Computing Influence of a Product through Uncertain Reverse Skyline

Md. Saiful Islam[‡], Wenny Rahayu[#], Chengfei Liu[†], Tarique Anwar[†] and Bela Stantic[‡]

[‡]Griffith University, Gold Coast, Australia
[#]La Trobe University, Melbourne, Australia
[†]Swinburne University of Technology, Melbourne, Australia
saiful.islam@griffith.edu.au,w.rahayu@latrobe.edu.au,{cliu,tanwar}@swin.edu.au,b.stantic@griffith.edu.au

ABSTRACT

Understanding the influence of a product is crucially important for making informed business decisions. This paper introduces a new type of skyline queries, called *uncertain reverse skyline*, for measuring the influence of a probabilistic product in uncertain data settings. More specifically, given a dataset of probabilistic products $\mathcal{P}$ and a set of customers C, an *uncertain reverse skyline* of a probabilistic product q retrieves all customers $c \in C$ which include q as one of their preferred products. We present efficient pruning ideas and techniques for processing the *uncertain reverse skyline query* of a probabilistic product using R-Tree data index. We also present an efficient parallel approach to compute the *uncertain reverse skyline* and *influence score* of a probabilistic product. Our approach significantly outperforms the baseline approach derived from the existing literature. The efficiency of our approach is demonstrated by conducting experiments with both real and synthetic datasets.

CCS CONCEPTS

• **Information systems** → **Data scans**; • **Theory of computation** → *Shared memory algorithms*;

KEYWORDS

UD-Dominance, Uncertain Reverse Skyline, Query Processing Algorithms, Parallel Computing.

ACM Reference format:
Md. Saiful Islam[‡], Wenny Rahayu[#], Chengfei Liu[†], Tarique Anwar[†] and Bela Stantic[‡]. 2017. Computing Influence of a Product through Uncertain Reverse Skyline. In *Proceedings of SSDBM '17, Chicago, IL, USA, June 27-29, 2017,* 12 pages.
DOI: http://dx.doi.org/10.1145/3085504.3085508

1 INTRODUCTION

These days we are experiencing voluminous customer preference and the product popularity rating data available from the product related websites, e.g., search queries in CarSales[1], YahooAutos[2] etc and the product ratings in Amazon[3], eBay[4] etc. The popularity ratings of the products in these sites can be treated as the probabilities by which the products match the customer preferences. Making intelligent use of these customer preference and popularity rating data might help production companies to optimize their (probabilistic) selling strategy or promotion plans and thereafter, increase their revenues [7]. To illustrate the problem settings studied in this paper, consider the datasets of wine products and the customer preferences as given in Fig. 1. In general, a product is assumed to be liked by a customer if it closely matches her stated preference. However, the popularity rating of a product may also play an important role in her buying decision in reality. For example, though w_3 matches the preference of the customer c_3 better than w_5, w_5 still has the chance to attract c_3 as its popularity rating is higher than w_3. We argue that both of the above factors need to be modeled in determining the influence of a product for the manufacturers to sustain in the global market.

Wines	1-GraCon(%)	Price($)	Rating
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

(a) Grape wines, $\mathcal{W}$

Preferences	1-GraCon(%)	Price($)
c_1	55	130
c_2	35	110
c_3	70	140
c_4	55	130
c_5	35	110
c_6	70	140

(b) Customer Preferences, C

Figure 1: Example datasets of (a) wines and (b) customer preferences. The "Rating" column denotes the popularity of a wine in the market.

The first operator for preference-based data retrieval over certain data is the *skyline operator* introduced by Börzsönyi et al. [4] to the database research community. Since then, this operator has received lots of attention and is studied extensively in multi-criteria decision making applications ([17], [13], [21], [5], [24], [23] for survey). Given a dataset of products $\mathcal{P}$, the standard skyline query returns all products $p \in \mathcal{P}$ that are not dominated by any other products $p' \in \mathcal{P}$. A product p is considered to dominate another product p' *iff* it is as good as p' in every aspects of p', but better than p' in at least one aspect of p'. Mathematically, p dominates p', denoted by $p \prec p'$, iff: (i) $\forall i \in \{1, 2, ..., d\}$, $p^i \leq p'^i$ and (ii) $\exists j \in \{1, 2, ..., d\}$, $p^j < p'^j$, assuming that smaller values are preferred in all dimensions, p^i

SSDBM '17, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/3085504.3085508

[1]http://www.carsales.com.au/
[2]http://autos.yahoo.com/
[3]https://www.amazon.com/
[4]http://www.ebay.com/

and p'^i denote the ith dimensional values of p and p', respectively and $\mathcal{P}$ is a set of d-dimensional data objects. For example, consider the dataset of wine products given in Fig. 1(a), the standard skyline operator [4] on this wine dataset returns $\{w_1, w_2\}$ as no other wine can dominate these wines in terms of *1- percentage of grape juice content* (1-GraCon(%)) and *price*($).

