Top k Favorite Probabilistic Products Queries

Xu Zhou, Kenli Li, *Senior Member, IEEE*, Guoqing Xiao, Yantao Zhou, and Keqin Li, *Fellow, IEEE*

Abstract—With the development of the economy, products are significantly enriched, and uncertainty has been their inherent quality. The probabilistic dynamic skyline (PDS) query is a powerful tool for customers to use in selecting products according to their preferences. However, this query suffers several limitations: it requires the specification of a probabilistic threshold, which reports undesirable results and disregards important results; it only focuses on the objects that have large dynamic skyline probabilities; and, additionally, the results are not stable. To address this concern, in this paper, we formulate an uncertain dynamic skyline (UDS) query over a probabilistic product set. Furthermore, we propose effective pruning strategies for the UDS query, and integrate them into effective algorithms. In addition, a novel query type, namely the top k favorite probabilistic products (TFPP) query, is presented. The TFPP query is utilized to select k products which can meet the needs of a customer set at the maximum level. To tackle the TFPP query, we propose a TFPP algorithm and its efficient parallelization. Extensive experiments with a variety of experimental settings illustrate the efficiency and effectiveness of our proposed algorithms.

Index Terms—Data management, dynamic skyline query, parallel algorithm, probabilistic products

1 Introduction

With the development of the economy, we have entered the era of product explosion. Amazon.com, for example, offers 236,080 types of wrist watches, 41,063 types of grape wines, and 1,046,681 types of milk. Moreover, probabilistic products, which customers can obtain with certain probabilities, are being given increasing attention [1]. Similar to the specialized probabilistic products that companies provide, the most common products on an on-line shopping platform are also uncertain, and can be considered probabilistic products. These products are associated with favorable rating values derived from historical customer feedback. The rating value of each product can be considered its existential probability because it represents the probability that the product matches the description in the advertisement [2]. As a result, it brings a great challenge for customers, who must process massive numbers of probabilistic products to identify the items that satisfy their preferences.

To identify products that can meet a customer's demands, there are mainly two approaches. One approach is to execute a dynamic skyline query in view of the customer [3]. Another approach, as seen in [4] and [5], allows customers to specify the worst acceptable value of each attribute. A product p is said to satisfy customer c if all of the attribute values of p are not worse than the worst acceptable values, respectively. The above two approaches are useful for processing traditional products with certainty. However, these processes are not applicable when handling probabilistic products because they do not account for the uncertainty of each candidate product. Moreover, one serious limitation of the second approach is that it fails to handle subjective types of attributes [3].

In [6], Lian et al. formulated a probabilistic reverse skyline query based on a dynamic skyline query over uncertain data. To find the probabilistic reverse skylines, it is necessary to first execute the probabilistic dynamic skyline (PDS) query. In [7], Lian et al. researched the probabilistic group subspace skyline query, which is also based on the PDS query. The PDS query retrieves the objects whose dynamic skyline probabilities are not less than the specified probabilistic threshold. Specifically, a data object p dominates another object p' in view of a query point q, if it holds that (1) $|p[i] - q[i]| \leq |p'[i] - q[i]|$ for all dimensions $1 \leq i \leq d$; and (2) it has at least one dimension j in which $|p[j] - q[j]| < |p'[j] - q[j]|$, where $p[j]$ and $p'[j]$ are the jth dimensions of objects p and p', respectively. This PDS query is easy to adjusted and utilize to help find probabilistic products that satisfy a customer's requirements.

Consider an on-line trading system that has a grape wine set $W = \{w_1, w_2, w_3, w_4, w_5, w_6\}$ for sale, as illustrated in Table 1. We consider two attributes of each wine, which are the grape juice content (GraCon) and price (Pri). For ease of description, we use $1-$GraCon instead of GraCon to describe the grape content in this paper. Without loss of generality, for $1-$GraCon and Pri, the lower values are considered preferable. Furthermore, as shown in Table 1, each grape wine has a rating (Rat), which indicates the probability that the product matches the description in the advertisement in terms of quality. Table 2 provides a customer set $C = \{c_1, c_2, c_3\}$, which describes the customers' different preferences for the grape wine.

Fig. 1 shows an example of the PDS query with respect to the customer c_1. Based on [6], in view of c_1, because no wine

- *X. Zhou and K. Li are with the College of Information Science and Engineering, Hunan University, the National Supercomputing Center in Changsha, Hunan, Changsha 410082, and the Collaborative Innovation Center of High Performance Computing, National University of Defense Technology, Changsha 410073, China. E-mail: zhouxu2006@126.com, kl@hnu.edu.cn.*
- *G. Xiao and Y. Zhou are with the College of Information Science and Engineering, Hunan University, and the National Supercomputing Center in Changsha, Hunan, Changsha 410082. E-mail: {xiaoguoqing, yantao_z}@hnu.edu.cn.*
- *K. Li is with the College of Information Science and Engineering, Hunan University, the National Supercomputing Center in Changsha, Hunan, Changsha 410082, the Collaborative Innovation Center of High Performance Computing, National University of Defense Technology, Changsha 410073, China, and the Department of Computer Science, State University of New York, New Paltz, NY 12561. E-mail: lik@newpaltz.edu.*

Manuscript received 15 June 2015; revised 9 May 2016; accepted 14 June 2016. Date of publication 23 June 2016; date of current version 7 Sept. 2016. Recommended for acceptance by X. Lin.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2016.2584606

1041-4347 © 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

TABLE 1
The Grape Wines

Wines	1-GraCon(%)	Pri($)	Rat
w_1	40	70	0.90
w_2	20	90	0.80
w_3	60	170	0.40
w_4	30	220	0.50
w_5	90	190	0.70
w_6	70	80	0.60

TABLE 2
The Preferences of Customers

Customers	1-GraCon(%)	Pri($)
c_1	55	130
c_2	35	110
c_3	70	140

dynamically dominates w_3, the dynamic skyline probability of w_3 is equal to 0.40. Considering that w_1 is dynamically dominated by w_3 and w_6, its dynamic skyline probability is $0.90 \times (1 - 0.40) \times (1 - 0.60) = 0.22$. Similarly, we obtain the dynamic skyline probabilities of w_2, w_4, w_5 and w_6, which are 0.48, 0.03, 0.04 and 0.36, respectively. As a consequence, given a probabilistic threshold $\alpha = 0.35$, the PDS query returns a set $\{w_2, w_3, w_6\}$, where the dynamic skyline probability of each wine is not less than 0.35, as depicted in Fig. 1b. Fig. 1c shows that if the probabilistic threshold α is set to 0.45, the PDS query reports a set $\{w_2\}$.

From Fig. 1, it is clear that the PDS query has several limitations. First, it is necessary to specify a threshold α, which is difficult for customers without guidelines. Second, the PDS query may obtain some undesirable objects that have no advantages in terms of either their attributes or dynamic skyline probabilities. For example, w_6 is considered an undesirable result in view of c_1 because w_6 is dynamically dominated by w_3 and its dynamic skyline probability is also less than that of w_6. Additionally, the PDS query may overlook some important objects. As shown in Fig. 1c, the wine w_3, which has the best attributes, is not returned to the customer because the dynamic skyline probability of w_3 is equal to 0.40, which is slightly lower than the threshold 0.45. Finally, the PDS query results change with different probabilistic thresholds. Fig. 1 shows that it returns a set $\{w_2, w_3, w_6\}$ with $\alpha = 0.35$ and reports a set $\{w_2\}$ with $\alpha = 0.45$.

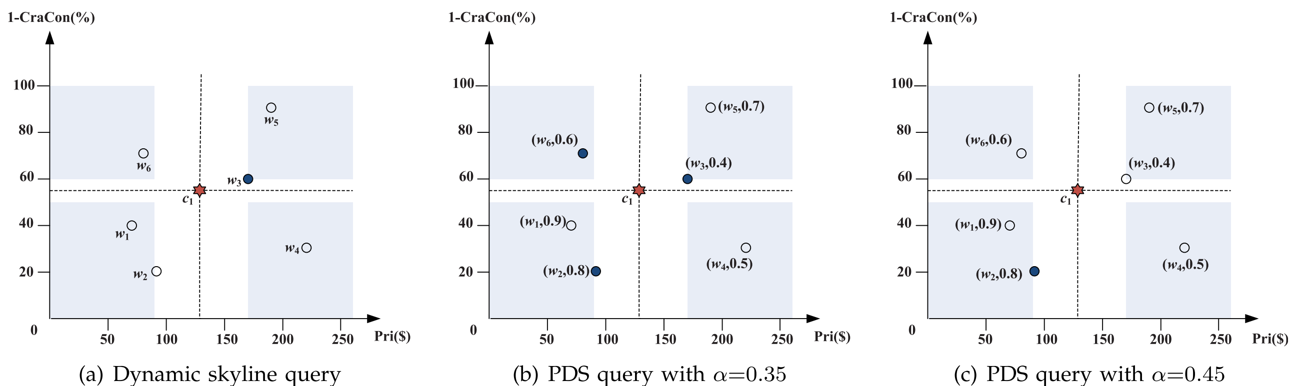
In this paper, we are concerned with the limitations that the PDS query faces, and propose a new dominant operator: uncertain dynamic dominance (UD-Dominance). To obtain products with higher satisfaction levels, in the UD-Dominance operator, we account for both the attributes and the dynamic skyline probabilities of the products. A product p UD-dominates another product p' in view of a customer c if it holds that (1) p is not worse than p' in both all the attributes and the dynamic skyline probability, and p is better than p' in at least one attribute; or (2) p is not worse than p' in all attributes and the dynamic skyline probability of p is larger than that of p'. Moreover, we develop an uncertain dynamic skyline (UDS) query. Given a probabilistic product set UP and a customer set c, the UDS query reports all of the products $p \in UP$ that are not UD-dominated by any other $p' \in UP$ in view of c .

In the UDS query, it only takes a customer's preference into consideration, whereas many applications need to consider the various preferences of a group of customers. For instance, an online shopping mall plans to implement a sales strategy. It is important to maximize the profits of the mall by selecting some "best" products that can satisfy a set of customers according to their different preferences. This essentially describes another useful query that concerns UDS queries with a size constraint and in view of different customers. To concern this, we define this query operator as a top k favorite probabilistic products (TFPP) query. This TFPP query reports the k products that have the highest favorite probabilities.

