle skyline query on such uncertain datasets, previous work proposed the P-Skyline model [7], which can get the skyline data points whose probabilities to be the skyline are no smaller than a user-defined probability p. Here gives an example. Assume there are a couple of hotels with price and distance as shown in Fig. 1(b), where the probability of a data point x having the price and distance displayed in the axises is denoted as $P(x)$. Then a P-Skyline query with a threshold probability of 0.7 over this data would return a skyline tuple of $\{a, c\}$. Similarly, when the threshold probability is 0.5, the result is $\{a, c, f\}$.

1.2. P-Skyline query is not enough

While P-Skyline query will return a result with at least p probability to be skyline, it returns no alternative skyline results if we are unlucky to hit the rest $1 - p$ probability. Although such all-or-nothing manner may not be so critical under some scenarios (*e.g.*, NBA player statistics case in [7]), it does greatly hurt user experience for those recommendation scenarios where finding out query results to be wrong brings non-negligible sunk cost. For example, in the previous hotel recommendation scenario, P-Skyline query only returns a set of hotels that are closer to the querying user and/or have lower price with at least a certain probability. However, once a user goes to a hotel and unluckily finds its price is not accurate, he needs to query the skyline again (may use a higher probability threshold) and go to other recommended hotels which may be far away from the current one. More worse, such steps may repeat several times until the user finally gets the right hotel.

* Corresponding author.
E-mail addresses: zengyifu@hnu.edu.cn (Y. Zeng), guochen@hnu.edu.cn (G. Chen), lkl@hnu.edu.cn (K. Li), yantao_z@hnu.edu.cn (Y. Zhou), zhouxu@126.com (X. Zhou), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.knosys.2018.08.024
0950-7051/© 2018 Elsevier B.V. All rights reserved.

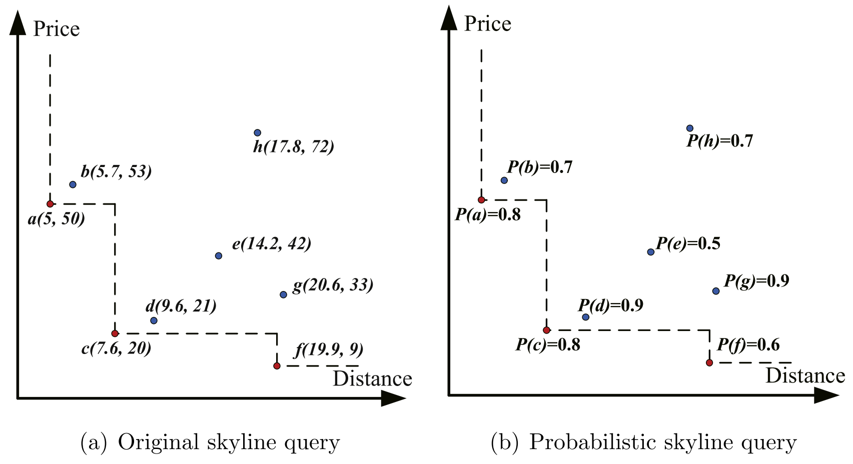


Fig. 1. Skyline query illustration.

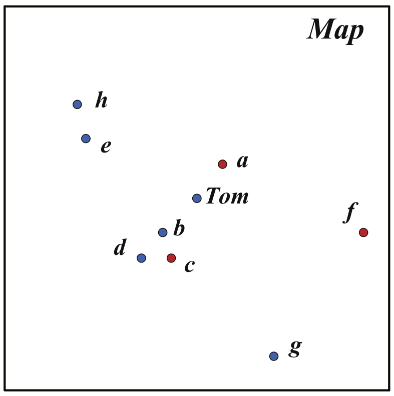


Fig. 2. Nearby map.

1.3. M-Skyline query

Therefore, to deal with aforementioned recommendation scenario, a skyline query that takes consideration of sunk cost and offers backup recommendation would be very helpful in regards of improving user experience. Let us also use the previous hotel recommendation as an example to illustrate. Assume the accurate positions of all hotels and the position of user *Tom* are as shown in Fig. 2. Although hotel a is the skyline with probability of 0.8, hotel $\{b, d\}$ are very close to each other, and also, their distance and price are reasonably close to the skyline as shown in Fig. 1. As such, it may be better to encourage user to visit group $\{b, d\}$ rather than a, since it is not likely that *Tom* does not satisfy with both hotels in the group. However, because other hotels are faraway from a, *Tom* will be hard to get alternative hotels if a is recommended and found to be incorrect.

Thus, in this paper, we propose a new model called M-Skyline query to handle this situation. Specifically, M-Skyline returns skyline groups instead of skyline points, and considers the expected sunk cost and other attributes of the whole group as the criteria to determine the skyline. As such, with M-Skyline, user can choose a group according to the expected sunk cost, and has alternative recommendations within the group. This brings better user experience for recommendation over uncertain datasets.

One more thing to notice is that, group-based M-Skyline query apparently incurs much larger computation cost than original point-based ones. For a probabilistic database with a size of n, the number of possible groups could be $\sum_{m=0}^{n}(n-m)! \times C_n^m$ (details in Section 2). For the example shown in Figs. 2 and 3 which contains eight candidate hotels, it turns into $8!+7!\times C_8^1+6!\times C_8^2+5!\times C_8^3+4!\times C_8^4+3!\times C_8^5+2!\times C_8^6+1!\times C_8^7 = 190,600$ different groups. Obviously, a brute-force method is not suitable for M-Skyline query on any meaningful-size datasets. Therefore, in this work, we also propose several optimized processing algorithms that can greatly reduce the computation complexity of M-Skyline query, and make it computationally practical for real datasets.

1.4. Contribution

In overall, we make the following contributions in this paper.

- We present the M-Skyline model instead of traditional probabilistic skyline that takes sunk cost and alternative recommendation in consideration for skyline query on uncertain data.
- To accelerate the M-Skyline query processing speed, we propose several pruning approaches to reduce the searching space effectively, and further devise several fast M-Skyline query algorithms.
- We perform an extensive experimental study with both synthetic and real datasets to verify the efficiency and effectiveness of our proposed model and algorithms.

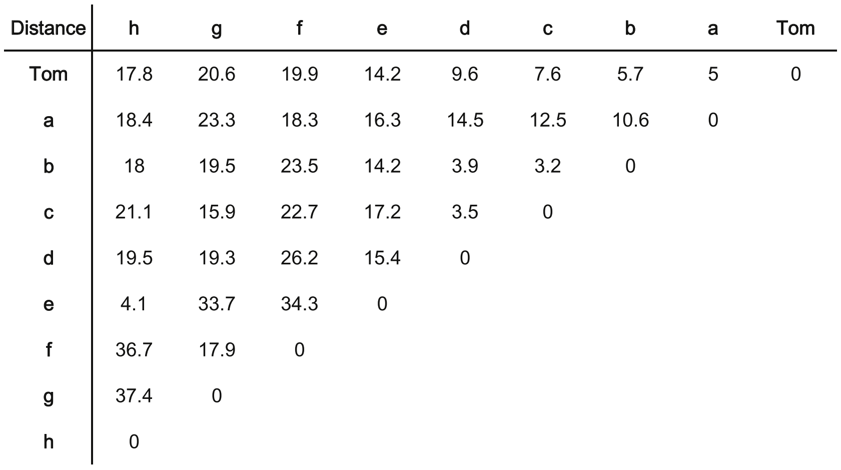
The rest of the paper is organized as follows. M-Skyline and necessary definitions are introduced in Section 2. In Section 3, we design some effective pruning strategies and algorithms to deal with the process of M-Skyline. In Section 4, we evaluate the performance of the proposed algorithm using extensive experiments. Related works are reviewed in Section 5. In Section 6, we conclude the paper and also suggest the directions for future work.