Though standard skyline queries [4] can trade-off well if there are multiple dimensions of a product and a customer is unable to weight these dimensions, not all customers may prefer to minimize/maximize every dimensional value of a product, rather s/he may like certain range for it, e.g., laptop screen size, GraCon(%) etc. To address this, Papadias et al. [17] propose *dynamic skyline query*, which retrieves data objects p that are not *dynamically dominated* by another data object p' w.r.t. a customer preference c, where c is also a d-dimensional data object. Unlike standard skyline queries [4] where the aspects of p is directly compared with the corresponding aspects of p' without considering any customer object, the dynamic skyline query compares the absolute differences of the aspects of p and the customer object c with the corresponding absolute differences of the aspects of p' and the customer object c in deciding the dominance between p and p'. Mathematically, a data object p dynamically dominates another data object p' w.r.t. a customer object c, denoted by $p \prec_c p'$, iff: (i) $\forall i \in \{1, 2, ..., d\}$, $|p^i - c^i| \leq |p'^i - c^i|$ and (ii) $\exists j \in \{1, 2, ..., d\}$, $|p^j - c^j| < |p'^j - c^j|$. For example, consider the dataset of wines given in Fig. 1(a) and the customer preferences in Fig. 1(b), the dynamic skyline query of c_1 on the wine dataset returns w_3 as no other wines can dominate w_3 in view of c_1, i.e., w_3 matches the customer preference c_1 better than any other wines given in Fig. 1(a).

Both the standard skyline [4] and dynamic skyline [17] queries retrieve data objects from P based on the customer's point of view, not the company's perspective. Dellis et al. [5] present a new type of skyline queries, called *reverse skyline*, which retrieves data objects from the company's point of view. Given a dataset of products $\mathcal{P}$, a set of customer preferences C and a product query q, the reverse skyline query retrieves all customers $c \in C$ that include q as one of their preferred products. Mathematically, given datasets $\mathcal{P}$ and C and a query q, a customer $c \in C$ is a reverse skyline of q, iff $\nexists p \in \mathcal{P}$ such that (i) $\forall i \in \{1, 2, ..., d\}$, $|p^i - c^i| \leq |q^i - c^i|$ and (ii) $\exists j \in \{1, 2, ..., d\}$, $|p^j - c^j| < |q^j - c^j|$. For example, consider the dataset of wine products given in Fig. 1(a) and the customer preferences in Fig. 1(b), the reverse skyline query of w_1 returns c_2 as no other wines in Fig. 1(a) can dominate w_1 in view of c_2, i.e., w_1 is one of the preferred products of the the customer c_2. Like the standard and dynamic skyline queries, reverse skylines are also studied with great importance in the literature, specifically for measuring the *influence* of a product and evaluating the market research queries ([24], [2], [12], [10] for survey).

Though the above skyline queries are important findings for studying the customer-product relationships over certain data, none of them is applicable over uncertain data. In works [14], [15] Lian et al. present a threshold-based approach for evaluating reverse skyline queries over uncertain data. To find the threshold-based reverse skyline of a probabilistic product $p \in \mathcal{P}$, the authors first discover the probable alternative products of a customer $c \in C$, called *probabilistic dynamic skyline*. The probabilistic dynamic skyline of a customer c, denoted by $PDS(c)$, is computed as follows: $\{\forall p \in \mathcal{P} | Pr^c_{DSky}(p) \geq \delta\}$, where $Pr^c_{DSky}(p)$ denotes the *dynamic skyline probability* of a product p w.r.t. c and is computed as follows: $Pr^c_{DSky}(p) = Pr(p) \times \prod_{\forall p' \in \mathcal{P} \setminus \{p\}, p' \prec_c p} (1 - Pr(p'))$, $Pr(p)$ denotes the probability of p and δ is a given threshold. Then, the *probabilistic reverse skyline* of a product $p \in \mathcal{P}$, denoted by $PRS(p)$, consists of all customers $c \in C$ that include p in its probabilistic dynamic skyline, i.e, $\{\forall c \in C | p \in PDS(c)\}$. For example, consider the wine products and the customers given in Fig. 1. Assume that the popularity ratings in Fig. 1(a) are the probabilities of wines. The probabilistic reverse skyline of w_2 retrieves customers c_1 and c_2 for $\delta \geq 0.48$. Certainly, the study of probabilistic reverse skylines [14], [15] is an advancement for measuring the *influence* of a product over uncertain data. However, these skylines are not that friendly from usability point of view. (**Friendliness**) One has to mention the *threshold* δ, which is certainly a burden. (**Stability**) The result set can also vary based on the settings of δ and therefore, is not stable. (**Fairness**) Furthermore, it is not favorable towards products with small dynamic skyline probabilities.