The TFPP query is also important in many other applications. In a general scenario, consider a family that wants to book a flight. Every family member has his/her own preferences, including price, boarding time, and flight duration. In this application, the TFPP query can also help select the flights that are most suitable for this family.

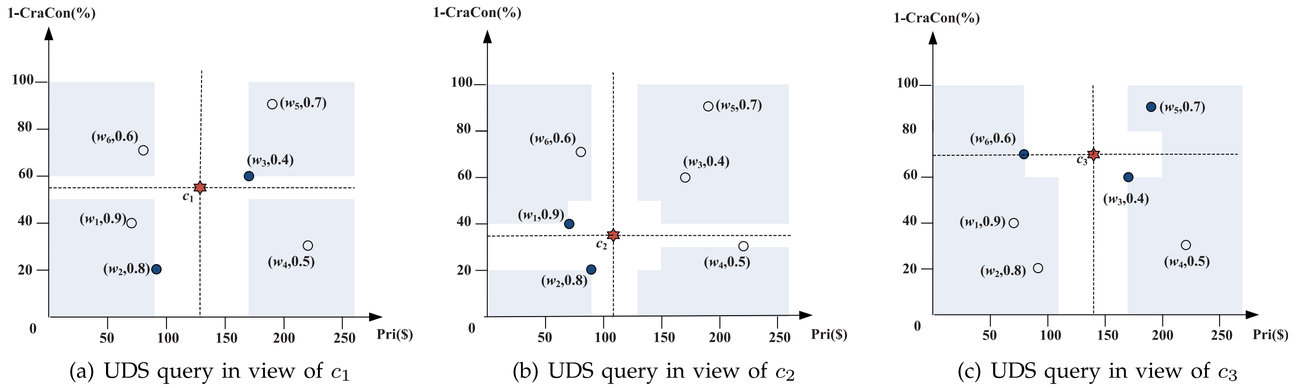
Briefly, the significant contributions of this paper are summarized as follows.

- We formulate an uncertain dynamic skyline query in the context of a probabilistic product set.
- We propose several pruning approaches to reduce the search space of the UDS query, and integrate them into a UDS query (UDSQ) algorithm. Moreover, we introduce the reuse mechanism to accelerate the UDS query and develop an enhanced UDSQ (EUDSQ) algorithm.



(a) Dynamic skyline query (b) PDS query with α=0.35 (c) PDS query with α=0.45

Fig. 1. The illustration of PDS queries with respect to c_1.



(a) UDS query in view of c_1 (b) UDS query in view of c_2 (c) UDS query in view of c_3

Fig. 2. The illustration of UDS queries.

- We develop a top k favorite probabilistic products query, which is concerned with the preferences of a customer set.
- We design an efficient approach to address the TFPP query. Furthermore, we offer a parallel approach to improve the efficiency of the TFPP query.
- We perform an extensive experimental study with both synthetic and real datasets to verify the efficiency and effectiveness of our proposed algorithms.

The rest of the paper is organized as follows. In Section 2, we propose the UDS query and TFPP query respectively, and review the related work. In Section 3, we design effective pruning strategies and present two algorithms for the UDS query. In Section 4, we present a TFPP algorithm and its efficient parallelization for the TFPP query. In Section 5, we evaluate the performance of the proposed algorithms by extensive experiments. In Section 6, we conclude the paper and also expatiate the directions for future work.

2 Problem Statement

In this paper, we research our problems on the basis of the existentially uncertain data model where each uncertain object exists to be a certain object with a probability [8]. Based on this model, we formulate the UDS and the TFPP queries in this section. Then, we review the related research about our problems.

2.1 UDS Query

First, we introduce the notion of a dynamic dominance operator and dynamic skyline query over precise datasets.

Consider a product set $P = \{p_1, p_2, \ldots, p_n\}$ and a customer set $C = \{c_1, c_2, \ldots, c_m\}$, which store the information of different products and customers' preferences, respectively. Each product $p \in P$ is represented by a multi-dimensional point $< p[1], p[2], \ldots, p[d] >$, where $p[i]$ denotes its ith attribute value. Without loss of generality, we assume that a smaller value of $p[i]$ for $1 \leq i \leq d$ is preferable. In addition, for a customer $c \in C$, the preference on the product is represented by a multi-dimensional point $< c[1], c[2], \ldots, c[d] >$. Here, $c[j]$ denotes the special requirement on the jth attribute value.

Definition 2.1 (Dynamic Dominance [3]). *In view of a customer $c \in C$, a product $p \in P$ dynamically dominates another $p' \in P$ is expressed as follows:*

$$p \prec_c p' = (\forall i, |p[i] - c[i]| \leq |p'[i] - c[i]|) \wedge (\exists i, |p[i] - c[i]| < |p'[i] - c[i]|)$$

where $1 \leq i \leq d$.

In this paper, we utilize the dynamic skyline query to return "attractive" products because this query is a powerful tool that is used to return products based on a customer's preferences and can also capture subjective attributes [3].

Definition 2.2 (Dynamic Skyline Query, DSQ [3]). *Let C be a set of customers and P be a set of products. For a customer $c \in C$, the DSQ returns all the products $p \in P$ such that there is not any other $p' \in P - \{p\}$ satisfying $p' \prec_c p$.*

Based on Definitions 2.1 and 2.2, we propose an uncertain dynamic dominance operator and an uncertain dynamic skyline query in the following.

Definition 2.3 (Uncertain Dynamic Dominance, UD-Dominance). *Let C be a set of customers and $UP = \{(p_1, Pr(p_1)), (p_2, Pr(p_2)), \ldots, (p_n, Pr(p_n))\}$ be a set of probabilistic products, where $Pr(p)$ represents the existential probability of a product $p \in UP$. A product $p \in UP$ UD-dominates another $p' \in UP$ with respect to c is expressed as follows:*

$$p \prec_c^u p' = \big(\forall i\ |p[i] - c[i]| \leq |p'[i] - c[i]| \wedge Pr^c_{DSky}(p) \geq Pr^c_{DSky}(p')\big) \wedge \big(\exists i\ |p[i] - c[i]| < |p'[i] - c[i]| \vee Pr^c_{DSky}(p) > Pr^c_{DSky}(p')\big).$$

Here, $1 \leq i \leq d$ and the dynamic skyline probability of a product $p \in UP$ is computed by

$$Pr^c_{DSky}(p) = Pr(p) \times \prod_{p' \in UP, p' \prec_c p} (1 - Pr(p')). \tag{1}$$

Lemma 2.1. *Given two products p and p' within UP, if $p \prec_c p'$ and $Pr^c_{DSky}(p) \geq Pr^c_{DSky}(p')$, then we have $p \prec_c^u p'$.*

Due to space limitation, the proofs of some lemmas and theorems are given in the supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2016.2584606, of this paper.

As illustrated in Fig. 2a, $w_3 \prec_{c_1} w_6$. The dynamic skyline probability of w_6 is $Pr(w_6) \times (1 - Pr(w_3)) = 0.60 \times (1 - 0.40) = 0.36$. Since $w_3 \prec_{c_1} w_6$ and the dynamic skyline probability of w_3 is larger than that of w_6, we gain $w_3 \prec_{c_1}^u w_6$ according to Lemma 2.1.

Lemma 2.2 (Transitive Property of Uncertain Dynamic Dominance). *Given three products p, p', and p'' within UP, if $p \prec_c^u p'$ and $p' \prec_c^u p''$, then we gain $p \prec_c^u p''$.*

Based on Definition 2.2, we formulate the uncertain dynamic skyline query as follows.

Definition 2.4 (Uncertain Dynamic Skyline Query, UDS Query). *Given a probabilistic product set UP and a customer c, the uncertain dynamic skyline query retrieves all the products $p \in UP$ which are not UD-dominated by any other product $p' \in UP$ with respect to c. That is, the UDS query returns a set $UDS(UP, c)$ such that:*

$$UDS(UP, c) = \{p \in UP | \nexists p' \in UP - \{p\}, p' \prec_c^u p\}.$$

Continuing the example in Fig. 1, Fig. 2 shows the UDS query results with respect to customer c_i $(1 \leq i \leq 3)$, respectively. In view of customer c_1, the UDS query retrieves a set $\{w_2, w_3\}$. Similarly, in view of customers c_2 and c_3, the UDS queries report $\{w_1, w_2\}$ and $\{w_3, w_5, w_6\}$, respectively.

Lemma 2.3. *The $UDS(UP, c)$ is complete and correct.*

Consider a set of candidate products UP with $|UP| = N$ and dimensionality d. According to the expected number of skyline query result in [9], the size of the UDS query results is computed by

$$S(d, N) = H(d+1, N) \approx \frac{1}{d!}((\ln N) + r)^d, \quad (2)$$

where r is the Euler-Mascheroni constant approximately equal to 0.577.

Our UDS query differs from the PDS query in [6], [7] mainly in the following aspects:

- Friendliness. In the PDS query, it is necessary to specify a probabilistic threshold, which is difficult for customers without guidelines. Moreover, the PDS query may return undesirable results; and overlook some important objects. To address this, we formulate a UDS query that dispenses with the probabilistic threshold, is able to obtain significantly better results, and can report all the important results.
- Fairness. The PDS query does not satisfy the fairness requirement because it only focuses on the objects that have large dynamic skyline probabilities. However, in most cases, the data objects with small dynamic skyline probabilities are also of great significance for the customers [10]. Accordingly, in our UDS query, we take both the dynamic skyline probabilities and the attributes of the objects into account.
- Stability. The PDS query results always change with different probabilistic thresholds, whereas our UDS query results are stable over a given dataset.

2.2 TFPP Query

The UDS query only takes a customer's preferences into account. Moreover, in this section, we propose an interesting query that retrieves the top k favorite probabilistic products from the perspective of various customers.

To choose the k products that can meet various customers' maximum demands, the typical approach is to specify a function to rank the query results. However, it is inconvenient and difficult for users to provide an appropriate ranking function without guidelines [3]. Motivated by these aspects, we introduce a new selection strategy due to the favorite probabilities.