2. M-Skyline query

In this section, we first introduce some basic definitions. Then based on that, we propose the formal definition of M-Skyline model. Table 1 shows the symbols and explanations which are used in this paper.

In M-Skyline we recommend a points group instead of a single point. A recommended group is a group of points that have certain recommendation sequence, *i.e.*, the latter points are the alternative of the former ones. Assuming the i_{th} point in dataset t is denoted as t_i, a recommend group is formally defined as:

Definition 1. A M-Skyline recommended group is a permutation of n points, where n is the number of points in this group. $n > 0$ and $n \leq$ the total size of dataset t.



Distance	h	g	f	e	d	c	b	a	Tom
Tom	17.8	20.6	19.9	14.2	9.6	7.6	5.7	5	0
a	18.4	23.3	18.3	16.3	14.5	12.5	10.6	0	
b	18	19.5	23.5	14.2	3.9	3.2	0		
c	21.1	15.9	22.7	17.2	3.5	0			
d	19.5	19.3	26.2	15.4	0				
e	4.1	33.7	34.3	0					
f	36.7	17.9	0						
g	37.4	0							
h	0								

Fig. 3. Sunk cost of moving between each point (*i.e.*, distance in this example).

Table 1
Symbols and description.

Notation	Description
t_p	The p_{th} point in dataset t, starting from 1
$t_p \prec t_q$	t_p dominates t_q
$t_p \not\prec t_q$	t_p does not dominate t_q
P_i	The probability of the i_{th} point in a recommended group having all its attributes equal to the value as recorded in the dataset
P_{t_p}	The probability of point t_p having all its attributes equal to the value as recorded in the dataset
ξ	The disappointment probability of a recommended group
$D_{i,j}$	The extra cost for moving to the j_{th} point after trying the i_{th} point in a recommended group. $D_{0,1}$ refers to the initial cost of trying the first point in a recommended group
D_{t_i,t_j}	The extra cost for moving to point t_j after trying point t_i. D_{0,t_j} refers to the initial cost of trying point t_j in the database
D^+	The expected sunk cost of a recommended group
N_i^k	Value of the k_{th} attribute of the i_{th} point in a recommended group
$N_{t_i}^k$	Value of the k_{th} attribute of point t_i in the database
N^{k+}	The expected value of the k_{th} attribute of a recommended group
α	The threshold of disappointed possibility
$M(S)$	The M-Skyline answers of database S

Note that the order of points matters. The same points can be different groups if they have different orders of permutation.

In P-Skyline model, any point t_i has a probability of P_{t_i} to have all its attributes equal to the values recorded in the given datasets. So similarly, we need a metric to characterize the probability of a M-Skyline group to have its points' attributes equal to the values recorded. As such, we define a metric ξ, whose formal definition is as follows. Assume a recommended group contains n points.

Definition 2. ξ implies the possibility of no point in a recommended group having its attributes equal to the recorded values. ξ is calculated by $\xi = \prod_{i=1}^{n}(1 - P_i)$.

P_i is the probability of the i_{th} point in a recommended group having all its attributes equal to the value as recorded in the dataset. Assuming the group's i_{th} point is t_j, then $P_i = P_{t_j}$. The lower ξ, the lower chance that none of the points in this group has the recorded value. Intuitively, ξ describes the probability that a whole group is not satisfying for the user, so we call it the *disappointment probability*.

Now we define a metric D^+ to describe the expected sunk cost of a recommended group. Specifically, assuming the user finally finds a point in this group with all its attributes equal to the values recorded in the dataset (denoted as the *right* point), D^+ describes how much cost it will take to find this point in expectation. We define $D_{i,j}$ as the extra cost for moving to the j_{th} point after trying the i_{th} point, where $D_{0,1}$ refers to the initial cost of trying the first point. As such, assuming a group has n points, we have the following model of calculating D^+:

Definition 3. D^+ describes the expected sunk cost of a recommended group, which is calculated by

$$D^+ = \sum_{m=1}^{n-1}[P_m \times (\prod_{p=1}^{m-1}(1 - P_{(p)})) \times (\sum_{q=0}^{m-1} D_{q,q+1})] + \sum_{q=0}^{n-1} D_{q,q+1} \times \prod_{p=1}^{n-1}(1 - P_p)$$

In overall, D^+ is the sum of two parts. The first part is the expected sunk cost of finding one of the first $n-1$ points to be the right one, which equals the summation of the expected sunk cost of reaching point 1 to $n-1$ and stop (got the right one), respectively. The second part is the cost of finding the n_{th} point to be the right one. Note that since D^+ is calculated with the precondition of assuming the user finally finds a right point in this group, we do not need to multiply P_n in the second part.

Now we define a metric N^{k+} to describe the expected attribute value of a recommended group. Assume a group has n points. Similar as D^+, assuming the user finally finds a right point in this group:

Definition 4. N^{k+} describes the expected value of the k_{th} attribute that the user gets from a recommended group, which is calculated by

$$N^{k+} = \sum_{m=1}^{n-1}[P_m \times \prod_{p=1}^{m-1}(1 - P_p) \times N_m^k] + N_n^k \times \prod_{p=1}^{n-1}(1 - P_p)$$

In overall, N^{k+} is the sum of two parts. The first part is the expected value of the k_{th} attribute of finding one of the first $n-1$ points to be the right one, which equals the summation of the expected attribute value of reaching point 1 to $n-1$ and stop (got the right one), respectively. The second part is the expected attribute value of finding the n_{th} point to be the right one. Note that since N^{k+} is calculated with the precondition of assuming the user finally finds a right point in this group, we do not need to multiply P_n in the second part.

Let us use an example in Fig. 2 to show ξ, D^+ and N^{k+} of a recommended group. Since it has only one attribute in this example, N^{k+} is N^{1+} and N_a^k is N_a^1. They represent the expected price of the group and the price of a. Assume the sunk cost of moving between each point (*i.e.*, distance in this case) is given by Fig. 3. Therefore, for the group containing a single point $\{a\}$, its $\xi = 0.2$, $D^+ = D_{0,a} = 5$ and $N^{1+} = N_a^1 = 50$. However, for the group $\{a, c\}$, its $\xi = 0.04$, $D^+ = P(a) \times D_{0,a} + (D_{0,a} + D_{a,c}) \times (1 - P(a)) = 7.5$, $N^{1+} = P(a) \times N_a^1 + N_c^1 \times (1 - P(a)) = 44$.

With Definitions 3 and 4, the dominance relationship between two groups, can be defined as Definition 5. Assume there are K attributes for each point in this dataset.

Definition 5 (*Group Dominance Relationship*)**.** For two groups G_1 and G_2, G_1 dominates G_2 (denoted by $G_1 \prec G_2$), if and only if, $\forall k$ (k=1, ...,K) $N^{k+}(G_1) \leq N^{k+}(G_2)$ and $D^+(G_1) \leq D^+(G_2)$, and $\exists k (k = 1, \dots, K)\ N^{k+}(G_1) < N^{k+}(G_2)$ or $D^+(G_1) < D^+(G_2)$.

Now, we define the M-Skyline query as follows,

Definition 6 (*M-Skyline Query*)**.** The M-Skyline query returns skyline groups, that their $\xi \leq \alpha$, and not dominated by any other group. α is a user-defined disappointment probability threshold.

We give an example to illustrate M-Skyline query. For the database which is shown in Figs. 2 and 3, the result M-Skyline groups with $\alpha = 0.05$ are $\{c, d\}$ and $\{b, c, d\}$. The first recommendation group has the best expected price guarantee with acceptable expected distance and the second recommendation group has the shortest expected distance with acceptable expected price. The detail of the computation is illustrated in Section 3.

3. Processing algorithms to M-Skyline query

Group-based M-Skyline query apparently incurs much larger computation cost than original point-based ones. For a probabilistic database with a size of n, the number of possible groups could be $\sum_{m=0}^{n}(n-m)! \times C_n^m$. That is the permutation and combination of groups contains n tuples, plus groups contains $n - 1$ tuples, and so on.

For the example shown in Figs. 2 and 3 which contains eight candidate hotels, it turns into 190,600 different groups. Obviously, a brute-force method is not suitable for M-Skyline query on any meaningful-size datasets. Therefore, in this section, we propose several optimized algorithms that can accelerate M-Skyline query.

First, we begin with a baseline algorithm.

3.1. Baseline algorithm

A straightforward algorithm can be put forward according to the definition of M-Skyline. It is realized by exhaustively combining all tuples in the database and then all possible groups can be got. Subsequently, groups that $\xi > \alpha$ will be eliminated. Then, D^+ and N^{k+} of all groups will be computed. After that, it can be processed just like traditional probabilistic skyline query. However, it is too costly. As illustrated in Section 1, even a small database will extend to a great deal of groups. Obviously, it is unpractical. Therefore, we need to impose restrictions on the extending of unnecessary possible groups.

With Definition 2, it is noticeable that only the groups that $\xi \leq \alpha$ are eligible to be candidate groups. On the contrary, groups with $\xi > \alpha$ are ineligible to be candidate groups before extending. We define two different states of groups in extending process by Definitions 8 and 7. Briefly, root groups (Definition 7) are waiting to be extended to many different complete groups, while complete groups (Definition 8), are extended from root groups and they can be considered as complete schemes to compare with each other.

Definition 7 (*Root Group*)**.** For any sequential tuples combination, $\{t_1, t_2\}$, if $\xi(\{t_1, t_2\}) > \alpha$, it needs to be extended before eligible to be a recommendation group. It is the root group of its possible extensions, such as $\{t_1, t_2, t_3\}$ and $\{t_1, t_2, t_4\}$.

Definition 8 (*Complete Group*)**.** For any sequential tuples combination, $\{t_1, t_2\}$, if $\xi(\{t_1, t_2\}) \leq \alpha$, it is a complete group, which is eligible to be a recommendation group without further extending.

For any root group G_i, it does not reach the requirement of users, so that the calculation of its expected sunk cost D^+ and expected attribute value N^{k+} is meaningless in the processing of M-Skyline. On the other hand, the D^+ and N^{k+} of complete group G_j can be counted during the processing of M-Skyline. In this way, many unnecessary groups are not generated and the spatial and non-spatial characteristics of many ineligible groups are not calculated. After all the groups are generated, they can be processed as certain skyline query. The *OSPSPF* algorithm in [8] is the most efficient algorithm for certain data skyline query. Therefore, it can be used in our baseline algorithm to provide better performance. Obviously, the baseline algorithm with these designs is more efficient than the straightforward algorithm.

Algorithm 1 Baseline Algorithm

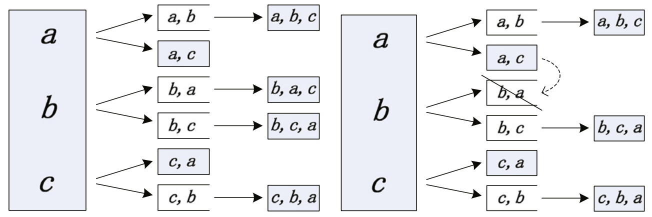
```
Input: d-dimensional probabilistic data set $S$.
Output: $M(S)$.
1: Initialize a tuple group for each tuples;
2: for each group $G_i$ do
3:     if $\xi(G_i) \leq \alpha$ then
4:         count $N^{k+}$ and $D^+$ for $G_i$;
5:     else
6:         extend $G_i$ to different groups $G_{ij}$;
7: Process skyline query for all $G_{ij}$ with OSPSPF algorithm;
```

The baseline algorithm can be illustrated briefly by a dataset that includes $\{a, b, c\}$. The ξ of each tuple can be calculated according to Fig. 2, which is $\xi(a) = 0.2$, $\xi(b) = 0.3$, $\xi(c) = 0.2$. Since all of them are larger than α, they are root groups and extending is required. Then we have six groups, which are $\{a, b\}$, $\{a, c\}$, $\{b, a\}$, $\{b, c\}$, $\{c, a\}$, $\{c, b\}$. Since $\xi(\{a, b\}) > \alpha$, it is still a root group and needs to be extended to $\{a, b, c\}$. On the contrary, $\xi(\{a, c\}) < \alpha$, so that it is a complete group and $N^{1+}(\{a, c\})$ and $D^+(\{a, c\})$ can be counted. In this way, the M-Skyline answer of database $\{a, b, c\}$ can be given.

3.2. Inter-group structure optimization

Candidate groups that are generated in Algorithm 1 are still a big challenge for processing. It will enhance the efficiency of M-Skyline query if we properly filter some root groups before they extended to complete groups. In order to realize the optimization, a reasonable processing flow is required. Fig. 4(a) shows the processing flow of Algorithm 1.

As illustrated in Algorithm 1, all root groups will extend to complete groups. Afterwards, the characteristics of all complete groups will be counted. However, some root groups are unnecessary to be extended to complete groups since they have inferior characteristics. The number of complete groups can be reduced if we can prune some root groups before extending. In order to find out these inferior root group, we can use a different way to calculate the attributes of root groups. For the calculation of a complete group, Definitions 4 and 3 are applicable, since tuples in group are fixed. However, for a root group, other different tuples are possible to join the group so that the calculation of its attributes cannot be finished. For example, $\{b, a\}$ is a root group, since $\xi(\{b, a\}) > \alpha$.



(a) Group processing in baseline algorithm (b) Pruning of inferior root group

Fig. 4. Processing flow optimizing.

Only transient states of its attributes can be counted, which are $D^+\{b,a\} = P(a)\times D_{0,a} + (1-P(a))\times P(b)\times(D_{0,a}+D_{a,b}) = 7.902$ and $N^{1+}\{b,a\} = P(a)\times N_a^1 + (1-P(a))\times P(b)\times N_b^1 = 49.1$, respectively.

With the transient states attributes of a root group, we have Lemmas 1 and 2, to greatly reduce the generation of complete groups.