TABLE 3
The UDS Query Results in View of c_i $(1 \leq i \leq 3)$

Wines \ Customers	c_1	c_2	c_3
$(w_1, 0.9)$	0.00	0.90	0.00
$(w_2, 0.8)$	0.48	0.80	0.00
$(w_3, 0.4)$	0.40	0.00	0.40
$(w_4, 0.5)$	0.00	0.00	0.00
$(w_5, 0.7)$	0.00	0.00	0.42
$(w_6, 0.6)$	0.00	0.00	0.60

Definition 2.5 (Favorite Probability). *Given a probabilistic product set UP, a customer c, and a product set $UDS(UP, c)$ which contains the UDS query result set in view of c, the favorite probability of each product $p \in UDS(UP, c)$, denoted as $Pr_{Fav}(p, c)$, is computed by*

$$Pr_{Fav}(p, c) = \frac{Pr_{DSky}^c(p)}{\sum_{p' \in UDS(UP,c)} Pr_{DSky}^c(p')}.$$

Furthermore, we measure the favorite probability of the product p for a customer set C by

$$\begin{aligned} Pr_{Fav}(p, C) &= \sum_{c \in C} Pr_{Fav}(p, c) \\ &= \sum_{c \in C} \frac{Pr_{DSky}^c(p)}{\sum_{p' \in UDS(UP,c)} Pr_{DSky}^c(p')}. \end{aligned} \quad (3)$$

Based on the favorite probability, we propose the formulation of the TFPP query.

Definition 2.6 (Top k Favorite Probabilistic Products Query, TFPP). *Given a customer set C and a probabilistic product set UP, the TFPP query retrieves k products $p \in \cup_{c \in C} UDS(UP, c)$ with the highest favorite probabilities in view of C.*

Consider the set of grape wines in Fig. 1. Table 3 shows the UDS query results and their dynamic skyline probabilities in view of customer c_i $(1 \leq i \leq 3)$, respectively. Here, if wine w_i is not contained in the UDS query results in view of customer c_j for $1 \leq i \leq 6$ and $1 \leq j \leq 3$, then its dynamic skyline probabilities is set to 0.00.

According to Definition 2.5 and Equation (3), with respect to the customer set $C = \{c_1, c_2, c_3\}$, the favorite probabilities of w_1 and w_2, respectively, are computed by

$$\left(\frac{0.00}{0.48+0.40} + \frac{0.90}{0.90 + 0.80} + \frac{0.00}{0.40 + 0.42 + 0.60}\right) = 0.53,$$

and

$$\left(\frac{0.48}{0.48 + 0.40} + \frac{0.80}{0.90 + 0.80} + \frac{0.00}{0.40 + 0.42 + 0.60}\right) = 1.02.$$

Similarly, we gain the favorite probabilities of w_3, w_4, w_5, and w_6, which are 0.74, 0.00, 0.30, and 0.42, respectively. Let

TABLE 4
Symbols and Descriptions

Symbol	Description
p, p'	A product
c	A customer (query point)
C	A set of customers (query points)
m	The size of C
UP	The set of probabilistic products
N	The size of UP
$Pr(p)$	The existential probability of p
$p \prec_c p' (p \prec_c^u p')$	The product p dynamically dominates (UD-dominates) product p' in view of c
$Pr_{DSky}^c(p)$	The dynamic skyline probability of the product p in view of c

$k = 2$. The TFPP query retrieves w_2 and w_3, which have the top two favorite probabilities, as the query results.

2.3 Related Work

In this section, we summarize the most relevant problems to our work, which are the dynamic skyline query and query processing for product positing problem.

2.3.1 Dynamic Skyline Queries

Papadias et al. [11] first formulated a dynamic skyline query, which is to find points whose dynamic attributes are not dominated by those of other points [11]. There have been abundant works that focus on this query. Sharifzadeh and Shahabi [12] introduced the spatial skyline query where the dynamic attributes are computed with respect to euclidean distances to query points. Deng et al. [13] proposed a usual definition of dynamic skyline queries where the dynamic attributes of each point are computed as its absolute coordinates with respect to a query point.

In addition, there are some variants of the dynamic skyline query which were studied in the literatures [14], [15], [16], [17]. Recently, one of the variants, the reverse skyline query, is paid much attention. Gao et al. [14] proposed two new reverse skyline queries, which are reverse k-skyband and ranked reverse skyline queries. In addition, reverse skyline queries over wireless sensor networks [15], data streams [16], and arbitrary non-metric similarity measures [14] were researched successively [14]. Moreover, Islam et al. [17] studied why-not questions in reverse skyline queries.

Skyline queries over uncertain data have recently garnered considerable attention. Pei et al. [18] first proposed the probabilistic skyline query over uncertain database and it has some follow-up work [2], [6], [7]. In [6], it formulated a probabilistic reverse skyline query which is based on a dynamic skyline query over an uncertain database. Lian et al. [7] researched the probabilistic group subspace skyline queries which are also based on the PDS queries. However, as analyzed in Section 1, the PDS query results mainly depend on the specified probabilistic threshold. As a result, it returns undesirable results and overlooks important results.

2.3.2 Query Processing for Product Positing

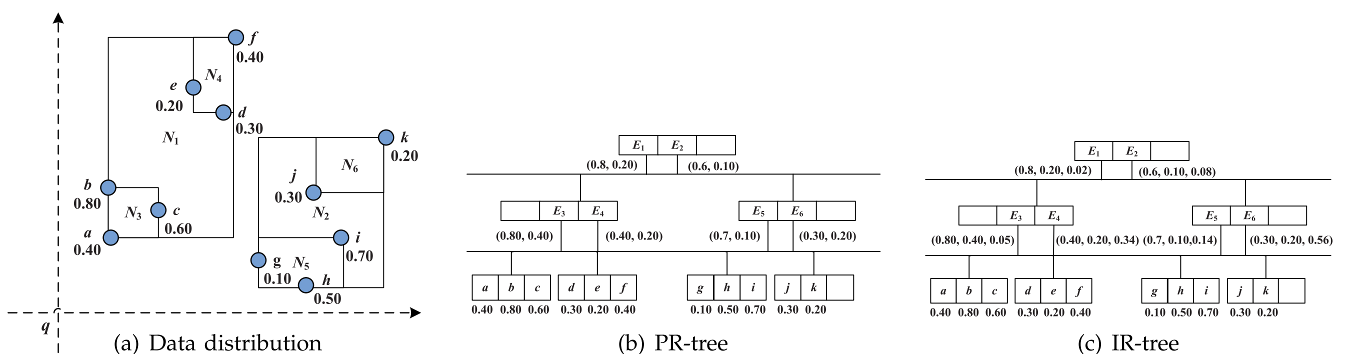
The purpose of product positing problem investigated in [4], [19], [20] is to help companies develop new products that can satisfy the customers' demands. This is also our goal in this paper.

The most closely related work about our problems is [4]. It formulated a k-most demanding products (k-MDP) discovering problem, which is to help the company to select k products with the maximum expected number of customers. The k-MDP discovering problem is an efficient tool for companies to find competitive products, but it also has some limitations. First, the k-MDP discovering problem is NP-hard when the number of attributes for a product is no less than 3, such that its computation is expensive and only approximate algorithms can be used to handle it effectively [4]. Besides, this problem is inapplicable to our scenario where only candidate products are taken into account. Second, [4] takes the customer preferences into account. It allows customers to specify their worst acceptable values for each attribute and products having better values than the special ones are considered to be satisfiable. Such a formulation cannot be used for handling subjective types of attributes [3]. In addition, it cannot measure how relevant each product with respect to the customer preferences [3]. Finally, this approach is only applied to handle the traditional products with certainty and inappropriate to process probabilistic products, which have been paid a growing concern.

Table 4 summarizes the frequently used symbols.

3 UDS Query Processing

To improve the query efficiency, an index is built on databases as usual [18], [6], [7], [21]. In this paper, we employ the PR-tree proposed in [21] to organize the datasets. As an example shown in Fig. 3, a PR-tree PR is constructed over the dataset in Fig. 3a, which contains eleven data objects. In the PR-tree, each leaf entry contains the existential probabilities



(a) Data distribution (b) PR-tree (c) IR-tree

Fig. 3. Examples of PR-tree and IR-tree.

of the objects apart from the attribute values. In particular, each intermediate entry e includes the minimum and maximum existential probabilities of its child entries.

3.1 The UDSQ Algorithm

In this section, we design pruning strategies to improve the UDS query performance, and integrate them into an effective algorithm.

3.1.1 Pruning Strategies

Based on the characteristics of the UDS query and the PR-tree, some effective lemmas are proposed for reducing the search space.

Lemma 3.1. *Given two products $p, p' \in UP$, if $(p \prec_c p') \wedge \Big((Pr(p) \geq Pr(p')) \vee (Pr(p) \geq Pr(p') \times (1 - Pr(p)))\Big)$, then p' can be pruned safely.*

Proof. Since $p \prec_c p'$, the dynamic skyline probability of p' is computed by

$$\begin{aligned} & Pr_{DSky}(p') \\ & = Pr(p') \times (1 - Pr(p)) \times \prod(1 - Pr(p_i)) \times \prod(1 - Pr(p_j)) \\ & = Pr_{DSky}(p) \times \frac{Pr(p')}{Pr(p)} \times (1 - Pr(p)) \times \prod(1 - Pr(p_j)), \end{aligned} \tag{4}$$

where $p_i, p_j \in UP$, $p_i \prec_c p$, $p_j \not\prec_c p$ and $p_j \prec_c p'$. Since the assumption in this lemma holds, by rewriting Equation (4), we have $Pr_{DSky}(p) \geq Pr_{DSky}(p')$.

As $p \prec_c p'$ and $Pr_{DSky}(p) \geq Pr_{DSky}(p')$, thus $p \prec_c^u p'$ due to Lemma 2.1. Therefore, this lemma holds due to Definition 2.4.

Let E_{nea} be the nearest point of the bound rectangle of entry E to a customer c and $Pr(E_{nea}) = \max_{p \in E} Pr(p)$. Considering the relationship between a product and an intermediate entry, we have the following lemmas.