Lemma 1. *A root group dominates its extended groups.*

Proof. Assume that a root group G_i can be extended to a complete group G_j with tuple t_j and t_i is the last tuple in G_i. It can be deduced that $D^+(G_j) = D^+(G_i) + P\times D_{t_i,t_j}$. Since $0 < P < 1$ and $D_{t_i,t_j} > 0$, so that $D^+(G_j) > D^+(G_i)$. In the same way, $N^{k+}(G_j) > N^{k+}(G_i)$. Therefore, $G_i \prec G_j$. □

Lemma 2. *If a complete group G_i dominates a root group G_j, G_j can be safely pruned without affecting the result of M-Skyline query.*

Proof. According to Lemma 1, G_j dominates all its extension groups. G_i is a complete group, which means the calculation of its characteristics are finished. Therefore, G_i dominates G_j and all of its extension groups and the lemma holds. □

As shown in Fig. 4(b), since $D^+\{a,c\} = 7.5$ and $N^{1+}\{a,c\} = 44$, so that $\{a,c\} \prec \{b,a\}$. According to Lemma 2, $\{b,a\}$ can be pruned and the processing efficiency is improved.

In order to further reduce the generation of complete groups, we can focus on the superior tuples. Some tuples have not only better attributes but also higher reliable possibilities than other tuples. We should utilize these superior tuples to prune some relative complete groups before calculating group attributes. Superior tuples full dominant some other tuples, which is defined in Definition 9.

Definition 9 (*Full Dominance Relationship*)**.** Tuple t_i full dominates tuple t_j ($t_i \prec\prec t_j$), if $t_i \prec t_j$ and $P(t_i) \geq P(t_j)$.

Full dominance relationship is different with dominance relationship. The stricter condition helps to prune more groups when processing M-Skyline. Imagine that hotel a has not only better characteristics but also higher reliable possibility than b, it seems that going to a rather than b at first is reasonable. Lemma 3 provides the conditions to meet the imagination. With Lemma 3, some groups consist of same tuples but have different sequence can be pruned.

Lemma 3. *Complete group $\{t_j, t_i\}$ can be pruned by complete group $\{t_i, t_j\}$ when $t_i \prec\prec t_j$ and $P(t_j) + P(t_i) \geq 1$.*

Proof. Assume that $t_i \prec\prec t_j$ and $P(t_j) + P(t_i) \geq 1$. According to Definition 3, $D^+(\{t_i,t_j\}) = P(t_i)\times D_{0,t_i} + (1-P(t_i))\times D_{0,t_i} + D_{t_i,t_j} = D_{0,t_i} + D_{t_i,t_j} - P(t_i)\times D_{t_i,t_j}$. Similarly, $D^+(\{t_j,t_i\}) = D_{0,t_j} + D_{t_j,t_i} - P(t_j)\times D_{t_j,t_i}$. $D^+(\{t_j,t_i\}) - D^+(\{t_i,t_j\}) = D_{0,t_j} - D_{0,t_i} + D_{t_i,t_j}\times(P(t_i)-P(t_j))$. Since $t_i \prec\prec t_j$, $D_{0,t_j} - D_{0,t_i} \geq 0$ and $P(t_i) - P(t_j) \geq 0$. So that $D^+(\{t_j,t_i\}) \geq D^+(\{t_i,t_j\})$. According to Definition 4, $N^{k+}(\{t_i,t_j\}) = P(t_i)\times N_{t_i}^k + (1-P(t_i))\times N_{t_j}^k$ and $N^{k+}(\{t_j,t_i\}) = P(t_j)\times N_{t_j}^k + (1-P(t_j))\times N_{t_i}^k$. $N^{k+}(\{t_j,t_i\}) - N^{k+}(\{t_i,t_j\}) = (N_{t_j}^k - N_{t_i}^k)\times(P(t_i)+P(t_j)-1)$. Since $t_i \prec\prec t_j$, $N_{t_j}^k - N_{t_i}^k \geq 0$. Since $P(t_j)+P(t_i) > 1$, $(P(t_i)+P(t_j)-1) \geq 0$. So that $N^{k+}(\{t_j,t_i\}) \geq N^{k+}(\{t_i,t_j\})$. Since $D_{0,t_j} = D_{0,t_i}$ and $N_{t_j}^k = N_{t_i}^k$ are impossible to be true at the same time, we have $\{t_i,t_j\} \prec \{t_j,t_i\}$ and this lemma holds. □

With Lemmas 2 and 3, Algorithm 3 is designed below. As shown in Algorithm 3, three data pools, including root group pool, match pool and complete group pool are created in processing. These pools are designed to ensure an efficient orders when extending root groups. The monotonic sorting of the database S is a pre-processing method. Algorithm 3 topologically pre-sort the data based on a monotone function, which requires that if tuple t_i precedes t_j in the order, then t_j cannot dominate t_i.

In addition, Algorithm 2 is a part of Algorithm 3. Algorithm 2 describe the process flow when new extended groups are produced. When a new root group is created, it should be checked that if it is dominated by any complete group. Similarly, when a new complete group is created, it should be checked that if it is dominated by any other complete group and if it dominates any other root group. In this way, lots of root groups are pruned before extending and both of the root group pool and the complete group pool are kept in the minimum size.

Algorithm 2 New Extended Group Processing (*NEGP*)

```
Input: New Extended Group $G_i$.
1:  if $\xi(G_i) \leq \alpha$ then
2:      Put $G_i$ into complete group pool;
3:      if any group in complete group pool dominate $G_i$ then
4:          Prune $G_i$;
5:      if complete group $G_i$ dominate any root group in root group
        pool then
6:          Prune the dominated root groups;
7:  else
8:      Put $G_i$ into root group pool;
9:      if $G_i$ is dominated by any group in complete group pool then
10:         Prune $G_i$.
```

As an example of Algorithm 3, let us consider tuples a, b, c, d, e that shown in Fig. 1. The sequence of these tuples is c, d, a, b, e after sorting. First of all, c is picked from database. Since $\xi(c) > \alpha$, root group $\{c\}$ is created and tuple c is put into match pool. The second tuple is d. As illustrated in line 4, group $\{c,d\}$ is created. Since $\xi(\{c,d\}) \leq \alpha$, it is a complete group. Then, root group $\{d\}$ and complete group $\{d,c\}$ are created. However, $\{c,d\}$ dominates $\{d,c\}$ so that the complete group pool still has only one group, $\{c,d\}$. Match pool has two tuples, c and d. When b is picked into processing flow, the root group $\{b,a\}$ is pruned, since it is dominated by complete group $\{a,c\}$. Tuple e is full dominated by c and d. According to Lemma 3, $\{e,c\}$ and $\{e,d\}$ are pruned directly. Finally, only two groups are left in complete group pool, which are $\{c,d\}$ and $\{b,c,d\}$. They are the M-Skyline answer groups.

Algorithm 3 perform well before line 17. However, too much groups are created when executing line 18 and most of them are inferior groups. In order to enhance the efficiency of M-Skyline query, a new pruning rule is presented in Section 3.3.