Lemma 3.2. *Given an entry E and a product p, if $p \prec_c^u E_{nea}$, then $p \prec_c^u E$, i.e., $p \prec_c^u E_{nea} \Rightarrow p \prec_c^u E$.*

Proof. For any product $p_i \in E$, it holds that $E_{nea} \prec_c p_i$. The dynamic skyline probability of p_i is computed by

$$\begin{aligned} & Pr_{DSky}(p_i) \\ & = Pr(p_i) \times (1 - Pr(E_{nea})) \times \prod(1 - Pr(p')) \times \prod(1 - Pr(p'')) \\ & \leq Pr_{DSky}(E_{nea}) \times \frac{Pr(p_i)}{Pr(E_{nea})}, \end{aligned}$$

where $p', p'' \in UP$, $p' \prec_c E_{nea}$, $p'' \prec_c p_i$ and $p'' \not\prec_c E_{nea}$.

Since $Pr(p_i) \leq Pr(E_{nea})$, it holds that $Pr_{DSky}(p_i) \leq Pr_{DSky}(E_{nea})$. For $E_{nea} \prec_c p_i$ and $Pr_{DSky}(E_{nea}) \geq Pr_{DSky}(p_i)$, it holds that $E_{nea} \prec_c^u p_i$ based on Lemma 2.1. Since $p \prec_c^u E_{nea}$ and $E_{nea} \prec_c^u p_i$, we get $p \prec_c^u p_i$ due to Lemma 2.2. Because p_i denotes any product within E, it holds that $p \prec_c^u E$.

Lemma 3.3. *Given an entry E and a product p, if $(p \prec_c E_{nea}) \wedge ((Pr(p) \geq Pr(E_{nea})) \vee (Pr(p) \geq Pr(E_{nea}) \times (1 - Pr(p)))$, then the products within E can be pruned safely.*

Proof. Based on the proof of Lemma 3.1, it is easy to get $p \prec_c^u E_{nea}$. Moreover, due to Lemma 3.2, we have $p \prec_c^u E$. Therefore, each product within E is not a UDS query result and this lemma holds.

3.1.2 The UDSQ Algorithm

By integrating the pruning strategies in Section 3.1.1, we present a UDSQ algorithm for the UDS query. The UDSQ algorithm consists of two phases, pruning and refining phases. The pruning phase is applied to obtain a candidate product set by pruning products that are UD-dominated. The refining phase is utilized to refine the candidate products and obtain the final accurate UDS query results.

Algorithm 1. UDSQ_Algorithm

Input: A PR-tree PR over a probabilistic product set UP and a customer (query point) c.
Output: A set UDS containing all the final UDS query results.

1: Initialize a min-heap $PH = \emptyset$ and two sets $UDS_{can} = UDS = \emptyset$
2: Insert all entries in the root of PR into PH
3: **while** PH is not empty **do**
4: Remove the top entry E from PH
5: **if** E is a leaf node **then**
6: **if** $p \not\prec_c^u E$ for any $p \in UDS_{can}$ due to Lemma 3.1 **then**
7: Insert E into UDS_{can}
8: **else**
9: **if** E is an intermediate node **then**
10: **if** $p \not\prec_c^u E$ for any $p \in UDS_{can}$ based on lemma 3.3 **then**
11: **for** each entry E' in E **do**
12: **if** $p \not\prec_c^u E'$ for any candidate $p \in UDS_{can}$ based on lemma 3.3 **then**
13: Insert E' into PH
14: Refine UDS_{can} and add the final query answers to UDS
15: Return UDS

The UDSQ algorithm is presented in Algorithm 1. Initially, UDSQ keeps the candidate set UDS_{can}, which contains the candidate results of the UDS query. Furthermore, in the UDS query procedure, it visits the leaf and intermediate entries of the PR-tree in non-decreasing order of their L_1−norm distances with respect to the query point c and maintains a minimum heap PH. Then lemmas introduced in Section 3.1.1 are utilized to reduce the search space. After executing Lines 3-13 of the UDSQ algorithm, the remaining objects failed to be pruned are stored in UDS_{can}. Because our pruning approaches can ensure that all the pruned objects are the UDS query results, the set UDS_{can} following the pruning phase contains all of the query answers. Moreover, it is necessary to refine the candidates in UDS_{can} to obtain the actual query results. This is because some objects that are not part of the UDS query results still exist in UDS_{can}. Line 14 refines UDS_{can} by checking the UD-dominance relationship between the different candidate products after obtaining their accurate dynamic skyline probabilities. After this refining phase, the UDSQ algorithm terminates and the set UDS containing the final query answers is returned.

Example. As the example in Fig. 3, Table 5 depicts the heap contents for processing the UDS query in each

TABLE 5
Heap Contents of UDSQ

Action	Heap contents	UDS_{can}
Access root	$(E_1, 10), (E_2, 18)$	ϕ
Expand E_1	$(E_3, 10), (E_2, 18), (E_4, 27)$	ϕ
Expand E_3	$(a, 10), (b, 14), (c, 16), (E_2, 18), (E_4, 27)$	ϕ
Access a, b, c	$(E_2, 18), (E_4, 27)$	$\{a, b\}$
Expand E_2	$(E_5, 18), (E_4, 27), (E_6, 31)$	$\{a, b\}$
Expand E_5	$(g, 20), (h, 22), (E_4, 27), (i, 29), (E_6, 31)$	$\{a, b\}$
Access g, h	$(E_4, 27), (i, 29), (E_6, 31)$	$\{a, b, g, h\}$
Access E_4	$(i, 29), (E_6, 31)$	$\{a, b, g, h\}$
Access i	$(E_6, 31)$	$\{a, b, g, h\}$
Access E_6	ϕ	$\{a, b, g, h\}$

step of the UDSQ algorithm. It starts by inserting all the entries (i.e., E_1 and E_2) in root of the PR into a heap PH, where the entries are sorted in non-decreasing order of their minimum L_1-norm distance. The entry E_1 with the minimum L_1-norm distance is then "expanded". This expansion pops out E_1 from PH and adds its children E_3 and E_4 back. Next, by expanding E_3, its children a, b, and c are inserted. Since a and b cannot be pruned by any object in the UDS_{can} (empty), they are inserted into UDS_{can} as candidate results. However, c is pruned due to Lemma 3.1 for $a \prec_q c$ and $Pr(a) > Pr(c) \times (1 - Pr(a))$. The next entry to be expanded is E_2. After expanding E_2, we gain $PH = \{E_5, E_4, E_6\}$. Furthermore, after expanding E_5, we get another two candidate results, g and h. After that, since $a \prec_c^u E_4$, E_4 is pruned due to Lemma 3.3. Similarly, i and E_6 are pruned due to $h \prec_c^u i$ and $a \prec_c^u E_6$. Finally, we obtain $UDS_{can} = \{a, b, g, h\}$. In the refining phase, we compute the dynamic skyline probabilities of the candidates within UDS_{can}. The objects a, b, g, and h which are not UD-dominated are added to the set UDS and reported as the final UDS query results.

Theorem 3.4. *The UDSQ can return the final UDS query results correctly.*

Theorem 3.5. *The UDSQ visits the PR-tree PR $S(d, N) + 1$ times at least.*

3.2 The Enhanced UDSQ (EUDSQ) Algorithm

To accelerate the UDS query, the window query operator and reuse technique [14], [22] are employed in the EUDSQ algorithm. Moreover, we adjust the PR-tree and introduce an IR-tree, where each intermediate entry also stores its non-existential probability. As an example, the IR-tree over the dataset in Fig. 3a is shown in Fig. 3c.

3.2.1 Pruning Strategies

In this section, we first present a lemma to identify the final UDS query results as soon as possible. Based on this lemma, we can report the final results progressively.

Lemma 3.6. *Given a product p, if there is not any product $p' \in UP - \{p\}$ with $p' \prec_c p$, then p is a final UDS query result and its dynamic skyline probability is $Pr^c_{DSky}(p) = Pr(p)$.*

Proof. Since there is not any product $p' \in UP - \{p\}$ with $p' \prec_c p$, we gain $p' \not\prec_c p$. Therefore, it holds that

$$\neg(\forall i\ |p[i] - c[i]| \le |p'[i] - c[i]| \wedge \exists i\ |p[i] - c[i]| < |p'[i] - c[i]|)$$
$$= \neg(\forall i\ |p[i] - c[i]| \le |p'[i] - c[i]|) \vee \neg(\exists i\ |p[i] - c[i]| < |p'[i] - c[i]|).$$

Based on Definition 2.3, we have $p' \not\prec_c^u p$. Therefore, p is a final UDS query result due to Definition 2.4.

Due to Equation (1), we get

$$Pr^c_{DSky}(p) = Pr(p) \times \prod_{p' \in UP, p' \prec_c p} (1 - Pr(p')) = Pr(p).$$

Hence, this lemma holds.

To further improve the pruning capacity, we propose a novel pruning strategy with considering the UD-dominant relationship between different entries.

Given an entry E, E_{far} is the farthest point of the bound rectangle of the entry E to a customer c and $Pr(E_{far}) = \min_{p \in E} Pr(p)$.

Lemma 3.7. *Given two uncertain entries E and E', if $E_{far} \prec_c^u E'_{nea}$, then $E \prec_c^u E'$, i.e., $E_{far} \prec_c^u E'_{nea} \Rightarrow E \prec_c^u E'$. Accordingly, the products within E' can be pruned safely.*

Proof. For any product $p_i \in E$, it is easy to obtain that $p_i \prec_c E_{far}$ and $Pr(p_i) \ge Pr(E_{far})$. Similarly to Lemma 3.2, we have $p_i \prec_c^u E_{far}$. Since $E_{far} \prec_c^u E'_{nea}$, it holds that $p_i \prec_c^u E'_{nea}$ based on Lemma 2.2. As p_i represents any product within E', this lemma holds.

3.2.2 The EUDSQ Algorithm

Similarly to the UDSQ algorithm, the EUDSQ depicted in Algorithm 2 contains a pruning phase and a refining phase. In the pruning phase, apart from the lemmas in Section 3.1.1, we use Lemmas 3.6 and 3.7 in Section 3.2.1, to further reduce the search space. Moreover, Lemma 3.7 is applied in Line 17 of EUDSQ to prune all the unqualified entries in PH.

Algorithm 2. EUDSQ_Algorithm

Input: An IR-tree IR over a probabilistic product set UP and a customer c.
Output: A set UDS containing all the final UDS query results.