3.3. Optimization in root group extending

In order to enhance the efficiency of executing line 18 in Algorithm 3, the location information of all tuples should be utilized

Algorithm 3 Inter-Group Pruning (*IGP*)

```
Input: d-dimensional probabilistic data set $S$.
Output: $M(S)$.
1: Monotonic sort $S$;
2: for All tuples in $S$ do
3:   Pick $t_i$ from $S$;
4:   Extend all root groups in root group pool with $t_i$;
5:   NEGP($G_i$);
6:   if $\xi(t_i) \le \alpha$ then
7:     Put $\{t_i\}$ into complete group pool;
8:     if any group in complete group pool dominate $\{t_i\}$ then
9:       Prune $\{t_i\}$;
10:    else
11:      if complete group $\{t_i\}$ dominate any root group in root group pool then
12:        Prune the dominated root groups;
13:  else
14:    if any group in complete group pool dominate root group $\{t_i\}$ then
15:      Prune $\{t_i\}$;
16:    else
17:      Put root group $\{t_i\}$ into root group pool;
18:    Extend root group $\{t_i\}$ with all tuples in match pool;
19:    if any tuple $t_j$ in match pool full dominates $t_i$ and $P(t_i) + P(t_j) \ge 1$ then
20:      Prune group $\{t_i, t_j\}$ directly;
21:    NEGP($G_j$);
22:  Put $t_i$ into match pool;
23: Return all groups in complete group pool.
```

more efficient. In reality, the location of each tuple is planar. Therefore, the dominance relationship of location information should be planar as well. As shown in Fig. 5, for Tom, t_1 has a location dominance area and a reverse dominance area. For any tuple t_i in location dominance area and any tuple t_j in reverse dominance area, $D_{t_1,t_j} < D_{t_i,t_j}$. This rule is very important and it will be utilized to design a pruning rule soon afterwards.

The full dominance rules in Section 3.2 considers distance from user to each tuple. In this section, only non-spatial values of tuples are considered when judging full dominance relationships. Spatial characteristics and non-spatial characteristics are considered separately in this section. It provides more precise information in the design of pruning rules. With Lemma 4, much less groups will be created when executing line 18 in Algorithm 3.

Lemma 4. *For any tuple t_j in location reverse dominance area of t_1 and any tuple t_i in location dominance area of t_1, $\{t_j, t_1\} \prec \{t_j, t_i\}$ if $t_1 \prec\prec t_i$.*

Proof. Assume that t_j is in the location reverse dominance area of t_1 and tuple t_i is in the location dominance area of t_1 and $t_1 \prec\prec t_i$. According to Fig. 5, $D_{t_1,t_j} < D_{t_i,t_j}$. Since $D^+(\{t_j, t_i\}) - D^+(\{t_j, t_1\}) = P(t_j) \times D_{0,t_j} + (1 - P(t_j)) \times (D_{0,t_j} + D_{t_j,t_i}) - P(t_j) \times D_{0,t_j} - (1 - P(t_j)) \times (D_{0,t_j} + D_{t_j,t_1}) = (1 - P(t_j)) \times (D_{t_j,t_i} - D_{t_j,t_1}) > 0$, so that $D^+(\{t_j, t_i\}) > D^+(\{t_j, t_1\})$. Since $N^{k+}(\{t_j, t_i\}) - N^{k+}(\{t_j, t_1\}) = P(t_j) \times N^k_{t_j} + (1 - P(t_j)) \times (N^k_{t_i}) - P(t_j) \times N^k_{t_j} - (1 - P(t_j)) \times (N^k_{t_1}) = (1 - P(t_j)) \times (N^k_{t_i} - N^k_{t_1}) > 0$, so that $N^{k+}(\{t_j, t_i\}) > N^{k+}(\{t_j, t_1\})$. Therefore, $\{t_j, t_1\} \prec \{t_j, t_i\}$ and the lemma holds. □

Since Lemma 4 relays on full dominance relationship, a fast query algorithm of dominance relationship between tuples is valuable. In the judgment of full dominance relationship, probabilistic tuples are processed as certain tuples. Therefore, the LCRS tree [8] can be used to perform high efficiency full dominance judgment.

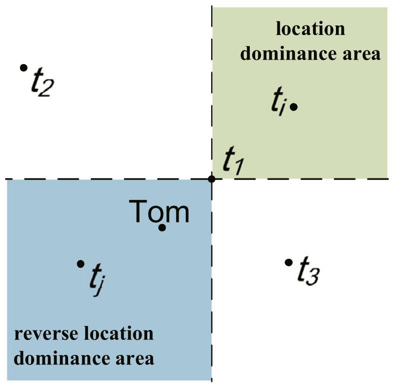


Fig. 5. Dominance area and reverse dominance area.

Once LCRS tree is built, the full dominance relationship between any two tuples can be given by checking locating partition address instead of repetitive computation. Moreover, a prepositional tuple in LCRS tree is impossible to be full dominated by a postpositional tuple, which is similar to the monotonic sorting in Algorithm 3. Therefore, Algorithm 4 is given as follows.

Algorithm 4 Extending Pruning of Groups (*EGP*)

```
Input: d-dimensional probabilistic data set $S$.
Output: $M(S)$.
1: LCRS($S$)
2: for All tuples in $S$ do
3:   Pick $t_i$ from $S$;
4:   Extend all root groups in root group pool with $t_i$;
5:   NEGP($G_i$);
6:   if $\xi(t_i) \le \alpha$ then
7:     Put $\{t_i\}$ into complete group pool;
8:     if any group in complete group pool dominate $\{t_i\}$ then
9:       Prune $\{t_i\}$;
10:    else
11:      if complete group $\{t_i\}$ dominate any root group in root group pool then
12:        Prune the dominated root groups;
13:  else
14:    if any group in complete group pool dominate root group $\{t_i\}$ then
15:      Prune $\{t_i\}$;
16:    else
17:      Put root group $\{t_i\}$ into root group pool;
18:    if $t_i$ is located in the reverse location dominance area of any $t_j$ in match pool then
19:      Find all $t_k$ which is located in the location dominance area of $t_j$;
20:      Extend root group $\{t_i\}$ with all tuples in match pool except $t_k$;
21:    else
22:      Extend root group $\{t_i\}$ with all tuples in match pool;
23:    NEGP($G_j$);
24:  Put $t_i$ into match pool;
25: Return all groups in complete group pool.
```

The improvement from Algorithm 3 to Algorithm 4 includes the utilization of LCRS tree and the pruning strategies in extending root group with tuples in match pool. As illustrated in Section 3.2, too much groups are created when executing line 18 of Algorithm 3. With the new pruning method, the number of extended groups is substantially declined in Algorithm 4. Moreover, LCRS tree helps to

Table 2
System parameter settings.

Parameter	Default value	Variation range
Dimensionality	5	4∼7
Database size	300	100∼10k
Threshold	0.05	0.09∼0.01

reduce the repetitive computation of full dominance relationship. Therefore, the processing time of Algorithm 4 is reduced sharply.

4. Performance evaluation

In this section, we verify the efficiency of our proposed algorithms. Our experiments use both the synthetic datasets and two real-life datasets. We analyze three aspects on the factors, which are dimensionality of the datasets d, the threshold α, and the number of tuples n. All the experiments are conducted on a PC with Intel® Xeon^{TM} E5-2667 3.3 GHz CPU (contains eight cores), 16 GB main memory, and under the Ubuntu 15.04 operation system. All algorithms were implemented in C++.

4.1. Experimental highlights

- M-Skyline query over varieties parameters database can be finished by Algorithm *EGP* in a few seconds.
- Real databases are captured from websites. M-Skyline query over two real databases can be finished by Algorithm *EGP* in acceptable waiting time.

4.2. Experiments on synthetic datasets

In order to study the scalability of the proposed algorithms, we first experiment on the synthetic datasets with three popular distributions: Independent (Ind), Correlated (Cor) and Anti-correlated (Ant). Specifically, for the Ind dataset, all attribute values are generated independently using a uniform distribution; for the Cor and Ant dataset, if some point is good in one dimension, and it tends to be good and bad in all of the other dimensionality(s), respectively. Similar to [9,10], we use uniform distribution to randomly generate an existential probability of each tuple to make them be uncertain. The existential probability of each tuple takes a random value between 0 and 1. In consideration of the reality that hotels and restaurants are unlikely to have a excessively low positive rates, such as 0.1, the median of the existential probability is set to 0.8, which is a typical value in reality.

In this subsection, we report the performance of three algorithms, *BA* (Algorithm 1), *IGP* (Algorithm 3), *EGP* (Algorithm 4) over one dataset by varying the dimensionality of datasets d, the disappointed probabilistic threshold α, and the number of tuples n. Unless specifically stated, each other parameter is set to its default, which is shown in Table 2. It is important to note that two dimensions of the dimensionality d is location information. All tuples has a two dimensional spatial characteristics and $d-2$-dimensional non-spatial characteristics. It is noticeable that even for $n = 10$, the number of possible group combinations is 9,864,100, which is computationally intensive. The variation range of n is from 100 to 10k. For reference, the number of hotels in New York and Beijing on Booking.com is 443 and 910, respectively.

4.2.1. Performance vs. dimensionality d

In the first set of experiments, we study the performance of *BA*, *IGP* and *EGP* with d varying from 4 to 7 by a step of 1, and the other parameters are kept to their default values.

The efficiency of the algorithms under various d is depicted in Fig. 6, where Ind, Cor, and Ant are reported, respectively.

As expected, the performance of the three algorithms, *BA*, *IGP* and *EGP*, all degrade sharply with the growth of d. It is because in a high-dimensional space, each tuple has a low probability of being dominated by other ones. In the same way, each group also has a low probability to be dominated, which makes the final skyline set become larger. More query answers lead to higher bandwidth cost. However, *EGP* performs much better than the others with different dimensionality. The time differences between *EGP* and the other two algorithms get larger with the increase of dimensionality.

As shown in Fig. 6(a), (b), and (c), considering the Ind, Cor, and Ant, the performance of both *IGP* and *EGP* are obviously influenced, while *BA* is almost not influenced. It is because both of *IGP* and *EGP* rely on finding some dominating complete groups to prune root groups as soon as possible. It is easier for these two algorithms to establish groups with stronger dominance ability early. Tuples in Ant database are not easy to be dominated by other tuples. Therefore, similar to high-dimensional database, it increase the computing difficulty for *IGP* and *EGP*.

4.2.2. Performance vs. database size n

In the second set of experiments, we examine the effect of the number of database size n on the performance of the M-Skyline query. The number of database size n varies from 100 to 10k by a step of threefold and the other parameters kept to their default values.

As illustrated in Fig. 7, *EGP* performs much better than the other two algorithms. *EGP* provides not only shortest processing time but also flattest growth rate when compared with *IGP* and *BA*. This is because *EGP* establishes some complete group with strong dominance ability in a short time. Many root groups are pruned before they are extended to complete groups. In addition, since LCRS tree is built in the beginning, the checking of full dominance relationship in *EGP* is easier than that in *IGP*. Moreover, tuples that picked in later period are dominated by many other tuples, which means any root groups with these tuples are highly possible to be dominated by some complete groups in complete group pool. Therefore, the processing time of *EGP* increases gently.

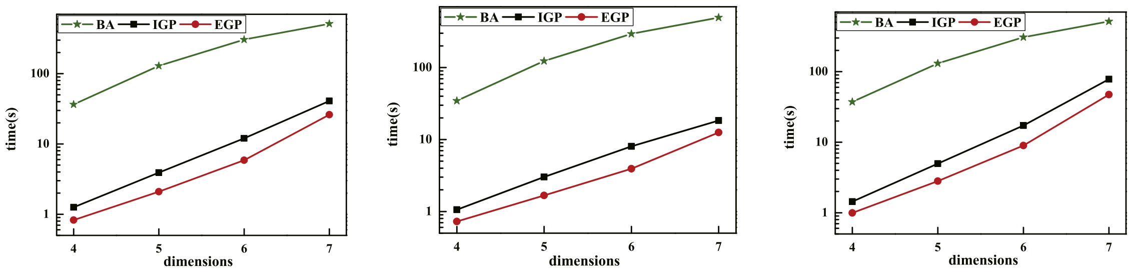
Considering Cor and Ant, when n becomes larger, their influence get larger. As illustrated in Fig. 7(b) and (c), the query times of *IGP* and *EGP* with $n = 10$k are around 20 vs. over 100, and around 5 vs. around 20. It depicts that the efficient processing of a large database requires early establishment of some complete groups with strong dominance ability, which is also illustrated in Section 4.2.1.

4.2.3. Performance vs. threshold α

In the third set of experiments, we explore the impact of threshold α on the performance of the algorithms. Specifically, α varies from 0.01 to 0.09 by a step of 0.02, and the other parameters are kept to their default values.

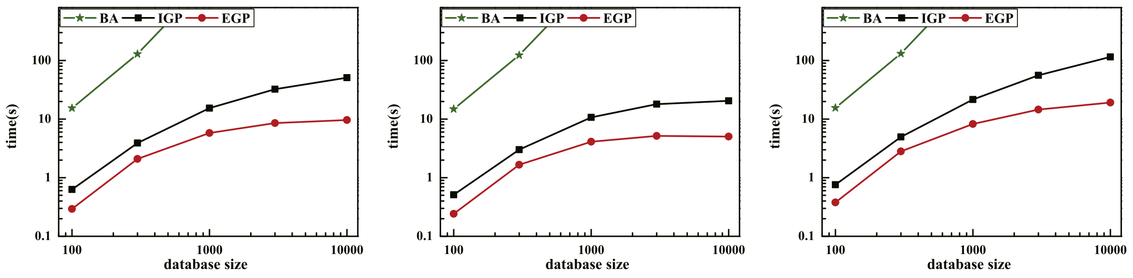
Fig. 8 illustrates the experimental results when we vary α from 0.01 to 0.09, under the Ind, Cor, and Ant distributions. As α increases, the performance of the three algorithms both become better. The reason is the size of the qualified combinations of tuples is sensitive to the probability threshold α. According to Definitions 2 and 6 in Section 2, the larger the probability threshold, the fewer the complete groups. It means more computation and procedures are required in extending a root group to complete groups. That is the reason of all of the three algorithm are influenced by the threshold α. In addition, it is even harder to establish a complete group with strong dominance ability under a small threshold value. Therefore, the query time reduces as α raises.

Since the influence of adjusting the *alpha* is similar to the correlatedness and anti-correlatedness of the database, the differences between Fig. 8(b) and (c) are smaller than that of Figs. 6 and 7. The processing times of *IGP* and *EGP* are around 2s and 1s in Ind, Cor, and Ant.