1: Initialize a min-heap $PH = \emptyset$ and two sets $UDS_{can} = UDS = \emptyset$
2: Insert all entries in the root of PR into PH
3: **while** PH is not empty **do**
4: Remove the top entry E from PH
5: **if** E is a leaf node **then**
6: **if** $p \not\prec_c E$ for any product $p \in UDS$ **then**
7: Insert E into UDS and $Pr^c_{DSky}(E) = Pr(E)$ according to Lemma 3.6
8: **else**
9: **if** $p \not\prec_c^u E$ for any candidate $p \in UDS_{can}$ due to Lemma 3.1 **then**
10: Insert E into UDS_{can}
11: **else**
12: **if** E is an intermediate node **then**
13: **if** $p \not\prec_c^u E$ for any $p \in UDS_{can}$ due to lemma 3.3 **then**
14: **for** each entry E' in E **do**
15: **if** $p \not\prec_c^u E'$ for any $p \in UDS_{can}$ due to lemma 3.3 **then**
16: Insert E' into PH
17: Prune unqualified entries in PH by Lemma 3.7
18: UDS = $UDS \cup$RefineReuse_Algorithm(UDS_{can}, IR)
19: Return UDS

TABLE 6
Heap Contents of EUDSQ

Action	Heap contents	UDS_{can}	UDS
Access root	$(E_1, 10)$, $(E_2, 18)$	ϕ	ϕ
Expand E_1	$(E_3, 10)$, $(E_2, 18)$, $(E_4, 27)$	ϕ	ϕ
Expand E_3	$(a, 10)$, $(b, 14)$, $(c, 16)$, $(E_2, 18)$	ϕ	ϕ
Access a, b	$(E_2, 18)$	$\{b\}$	$\{a\}$
Expand E_2	$(E_5, 18)$, $(E_6, 31)$	$\{b\}$	$\{a\}$
Expand E_5	$(g, 20)$, $(h, 22)$, $(i, 29)$, $(E_6, 31)$	$\{b\}$	$\{a\}$
Access g, h	$(E_6, 31)$	$\{b\}$	$\{a, g, h\}$
Access E_6	ϕ	$\{b\}$	$\{a, g, h\}$

In the refining phase of the UDSQ algorithm, it is necessary to visit the PR-tree $S(d, N) + 1$ times (see Theorem 3.5). There are an abundance of redundant I/O operators because each node of the PR-tree is accessed multiple times. In [14], [22], [23], the reusing technique is proven to be an effective way of reducing the redundant I/O operators. Accordingly, we utilize this technique to boost the UDS query performance.

The RefineReuse algorithm (Line 26) in EUDSQ is similar to the WindowQuery algorithm proposed in [23]. The accessed nodes are inserted into the heaps PH_w and PH_r. After getting the dynamic skyline probability of the first candidate product within UDS_{cand}, we can visit the PH_r instead of IR for computing the accurate dynamic skyline probabilities of the remaining candidate products. Moreover, for each product $p \in UDS_{cand}$, we employ a window query operator to compute its dynamic skyline probability.

Example. Continuing the example in Fig. 3a, the EUDSQ algorithm organizes the datasets with the IR-tree IR illustrated in Fig. 3c. As depicted in Table 6, the EUDSQ starts by inserting all the entries E_1 and E_2 in the root of IR into PH. Then, after expanding E_1 with the minimum L_1-norm distance, $PH = \{E_3, E_2, E_4\}$. Because $E_3 \prec_q^u E_4$ due to Lemma 3.7, E_4 is pruned and $PH = \{E_3, E_2\}$. After expanding E_3, $PH = \{a, b, E_2\}$ for $a \prec_q^u c$ due to Lemma 3.1. Next, we further pop out a and b from PH in sequence. Since a is not pruned by any object within UDS, it is inserted to UDS due to Lemma 3.6 as a final UDS query result. Considering $a \prec_q b$ but b cannot be pruned by any object within UDS_{can}, it is added to UDS_{can} for further verification. Repeating this iteration procedure, we obtain the candidate set $UDS_{can} = \{b\}$ and the final result set $UDS = \{a, g, h\}$ at last. After that, it needs to refine the set UDS_{can}. Since $a \prec_q b$ but $a \not\prec_q^u b$ for $Pr_{DSky}(b) = 0.48 > Pr_{DSky}(a) = 0.40$, b is added to UDS as a final UDS query result. Finally, we get the UDS query result set $UDS = \{a, b, g, h\}$.

Refer to Theorem 3.4, we can analogously prove that EUDSQ is correct. Moreover, we analyze the I/O cost and the progressiveness of the EUDSQ algorithm.

Theorem 3.8. *The EUDSQ needs to access IR two times.*

Theorem 3.9. *The EUDSQ can return the UDS query results progressively.*

Theorem 3.10. *The EUDSQ is more efficient than UDSQ.*

Proof. The UDSQ needs to visit the PR-tree PR for $S(d, N) + 1$ times based on Theorem 3.5, and it exists many redundant I/O cost. However, the EUDSQ visits the IR-tree IR for twice due to Theorem 3.8. It is easy to draw a conclusion that EUDSQ needs less I/O cost compared to UDSQ.

Furthermore, in the pruning phase, EUDSQ can gain better pruning capacity than UDSQ by employing more pruning strategies. In addition, in the refining phase of UDSQ, it needs refine $S(d, N)$ candidate products at least, while EUDSQ only needs to refine

$$S(d, N) - H(d, N) = H(d + 1, N) - H(d, N) \approx \frac{1}{d!}((\ln N) + r - d)((\ln N) + r)^{d-1}$$

candidate products at least, where r is the Euler-Mascheroni constant approximately equal to 0.577 [9]. Accordingly, EUDSQ needs less CPU cost than UDSQ.

From the analysis above, EUDSQ needs less I/O and CPU costs than UDSQ. Hence, this lemma holds.

4 TFPP Query Processing

In this section, we present two algorithms, TFPP and ParTFPP, for processing the TFPP query. In TFPP and ParTFPP, EUDSQ is invoked to compute the UDS query results due to Theorem 3.10.

4.1 The TFPP Algorithm

By employing EUDSQ in Section 3, we gain the candidate products for each customer in C. We then compute their favorite probabilities due to Equation (3), and sort them in non-increasing order. At last, the k products which have the highest favorite probabilities are returned as the TFPP query results.

Algorithm 3. TFPP_Algorithm

Input: An IR-tree IR over a probabilistic product set UP and a customer set C.
Output: A set $FavProd$ containing k products with the highest favorite probabilities.

1: **for** each customer $c_i \in C$ **do**
2: UDS_i=EUDSQ(IR, c_i, UP)
3: $UDS = \cup_{i=1}^{m} UDS_i$
4: **for** each product $p \in UDS$ **do**
5: Compute $Pr_{Fav}(p, C)$ due to Equation (3)
6: $\mathrm{FavPr}_{\min} = 0$
7: Initialize $FavProd$ with products p_i for $1 \leq i \leq k$
8: $\mathrm{FavPr}_{\min} = \min_{i=1}^{k} Pr_{Fav}(p_i, C)$
9: **for** each product $p' \in UDS - \{\cup_{i=1}^{k} p_i\}$ **do**
10: **if** $Pr_{Fav}(p', C) > \mathrm{FavPr}_{\min}$ **then**
11: Remove product p with $Pr_{Fav}(p, C) = \mathrm{FavPr}_{\min}$ from $FavProd$ and $FavProd = FavProd \cup \{p'\}$
12: Return $FavProd$

The TFPP algorithm is proposed in Algorithm 3. It consists of three phases:

(1) Executing the UDS query with respect to each customer within C (Lines 1-2).
(2) Computing the favorite probability of each candidate product within UDS (Lines 4-5).

(3) Choosing the k products that have the highest favorite probabilities as the final TFPP query results (Lines 6-11).

In the third phase of Algorithm 3, we first initialize a set $FavProd$ that contains the final TFPP query results, with the first k accessed products. We also maintain the minimum favorite probability with $FavPr_{min}$ and initialize it in Line 8. In Lines 9-11, we visit the remaining candidate products. Assume that a product $p' \in UDS - \{\cup_{i=1}^{k} p_i\}$ is being visited. If its favorite probability is larger than the present minimum favorite probability maintained in $FavPr_{min}$, we then refresh the set $FavPr$ in Line 11. After scanning the set UDS, we can obtain the final TFPP query results.

Theorem 4.1. *The TFPP algorithm needs to access the IR-tree IR $2|C|$ times.*

Theorem 4.2. *The computational complexity of the TFPP algorithm is $O((\hbar\partial + \delta + m + k)|UDS| - k^2 + k)$, where $UDS = \cup_{c_i \in C} UDS(UP, c_i)$, $m = |C|$, δ denotes the average cost of a window query operator, $\hbar$ and ∂ represent the height of the IR-tree and the average access cost of visiting a node, respectively.*

Proof. In the implementation of the TFPP algorithm, we use an IR-tree to index the probabilistic product set. The TFPP algorithm includes three phases. In the first phase, it executes UDS queries in view of customers within C by invoking the EUDSQ algorithm. Suppose that the height of the IR-tree is $\hbar$ and average access cost of visiting a node is ∂, the node accesses cost is $O(\hbar\partial|UDS|)$. In the refining phase of each UDSQ query, window query operators over each reuse heap are used. Suppose the average cost of a window query operator is δ, the total cost of refining all the candidates is $O(\delta|UDS|)$. Therefore, the cost of UDS queries is $O((\hbar\partial + \delta)|UDS|)$. In the second phase, the cost of computing favorite probabilities of all the probabilistic products is $O(|C||UDS|) = O(m|UDS|)$. In the final phase, it takes $O(k + k(|UDS| - k))$ cost to identify the k products with the highest favorite probabilities.

From the analysis above, the total computational complexity of the TFPP algorithm is

$$
\begin{aligned}
&O((\hbar\partial + \delta)|UDS| + m|UDS| + k + k(|UDS| - k)) \\
&= O((\hbar\partial + \delta + m + k)|UDS| - k^2 + k).
\end{aligned}
$$

4.2 The Parallel TFPP (ParTFPP) Algorithm

In the TFPP algorithm, it is necessary to execute UDS queries with respect to all of the customers within C in serial. This is the main issue affecting the TFPP query efficiency. To boost the TFPP algorithm, we adopt the parallel mechanism into the TFPP query and introduce a parallel TFPP algorithm, namely ParTFPP. In the ParTFPP algorithm, we divide the customer set C equally into r parts, $C_1, C_2, \ldots, C_r$. Thereafter, the favorite probability of product p with respect to C can be computed by

$$
\begin{aligned}
Pr_{Fav}(p, C) &= \sum_{i=1}^{r} Pr_{Fav}(p, C_i) \\
&= \sum_{i=1}^{r} \sum_{c \in C_i} \frac{Pr^{c}_{DSky}(p)}{\sum_{p' \in UDS(UP,c)} Pr^{c}_{DSky}(p')}.
\end{aligned}
$$

The ParTFPP algorithm is described in Algorithm 4. Each computing node N_i processes the UDS queries with respect to a customer set C_i for $1 \le i \le r$ in parallel. We can gain the favorite probability of the product p by summing its local favorite probabilities. As depicted in Algorithm 4, the ParTFPP algorithm includes the following two phases.

(1) Local-Computation phase. Each computing node N_i executes UDS queries with respect to the customers within C_i to obtain the local UDS query results (Line 3). The local favorite probability of each candidate product in UDS_i (Line 5) is computed; and sorted in non-increasing order of their local favorite probabilities in parallel (Line 6).

(2) Global-Merge phase. This involves merging the local query results and obtaining our TFPP query answers (Lines 7-56). This merging phase plays an important role in developing the efficiency of this TFPP algorithm. Therefore, we analyze this phase in detail.

Algorithm 4. ParTFPP_Algorithm

Input: An IR-tree IR over a probabilistic product set UP, a customer set C with size m, and computing nodes N_i for $1 \le i \le r$.
Output: A set $FavProd$ containing k favorite products.
1: Divide C into r parts $C_1, C_2, \ldots, C_r$ where $|C_i| = \lfloor \frac{m}{r} \rfloor$ for $1 \le i < r$ and $|C_r| = m\%r$
2: **for** each computing node N_i for $1 \le i \le r$ **do**
3: UDS_j = EUDSQ(IR, c_j, UP) where $c_j \in C_i$, and $UDS_i = \bigcup_{j=1}^{|C_i|} UDS_j$
4: **for** each product $p' \in UDS_i$ **do**
5: Compute $Pr_{Fav}(p', C_i)$ due to Equation (3)
6: Sort products within UDS_i in non-increasing order of their local favorite probabilities
7: Initialize $FavProd$ with the first visited k products from the sets UDS_i for $1 \le i \le r$ and compute their favorite probabilities by random access
8: Initialize $FavPr_{min} = \min_{p' \in FavProd} Pr_{Fav}(p', C)$
9: Define a threshold of the favorite probability $\alpha = \sum_{i=1}^{r} Pr_{Fav}(\hat{p}_i, C_i)$, where $\hat{p}_i$ is the latest visited product at N_i
10: **for** each product $p' \in FavProd$ **do**
11: **if** $Pr_{Fav}(p', C) > \alpha$ **then**
12: Return p' as a query result and $k = k - 1$
13: **while** UDS_i is not empty and $k \ne 0$ **do**
14: Do sequential access in parallel to each of the sets UDS_i for $1 \le i \le r$
15: Real-time update for α with the latest visited product $\hat{p}_i$
16: **if** $Pr_{Fav}(\hat{p}_i, C) > FavPr_{min}$ **then**
17: Refresh $FavProd$ and $FavPr_{min}$
18: **if** $FavPr_{min} \ge \alpha$ **then**
19: Return $FavProd$ and break
20: **else**
21: **for** each product $p' \in FavProd$ **do**
22: **if** $Pr_{Fav}(p', C) > \alpha$ **then**
23: Return p' as a query result and $k = k - 1$

In Line 7, we initialize a set $FavProd$ with the first visited products and obtain their favorite probabilities. We then define a variable $FavPr_{min}$ to store the minimum favorite probability of the products in $FavProd$. Line 9 computes a favorite probability threshold α by summing up the favorite

probabilities of products that are accessed currently. If $FavPr_{min} \geq \alpha$, then all of the products in $FavProd$ are our query results. Otherwise, to develop the progressiveness of the ParTFPP algorithm, it is crucial to return the query results as soon as possible. Therefore, Lines 10-12 are applied to find the qualified query results and retrieve them in a timely manner for customers. Lines 13-23 are executed to identify the remaining query results through an iteration procedure. Each iteration first accesses the results of local UDS queries at different computing nodes in parallel. Line 15 updates the value of α with the latest visited product in real time. Lines 16-17 are used to update the set $FavProd$ and $FavPr_{min}$. If $FavPr_{min} \geq \alpha$, then we return $FavProd$ as the final query result set. Otherwise, Lines 21-23 are run to produce new query results progressively.

This merging approach is similar to the famous TA algorithm for top k queries [24]. Moreover, to further improve the effectiveness and progressiveness of the TA algorithm, we update the threshold (Line 15) with the latest visited product in real time. For developing the ParTFPP algorithm's progressiveness, we return partial TFPP query results as soon as possible (Lines 21-23).

Theorem 4.3. *The ParTFPP algorithm needs to access the IR-tree IR $2|C_i|$ times at each computing node N_i.*

Theorem 4.4. *The total computational complexity of the TFPP algorithm is*

$$O\Bigg(\Big(\hbar\partial + \delta + \Big\lfloor \frac{m}{r} \Big\rfloor\Big)|UDS^*| + |UDS^*|\log|UDS^*| + (d^* r C_S + a^*(r-1)C_R)\Bigg),$$

where $|UDS^*| = \max_{i=1}^{r}|UDS_i|$, C_R and C_S represent the average cost of sequential and random access, respectively. Similarly to [24], we also assume the merging phase halts at depth d^* and visits a^* distinct products. In addition, the definitions of $\hbar$, ∂ and δ can refer to Theorem 4.2.

Discussion. It is clear that we can obtain the best query performance when the computational complexity in Theorem 4.4 is minimized. Assume that m is simply an integral multiple of r, since for given product and customer sets, $|UDS^*|$, d^* and a^* are constant. The ParTFPP algorithm can offer the best performance when

$$\Big\lfloor \frac{m}{r} \Big\rfloor \times |UDS^*| = (d^* C_S + a^* C_R) r.$$

Therefore, if

$$r = \sqrt{\frac{m \times |UDS^*|}{d^* C_S + a^* C_R}},$$

then the ParTFPP algorithm can offer the best performance.

5 Experimental Evaluation

To evaluate the performance of the proposed algorithms, we implemented all of them by C++. The experiments were performed on a cluster composed of eight nodes. Each node is with Intel Xeon E5-2667 3.3 GHz CPU (contains eight cores) and 64 GB main memory, and runs the 64 bit Microsoft Windows 7 operation system.

TABLE 7
Parameter Settings on Synthetic Datasets

Parameter	Range
Size of UP ($\lvert UP\rvert$)	64K, **128K**, 256K, 512K, 1024K
Size of C ($\lvert C\rvert$)	2K, **4K**, 6K, 8K, 10K
Dimensionality (d)	2, **3**, 4, 5
The probabilistic threshold (α)	**0.2**, 0.4, 0.5, 0.6, 0.8

5.1 Experimental Setup

The publicly data generator provided by [25] was used to generate synthetic datasets which follow independent (Ind) and anti-correlated (Ant) distributions. We investigate the performance of the proposed algorithms under various parameters, as summarized in Table 7. In each experiment, we only vary one parameter and fix the others to their defaults. The probabilistic threshold α is specified for the PDS query. In particular, the values in bold font denote the defaults.

We also evaluate the algorithms for two real-world datasets, i.e., CarDB and ColDB. Specifically, CarDB includes 45,311 6-dimensional points [14]. We consider two numerical attributes, price and mileage, of each car. ColDB contains 68,040 9-dimensional points, and each point captures several properties of an image [25]. We consider three dimensions that record the mean, standard deviation, and skewness of the pixels in the H channels into account. Due to the lack of a real data set with uncertainty, similar to [2], [21], we use uniform distribution to randomly generate the existential probability of each point. To generate the customer set C, we utilize the approach introduced in [3], where Gaussian noise is added to the actual points.

For both synthetic and real datasets, we normalize all attribute values to [0, 10,000] and index each data set with a PR-tree/an IR-tree with a 4 KB page size. The reported results are the average of 100 queries.

In particular, we evaluate the proposed algorithms from the following aspects:

- I/O cost (Average I/O cost): the number (average number) of nodes/page accesses occurrences charges 10 ms for each page, similar to [7].
- CPU cost: the time spent on CPU.
- Memory requirement (MH): the maximum number of entries in the heap.

5.2 Experimental Results for the UDS Query

In this section, we first compare the PDS query to our UDS query. We then analyze the performance of the UDSQ and EUDSQ algorithms.

5.2.1 Comparison of the PDS Query and Our UDS Query

As analyzed in 2.1, the PDS query in [6], [7] suffers limitations in terms of friendliness, fairness, and stability. To overcome these limitations, we formulate the UDS query.

Note that, the PDS and our UDS queries are different formulations of skyline query over uncertain data. The

TABLE 8
Analysis of the PDS Query Results versus d

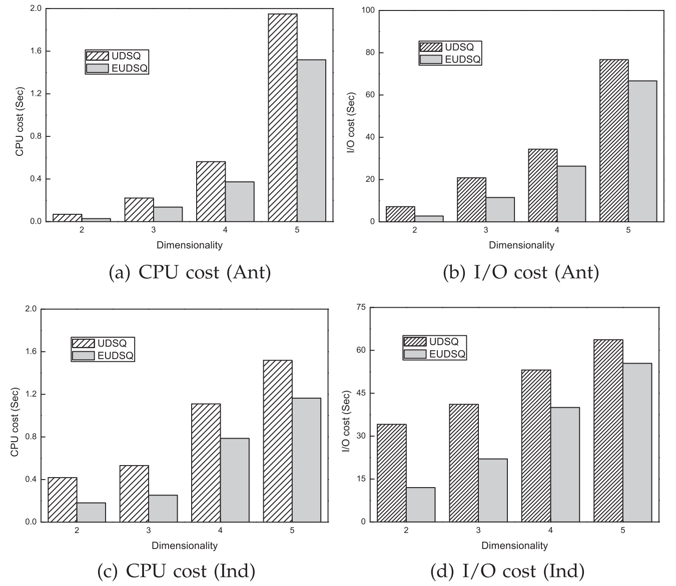
d	**Ind**		**Ant**	
	RedRat	MisRat	RedRat	MisRat
2	35.71%	20.00%	23.53%	16.07%
3	15.69%	24.32%	22.92%	18.80%
4	20.14%	16.17%	18.53%	20.75%
5	22.60%	22.24%	15.20%	20.03%

TABLE 9
Analysis of the PDS Query Results versus $|\mathrm{UP}|$

$\lvert\mathrm{UP}\rvert$	Ind		Ant	
	RedRat	MisRat	RedRat	MisRat
64K	23.91%	22.64%	19.47%	17.43%
128K	15.69%	24.32%	22.92%	18.80%
256K	31.19%	23.44%	20.32%	21.07%
512K	16.28%	17.91%	23.75%	19.46%
1024K	26.23%	20.73%	20.46%	22.99%

TABLE 10
Analysis of the PDS Query Results versus α

α	Ind		Ant	
	RedRat	MisRat	RedRat	MisRat
0.2	15.69%	24.32%	22.92%	18.80%
0.4	0.00%	39.19%	1.51%	37.47%
0.6	0.00%	56.76%	0.00%	58.45%
0.8	0.00%	81.08%	0.00%	79.02%



(a) CPU cost (Ant) (b) I/O cost (Ant)
(c) CPU cost (Ind) (d) I/O cost (Ind)

Fig. 4. Performance versus d.