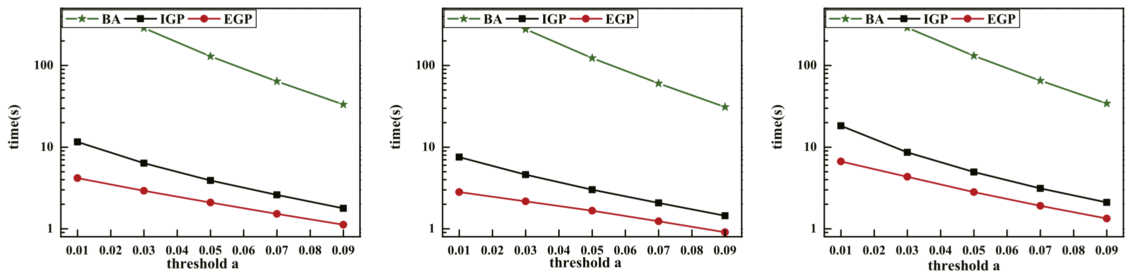
(a) Independent (b) Correlated (c) Anti-correlated

Fig. 6. Query times by dimensions *d*.



(a) Independent (b) Correlated (c) Anti-correlated

Fig. 7. Query times by database size *n*.



(a) Independent (b) Correlated (c) Anti-correlated

Fig. 8. Query times by threshold α.

4.3. Experiments on real datasets

In this section, we evaluate our proposed algorithms over two real dataset, CarDB and HotDB. Specifically, CarDB includes 45,311 6-dimensional points [11,12]. We consider two numerical attributes, price and mileage, of 4500 cars. Due to lack of uncertainty and location information in this dataset, similar to [13,12], we randomly generate the existential probability and location information of each point. It is a simulation of selecting used cars from the market. HotDB contains 1120 6-dimensional values, which represent the comments and positive ratios of hotels in Beijing. It is notable that the two-dimensional location information is acquired from the most popular map provider, map.baidu.com. The non-spatial attributes include facilities, service, sanitary condition, and price, which are acquired from ctrip.com. Clearly, HotDB is a probabilistic database with spatial information. It is a simulation of selecting hotel in Beijing.

BA is unable to handle these two databases so that only *IGP* and *EGP* are tested. The experimental results of these two databases are shown in Fig. 9. The results demonstrate *EGP* performs better than *IGP* in real databases. *CarDB* is similar to a anti-correlated database since a car with lower mileage tends to be more expensive. *EGP* can handle it with around three seconds while *IGP* costs near nine seconds. *HotDB* is similar to a correlated database since a good hotel tends to be good at not only service, facility, but also sanitary condition. *EGP* and *IGP* costs around 22 s and 82 s, respectively. Comparing with Fig. 7(b), *EGP* costs more time with *HotDB*, since *HotDB* is $d = 6$ rather than $d = 5$. It demonstrates that the dimensionality greatly influence the processing time. In summary, *EGP* is also very efficient for real data.

5. Related work

5.1. Skyline queries over certain data

Most of the previous researches focused on how to answer the query in a computationally efficient way. The approaches for processing skyline queries over certain data can be classified into two categories, which are non-index based approaches and index based approaches [14,15]. The first category involves solutions that do not utilize index to organize the databases. It mainly includes Block Nested Loop (BNL), Sort Filter Skyline (SFS), Sort and Limit Skyline Algorithm (SaLSa), ZSearch, and Objectbased Space Partitioning (OSP), et al. The OSP in [8] is considered to be the most efficient approach without index. The other category contains solutions which

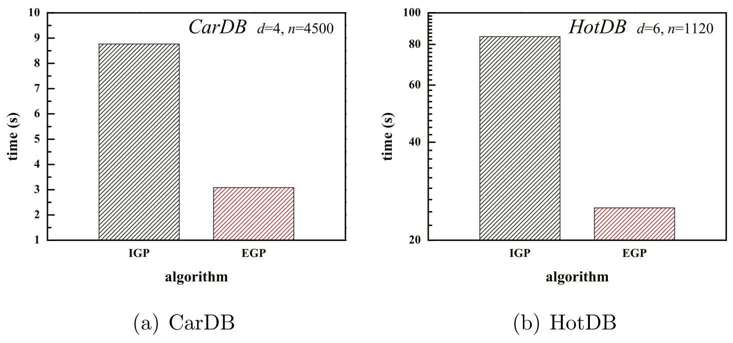


Fig. 9. Query times of real databases.

utilize index to accelerate the skyline query. The approaches in the second category utilize index structures such as R-tree, and ZB-tree to organize the databases. Some representative approaches based on index include a Nearest Neighbor (NN) algorithm, a Branch and Bound (BBS) algorithm, and a ZB-tree algorithm. The BBS algorithm based on R-tree is progressive and acknowledged to be I/O optimal.

In addition to the traditional skyline query, many skyline query variants have also been studied in the literatures. The variants include distributed skyline query [16], reverse skyline query [14], monochromatic and bichromatic mutual skyline queries [15], subspace skyline query [17], reverse k-skyband and ranked reverse skyline query [11], top-k skyline query [18], group skyline [19], to name just a few. In [20], they focused on the skyline query over big data.

For all of these approaches are geared toward certain data, they cannot be employed to process uncertain data directly.

5.2. Skyline queries over uncertain data

Uncertain data grows rapidly with the increasing popularity of applications such as data integration, scientific and sensor databases. Query processing over uncertain datasets has become an important research topic in the database community, such as uncertain skyline queries [21–25], uncertain top-k queries [26,27], location-based queries (LBS) [28,29], recommendation systems based on uncertain datasets [30,31], etc. The first study about skyline query over uncertain datasets, namely P-Skyline, was reported by Pei et al. [7]. It has been developed to return tuples whose skyline probabilities are larger than a specified probabilistic threshold. This pioneering work has inspired a number of follow-up studies. Ding and Jin proposed the notation of distributed skyline queries over uncertain data and designed two computation-efficient and communication-efficient algorithms [9]. Lian and Chen proposed a novel and important query in uncertain databases, the probabilistic group subspace skyline query (PGSS), and presented an efficient query procedure [32]. Lian and Chen focused on the probabilistic reverse skyline, considering both monochromatic and dichromatic cases [33]. Zhang et al. studied the problem of efficiently computing skylines against sliding windows over an uncertain data stream [34]. Those previous studies are mostly based on P-Skyline. The appropriate user-specified probability threshold is one of the challenges for the P-Skyline.

Different from P-Skyline, a new skyline query for uncertain databases [35], called U-Skyline, is developed. It reports a set of tuples having the highest probability. The processing of U-Skyline query is an NP-hard problem and in the worst case it needs to exhaust all the subsets of an uncertain database.

6. Conclusions and further study

Traditional skyline queries over uncertain database are unable in providing alternative answers. The potential sunk cost is also neglected by traditional skyline queries. In this paper, we first propose the M-Skyline query to select groups of sequential tuples that provides recommendations that takes consideration of sunk cost and alternative choices. An efficient algorithm, *IGP* is introduced to process the M-Skyline over uncertain database in acceptable time. In *IGP*, several efficient technologies, including complete group pruning and reverse location dominance area, are employed. Moreover, extensive experiments have been conducted to clarify the effectiveness and the efficiency of our algorithms.

As for our future work, on the one hand, considering the incomplete databases are common in reality, we will study M-Skyline query over incomplete databases. On the other hand, we are going to study approximate M-Skyline query algorithms that return quickly a good approximation of results. In particular, with the explosion of Internet information content, it requires query algorithms to report results in time. We plan to extend our approaches to parallel and distributed query processing algorithms for the future work.

Acknowledgments

The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant No. 61472126, 61432005), the National Outstanding Youth Science Program of National Natural Science Foundation of China (Grant No. 61625202), the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124), the International Science & Technology Cooperation Program of China (Grant Nos. 2015DFA11240, 2014DFB30010), and the National High-tech R&D Program of China (Grant No. 2015AA015305).

References

[1] S Borzsony, Donald Kossmann, Konrad Stocker, The skyline operator, in: Data Engineering, 2001. Proceedings. 17th International Conference on, IEEE, 2001, pp. 421–430.
[2] Ilaria Bartolini, Paolo Ciaccia, Marco Patella, Efficient sort-based skyline evaluation, ACM Trans. Database Syst. 33 (4) (2008) 31.
[3] Jongwuk Lee, Seung-Won Hwang, Scalable skyline computation using a balanced pivot selection technique, Inf. Syst. 39 (2014) 1–21.
[4] Ken CK Lee, Wang-Chien Lee, Baihua Zheng, Huajing Li, Yuan Tian, Z-sky: an efficient skyline query processing framework based on z-order, VLDB J. 19 (3) (2010) 333–362.
[5] Yunjun Gao, Qing Liu, Lu Chen, Gang Chen, Qing Li, Efficient algorithms for finding the most desirable skyline objects, Knowl.-Based Syst. 89 (2015) 250–264.
[6] Xu Zhou, Yantao Zhou, Guoqing Xiao, Yifu Zeng, Fei Zheng, Effective approach for an extended p-skyline query, J. Intell. Fuzzy Syst. 31 (2) (2016) 849–858.

[7] Jian Pei, Bin Jiang, Xuemin Lin, Yidong Yuan, Probabilistic skylines on uncertain data, in: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB Endowment, 2007, pp. 15–26.
[8] Shiming Zhang, Nikos Mamoulis, David W Cheung, Scalable skyline computation using object-based space partitioning, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, ACM, 2009, pp. 483–494.
[9] Xiaofeng Ding, Hai Jin, Efficient and progressive algorithms for distributed skyline queries over uncertain data, IEEE Trans. Knowl. Data Eng. 24 (8) (2012) 1448–1462.
[10] Xu Zhou, Kenli Li, Yantao Zhou, Keqin Li, Adaptive processing for distributed skyline queries over uncertain data, IEEE Trans. Knowl. Data Eng. 28 (2) (2016) 371–384.
[11] Yunjun Gao, Qing Liu, Baihua Zheng, Li Mou, Gang Chen, Qing Li, On processing reverse k-skyband and ranked reverse skyline queries, Inform. Sci. 293 (2015) 11–34.
[12] Xu Zhou, Kenli Li, Guoqing Xiao, Yantao Zhou, Keqin Li, Top *k* favorite probabilistic products queries, IEEE Trans. Knowl. Data Eng. 28 (10) (2016) 2808–2821.
[13] Wenjie Zhang, Xuemin Lin, Ying Zhang, Wei Wang, Jeffrey Xu Yu, Probabilistic skyline operator over sliding windows, in: Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on, IEEE, 2009, pp. 1060–1071.
[14] Yunjun Gao, Qing Liu, Baihua Zheng, Gang Chen, On efficient reverse skyline query processing, Expert Syst. Appl. 41 (7) (2014) 3237–3249.
[15] Tao Jiang, Yunjun Gao, Bin Zhang, Dan Lin, Qing Li, Monochromatic and bichromatic mutual skyline queries, Expert Syst. Appl. 41 (4) (2014) 1885–1900.
[16] Lin Zhu, Yufei Tao, Shuigeng Zhou, Distributed skyline retrieval with low bandwidth consumption, IEEE Trans. Knowl. Data Eng. 21 (3) (2009) 384–400.
[17] David Fuhry, Ruoming Jin, Donghui Zhang, Efficient skyline computation in metric space, in: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, ACM, 2009, pp. 1042–1051.
[18] Yufei Tao, Xiaokui Xiao, Jian Pei, Efficient skyline and top-k retrieval in subspaces, IEEE Trans. Knowl. Data Eng. 19 (8) (2007) 1072–1088.
[19] Hyeonseung Im, Sungwoo Park, Group skyline computation, Inform. Sci. 188 (2012) 151–169.
[20] Xixian Han, Jianzhong Li, Donghua Yang, Jinbao Wang, Efficient skyline computation on big data, IEEE Trans. Knowl. Data Eng. 25 (11) (2013) 2521–2535.
[21] Yan Wang, Zhan Shi, Junlu Wang, Lingfeng Sun, Baoyan Song, Skyline preference query based on massive and incomplete dataset, IEEE Access 5 (2017) 3183–3192.
[22] Yu Won Lee, Ki Yong Lee, Myoung Ho Kim, Efficient processing of multiple continuous skyline queries over a data stream, Inform. Sci. 221 (2013) 316–337.
[23] Yijie Wang, Xiaoyong Li, Xiaoling Li, Yuan Wang, A survey of queries over uncertain data, Knowl. Inf. Syst. 37 (3) (2013) 485–530.
[24] Hyountaek Yong, Jongwuk Lee, Jinha Kim, Seung-won Hwang, Skyline ranking for uncertain databases, Inform. Sci. 273 (2014) 247–262.
[25] Yifu Zeng, Kenli Li, Shui Yu, Yantao Zhou, Keqin Li, Parallel and progressive approaches for skyline query over probabilistic incomplete database, IEEE Access (2018).
[26] Guoqing Xiao, Kenli Li, Keqin Li, Reporting l most influential objects in uncertain databases based on probabilistic reverse top-k queries, Inform. Sci. 405 (2017) 207–226.
[27] Guoqing Xiao, Kenli Li, Xu Zhou, Keqin Li, Efficient monochromatic and bichromatic probabilistic reverse top-k query processing for uncertain big data, J. Comput. System Sci. 89 (2017) 92–113.
[28] Tao Jiang, Bin Zhang, Dan Lin, Yunjun Gao, Qing Li, Incremental evaluation of top-k combinatorial metric skyline query, Knowl.-Based Syst. 74 (2015) 89–105.
[29] Xin Lin, Jianliang Xu, Haibo Hu, Wang-Chien Lee, Authenticating location-based skyline queries in arbitrary subspaces, IEEE Trans. Knowl. Data Eng. 26 (6) (2014) 1479–1493.
[30] Zui Zhang, Hua Lin, Kun Liu, Dianshuang Wu, Guangquan Zhang, Jie Lu, A hybrid fuzzy-based personalized recommender system for telecom products/services, Inform. Sci. 235 (2013) 117–129.
[31] Dianshuang Wu, Guangquan Zhang, Jie Lu, A fuzzy preference tree-based recommender system for personalized business-to-business e-services., IEEE Trans. Fuzzy Syst. 23 (1) (2015) 29–43.
[32] Xiang Lian, Lei Chen, Efficient processing of probabilistic group subspace skyline queries in uncertain databases, Inf. Syst. 38 (3) (2013) 265–285.
[33] Xiang Lian, Lei Chen, Reverse skyline search in uncertain databases, ACM Trans. Database Syst. 35 (1) (2010) 3.
[34] Wenjie Zhang, Aiping Li, Muhammad Aamir Cheema, Ying Zhang, Lijun Chang, Probabilistic n-of-n skyline computation over uncertain data streams, in: Web Information Systems Engineering–WISE 2013, Springer, 2013, pp. 439–457.
[35] Xingjie Liu, De-Nian Yang, Mao Ye, Wang-Chien Lee, U-skyline: a new skyline query for uncertain databases, IEEE Trans. Knowl. Data Eng. 25 (4) (2013) 945–960.