TABLE 11
The MH of UDSQ and EUDSQ versus d

d	**UDSQ**		**EUDSQ**	
	Ind	Ant	Ind	Ant
2	3,086	646	876	202
3	3,138	1,602	1,258	675
4	3,639	2,191	2,337	1,411
5	4,533	6,153	3,718	5,154

approaches for the PDS query proposed in [6], [7] cannot be applied to process our UDS query. Thus, we do not compare with them in our experiments. Instead, we compare the two queries based on query results in the aspects of redundant rate (RedRat) and missing rate (MisRat). Here the RedRat and MisRat are the ratios of the undesirable results and the important results overlooked in the PDS query. The important results overlooked are those objects which are not dynamically dominated by other objects due to Definition 2.1, but are not returned as the PDS query results.

As shown in Tables 8 and 9, by varying d or $|UP|$, the PDS queries over the Ind and Ant datasets always report undesirable results and overlook important results. With the growth of d, RedRat and MisRat of the PDS query are up to 35.71 and 24.32 percent, respectively. As $|UP|$ increases, the highest values of RedRat and MisRat are 31.19 and 24.32 percent, respectively.

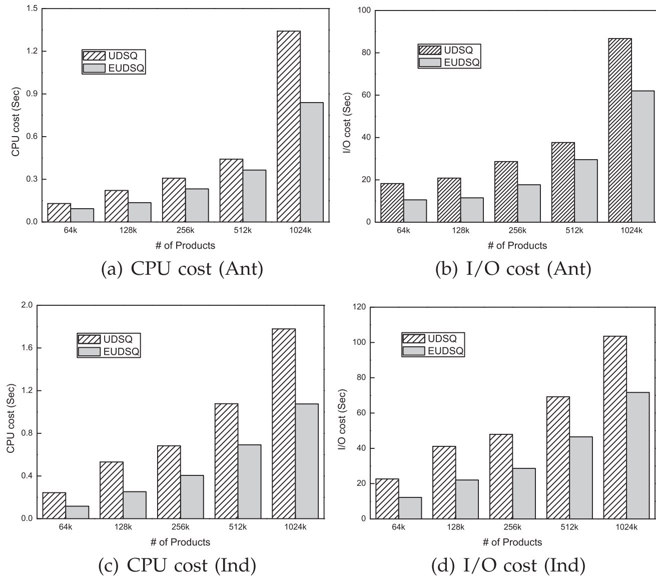
With growing the probabilistic threshold α, MisRat of the PDS query increases sharply as illustrated in Table 10. This is since with the growth of α, much more important results, of which the dynamic skyline probabilities are less than α, are pruned. When $\alpha = 0.8$, the PDS query overlooks 81.08 and 79.02 percent of the important results over the Ind and Ant datasets, respectively. As α increases, less undesirable results are reported by the PDS query. When $\alpha > 0.5$, the PDS query results do not contain undesirable results. This is reasonable. If $\alpha > 0.5$ and there are two different PDS query results t and t' with $t \prec_c^u t'$, we then gain $Pr_{DSky}(t) \geq \alpha > 0.5$ and $Pr_{DSky}(t') \geq \alpha > 0.5$. Therefore, it holds that $Pr(t) > 0.5$ and $Pr(t') > 0.5$. For $t \prec_c^u t'$, it holds that $t \prec_c t'$. Since $Pr(t) > 0.5$, we obtain $1 - Pr(t) < 0.5$. For $0 \leq Pr(t') \leq 1$, we have $Pr_{DSky}(t') = Pr(t') \times (1 - P(t)) \times \prod_{t'' \in \mathrm{UD}-\{t\}, t'' \prec_c t'} (1 - Pr(t'')) < 0.5$. This contradicts the assumption.

As analyzed above, the PDS query always overlooks some important results and contains undesirable results in most cases. However, our UDS query could overcome these limitations due to Definition 2.4. This is reasonable for two reasons. First, the UDS query does not return any undesirable results since it prunes the objects which are UD-dominated due to Definition 2.4. Second, it accounts for both the attributes and dynamic skyline probabilities of objects. Accordingly, important objects with small dynamic skyline probabilities, which have advantages in terms of their attributes, are also reported as part of our UDS query results.

5.2.2 Experimental Results for the UDS Query

We could also process the UDS query by another algorithm which computes the dynamic skyline probabilities of objects first and then adjusts the BBS algorithm to compute the UDS query results. However, this algorithm is inefficient to process the UDS query. Therefore, in this section, we mainly compare the performance of the proposed algorithms, UDSQ and EUDSQ, by varying two parameters: d and $|UP|$, individually.

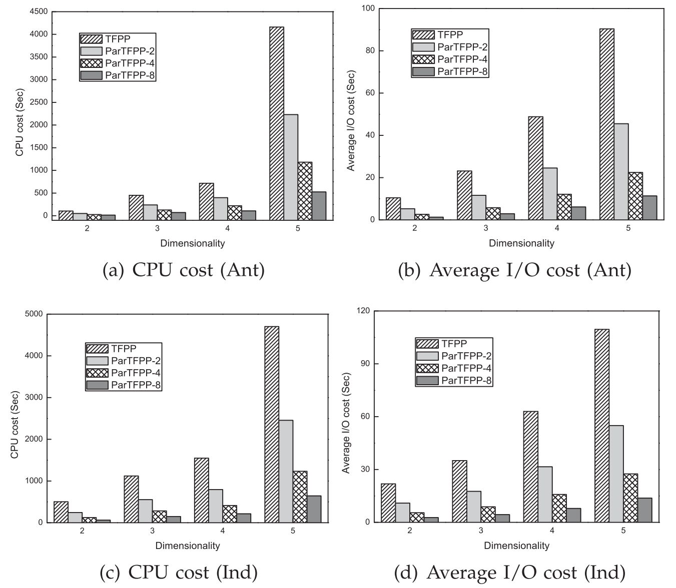
Experimental results versus d. As shown in Fig. 4 and Table 11, d has a significant impact on the performance of the UDSQ and EUDSQ algorithms. The I/O and CPU costs of the two algorithms increase quickly with the growth of d. This is because, as d increases, the datasets become sparse, which makes the size of the UDS query results grows. The



(a) CPU cost (Ant) (b) I/O cost (Ant)

(c) CPU cost (Ind) (d) I/O cost (Ind)

Fig. 5. Performance versus $|\mathrm{UP}|$.



(a) CPU cost (Ant) (b) Average I/O cost (Ant)

(c) CPU cost (Ind) (d) Average I/O cost (Ind)

Fig. 6. Performance versus d.

poor performance of the PR-tree or IR-tree in high dimensionality also leads to the degradation of the performance of the two algorithms.

Moreover, EUDSQ requires less I/O and CPU costs than UDSQ with the growth of d. In particular, compared to UDSQ, for the Ind and Ant datasets, EUDSQ reduces 59.10 CPU cost, 64.77 I/O cost, and 71.61 percent MH at most.

Experimental results versus $|UP|$. Fig. 5 shows the experimental results of UDSQ and EUDSQ as varying $|UP|$. Both UDSQ and EUDSQ require more CPU and I/O costs as $|UP|$ grows. This is because a larger data set has a larger number of UDS query results and results in higher CPU and I/O costs. In addition, a small data set sometimes may be more expensive than a large one. This is due to the positions of the skyline points and the order in which they are discovered. If the first query result is very close to the query point, both UDSQ and EUDSQ could prune a large part of the search space.

Moreover, in all cases, EUDSQ is much more efficient and scales better than UDSQ as $|UP|$ grows (as depicted in Table 12 and Fig. 5). With comparing to UDSQ, EUDSQ cuts down 52.50 CPU cost and 46.45 percent I/O cost at most. In terms of MH, as shown in Table 12, EUDSQ degrades the MH by up to 59.91 percent compared to UDSQ.

5.2.3 Performance on Real Datasets

In this section, we also research the performance of UDSQ and EUDSQ on two real-world datasets.

The EUDSQ clearly outperforms UDSQ in terms of MH, the CPU and I/O costs, on two real-world datasets. This is similar to the synthetic datasets. For clarity and readability, the detailed analysis of performance on real datasets are moved to the supplement.

TABLE 12
The MH of UDSQ and EUDSQ versus $|\mathrm{UP}|$

$\|\mathrm{UP}\|$	UDSQ		EUDSQ	
	Ind	Ant	Ind	Ant
64K	1,851	1,538	806	775
128K	3,138	1,602	1,258	675
256K	3,202	1,932	1,415	838
512K	4,021	1,656	1,795	847
1,024K	5,445	4,268	2,266	1,800

5.3 Experimental Results for the TFPP Query

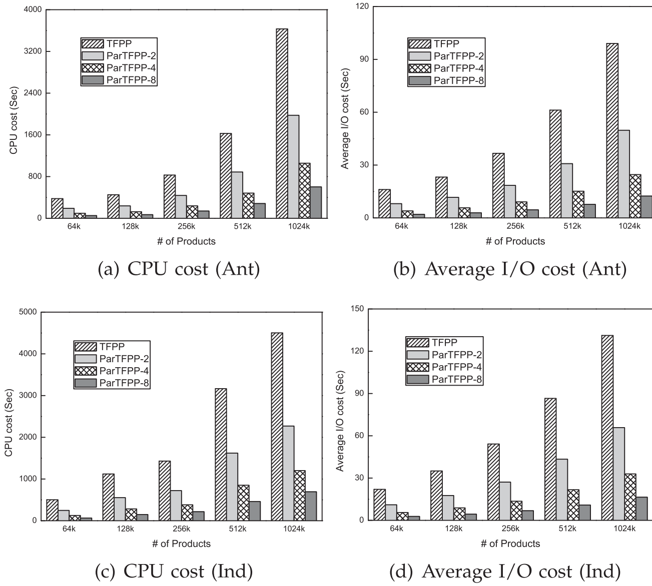
In this section, we study the performance of our proposed algorithms, TFPP and ParTFPP, for the TFPP query. Since the two algorithms have similar memory requirement (MH) under the same parameters, we only analyze the experimental results of CPU and the average I/O costs in the following experiments. Furthermore, the value of k does not have a significant impact on the performance of our proposed algorithms for the TFPP query. Thus, the experimental results under different k are skipped, and k is set to 40.

In the following experiments, we compare the performance of TFPP and ParTFPP by varying three parameters: d, $|UP|$, and $|C|$, individually. ParTFPP-i ($i = 2, 4, 8$) denotes the experimental results of i computing nodes, respectively.

Experimental results on d. We first examine the two algorithms, TFPP and ParTFPP, by varying d. Fig. 6 shows the CPU and average I/O costs of the above algorithms. Here the average I/O cost is denoted as the average I/O cost for a customer. As depicted in Fig. 6, the CPU and average I/O costs increase sharply as the growth of d. This trend is similar to that of the UDS query. Moreover, ParTFPP exhibits much better performance than TFPP. In terms of CPU and average I/O costs, the approximate linear speedups are obtained by ParTFPP without changing MH.

Experimental results on $|UP|$. We then conduct a performance analysis with the growth of $|UP|$. The experimental results are shown in Fig. 7. As $|UP|$ grows, both TFPP and ParTFPP require higher CPU and I/O costs. Moreover, ParTFPP has much better performance than TFPP. The speedup is approximate to the linearity by applying ParTFPP with varying $|UP|$.

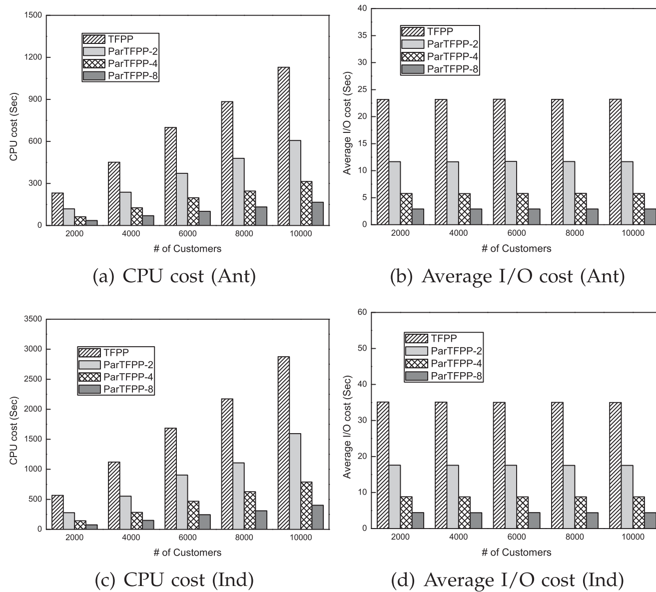
Experimental results on $|C|$. Fig. 8 illustrates that the CPU cost for the TFPP query becomes much larger as $|C|$ increases. With respect to the average I/O cost, the customer size $|C|$ has little impact as depicted in Fig. 8.



(a) CPU cost (Ant) (b) Average I/O cost (Ant)

(c) CPU cost (Ind) (d) Average I/O cost (Ind)

Fig. 7. Performance versus $|\mathrm{UP}|$.



(a) CPU cost (Ant) (b) Average I/O cost (Ant)

(c) CPU cost (Ind) (d) Average I/O cost (Ind)

Fig. 8. Performance versus $|C|$.

Furthermore, in term of CPU cost, ParTFPP can also offer an approximate linear speedup with changing $|C|$.

Experimental results on real datasets. Similar to the synthetic datasets, it can obtain an approximate linear speedup by adopting ParTFPP on real datasets. Due to space limitation, the detailed analyses of TFPP and ParTFPP on real datasets are moved to the supplement.

6 Conclusions

Customer preferences information is a growing concern in market analysis. In this paper, we first propose the UDS query to select products that can meet a customer's demands to the greatest extent. Compared to the PDS query, our UDS query does not need to specify a threshold and can return much better results. In addition, with respect to the preferences of different customers, we formulate the TFPP query, which retrieves the k products with the highest favorite probabilities. Moreover, to process the UDS and TFPP query effectively, some pruning strategies are proposed and integrated into several effective algorithms. Finally, the efficiency and effectiveness of the proposed algorithms have been verified with extensive experiments. As part of our future research, we will investigate the UDS and TFPP queries on big data.

Acknowledgments

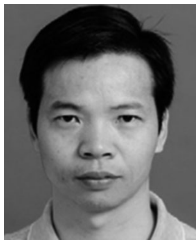
The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124, 61202109, and 61472126), the International Science & Technology Cooperation Program of China (Grant No. 2015DFA11240), the National High-tech R&D Program of China (Grant No. 2015AA015303), and the Outstanding Graduate Student Innovation Fund Program of Collaborative Innovation Center of High Performance Computing. Kenli Li is the corresponding author.

References

[1] S. Fay and J. Xie, "Probabilistic goods: A creative way of selling products and services," *Mark. Sci.*, vol. 27, no. 4, pp. 674–690, 2008.
[2] 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.
[3] A. Arvanitis, A. Deligiannakis, and Y. Vassiliou, "Efficient influence-based processing of market research queries," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 1193–1202.
[4] C.-Y. Lin, J.-L. Koh, and A. L. Chen, "Determining k-most demanding products with maximum expected number of total customers," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 8, pp. 1732–1747, Aug. 2013.
[5] Y. Peng, R.-W. Wong, and Q. Wan, "Finding top-k preferable products," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 10, pp. 1774–1788, Oct. 2012.
[6] X. Lian and L. Chen, "Reverse skyline search in uncertain databases," *ACM Trans. Database Syst.*, vol. 35, no. 1, 2010, Art. no. 3.
[7] 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.
[8] 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.
[9] L. Zhu, Y. Tao, and S. Zhou, "Distributed skyline retrieval with low bandwidth consumption," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 3, pp. 384–400, Mar. 2009.
[10] M. J. Atallah and Y. Qi, "Computing all skyline probabilities for uncertain data," in *Proc. 28th ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems*, 2009, pp. 279–287.
[11] 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, Mar. 2005.
[12] M. Sharifzadeh and C. Shahabi, "The spatial skyline queries," in *Proc. 32nd Int Conf. Very Large Data Bases*, 2006, pp. 751–762.
[13] E. Dellis and B. Seeger, "Efficient computation of reverse skyline queries," in *Proc. 33rd Int Conf. Very Large Data Bases*, 2007, pp. 291–302.
[14] Y. Gao, Q. Liu, B. Zheng, L. Mou, G. Chen, and Q. Li, "On processing reverse k-skyband and ranked reverse skyline queries," *Inf. Sci.*, vol. 293, pp. 11–34, Feb. 2015.
[15] G. Wang, J. Xin, L. Chen, and Y. Liu, "Energy-efficient reverse skyline query processing over wireless sensor networks," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 7, pp. 1259–1275, Jul.2012.
[16] M. Bai, J. Xin, and G. Wang, "Probabilistic reverse skyline query processing over uncertain data stream," in *Proc. 17th Int. Conf. Database Syst. Adv. Appl.*, 2012, pp. 17–32.

[17] M. S. Islam, R. Zhou, and C. Liu, "On answering why-not questions in reverse skyline queries," in *Proc. 29th Int. Conf. IEEE Int. Conf. Data Eng.*, 2013, pp. 973–984.
[18] 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.
[19] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng, "Creating competitive products," *Proc. Int. Conf. Very Large Data Bases*, vol. 2, no. 1, pp. 898–909, Aug. 2009.
[20] Q. Wan, R.-W. Wong, and Y. Peng, "Finding top-k profitable products," in *Proc. IEEE 27th Int. Conf. Data Eng.*, 2011, pp. 1055–1066.
[21] 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.
[22] 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.
[23] Y. Gao, Q. Liu, B. Zheng, and G. Chen, "On efficient reverse skyline query processing," *Expert Syst. Appl.*, vol. 41, no. 7, pp. 3237–3249, Jun. 2014.
[24] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *J. Comput. Syst. Sci.*, vol. 66, no. 4, pp. 614–656, Jun. 2003.
[25] 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.

Xu Zhou received the PhD degree from the Department of Information Science and Engineering, Hunan University, China, in 2016. She is currently a postdoctoral researcher of electrical engineering at Hunan University. Her research interests include parallel computing and data management.



Kenli Li received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2003. He is currently a full professor of computer science and technology at Hunan University. His major research includes parallel computing and cloud computing. He has published more than 150 papers in international conferences and journals, such as the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, *and the Journal of Parallel and Distributed Computing*. He is currently on the editorial board of the *IEEE Transactions on Computers*. He is a senior member of the IEEE and an outstanding member of CCF.

Guoqing Xiao is currently working towards the PhD degree in the Department of Information Science and Engineering, Hunan University, Changsha, China. His research interests include parallel computing and data management.

Yantao Zhou received the PhD 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 at Hunan University, Changsha. His major research interests include parallel computing and distributed data management.

Keqin Li is a SUNY distinguished professor of computer science. His current research interests include parallel computing and distributed computing. He has published more than 410 journal articles, book chapters, and refereed conference papers, and 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*, and the *IEEE Transactions on Cloud Computing*. He is a fellow member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.