Recently, Zhou et al. [26] proposed a new skyline query called *uncertain dynamic skyline* to compute the probable alternative choices for a customer. Unlike probabilistic dynamic skyline [14], [15], the uncertain dynamic skyline [26] is stable and one does not need to provide any *threshold* value. A product $p \in \mathcal{P}$ is considered a member of the uncertain dynamic skyline of a customer c as long as $\nexists p' \in \mathcal{P}$ such that (i) $p' \prec_c p$ and (ii) $Pr^c_{DSky}(p') \geq Pr^c_{DSky}(p)$. For example, consider the dataset of probabilistic wine products and the customer preferences given in Fig. 1, the uncertain dynamic skyline of c_1, denoted by $UDS(c_1)$, retrieves w_2 and w_3, as no other wines can dynamically dominate them or their dynamic skyline probabilities are greater than these two wines in view of c_1. To compute the *influence* of a probabilistic product $p \in \mathcal{P}$ through uncertain dynamic skyline, one has to compute the uncertain dynamic skyline of each customer $c \in C$, i.e., $UDS(c)$ and then, check whether $UDS(c)$ includes p. As UDS query is computationally very expensive by itself, computing the influence of a probabilistic product via uncertain dynamic skyline [26] is not efficient.

This paper presents a new skyline query, called *uncertain reverse skyline*, for measuring the *influence* of a product in uncertain data settings. We also present efficient pruning ideas and an approach for processing the uncertain reverse skyline query of a probabilistic product. To be specific, our main contributions are as follows:

(1) we introduce a novel skyline query, called *uncertain reverse skyline*, for measuring the influence of a probabilistic product in uncertain data settings;
(2) we present several pruning ideas and R-Tree data indexing based techniques to compute the uncertain reverse skyline and the influence score of a product in probabilistic databases;
(3) we also present an efficient parallel computing approach for processing the uncertain reverse skyline query and influence score of a probabilistic product; and
(4) finally, we demonstrate the efficiency of our approach by conducting extensive experiments.

The rest of the paper is organized as follows: Section 2 provides the preliminaries, Section 3 presents the uncertain reverse skyline query and analyses the complexity of computing the influence score of probabilistic product through uncertain reverse skyline, Section 4 describes our approach in detail, Section 5 presents our parallel approach, Section 6 presents the experimental results, Section 7 discusses the related work and finally, Section 8 concludes the paper.

2 PRELIMINARIES

Consider a set of product objects $\mathcal{P}$ and a set of customer preferences C, where a product object $p \in \mathcal{P}$ and a customer preference $c \in C$ are d-dimensional points modeled as $< p^1, p^2, ..., p^d >$ and $< c^1, c^2, ..., c^d >$, respectively. The p^i denotes the value of the product p in the ith dimension, whereas the c^i denotes the preferred value of the customer c in the ith dimension of a product. If the product objects $p \in \mathcal{P}$ are associated with probabilities (e.g., popularity ratings), then we call it a probabilistic product set. The probability of a product $p \in \mathcal{P}$ is denoted by $Pr(p)$. We use product and product object as well as customer and customer preference interchangeably in this paper.

Definition 2.1. **Dynamic Dominance [17]** A product $p \in \mathcal{P}$ dynamically dominates another product $p' \in \mathcal{P}$ w.r.t. a customer c, denoted by $p \prec_c p'$, iff the followings hold: (i) $\forall i \in \{1, 2, ..., d\}$, $|p^i - c^i| \leq |p'^i - c^i|$ and (ii) $\exists j \in \{1, 2, ..., d\}$, $|p^j - c^j| < |p'^j - c^j|$.

Example 2.2. Consider the datasets given in Fig. 1. According to the Definition 2.1, the wine product w_3 dominates the wine product w_6 w.r.t. the customer c_1, i.e., $w_3 \prec_{c_1} w_6$.

Definition 2.3. **Dynamic Skyline Probability [15, 18].** The dynamic skyline probability of a product $p \in \mathcal{P}$ w.r.t. a customer c, denoted by $Pr^c_{DSky}(p)$, is computed as follows:

$$Pr^c_{DSky}(p) = Pr(p) \times \prod_{\forall p' \in \mathcal{P} \setminus \{p\}, p' \prec_c p} (1 - Pr(p')) \quad (1)$$

Example 2.4. Consider the datasets given in Fig. 1. As no other objects in $\mathcal{W}$ dominates w_3 w.r.t. c_1, the dynamic skyline probability of w_3 w.r.t. c_1 is $Pr^{c_1}_{DSky}(w_3) = Pr(w_3) = 0.40$. Since $w_3 \prec_{c_1} w_6$, the dynamic skyline probability of w_6 w.r.t. c_1 is $Pr^{c_1}_{DSky}(w_6) = Pr(w_6) \times (1 - Pr(w_3)) = 0.60 \times (1 - 0.40) = 0.36$.

Lemma 2.5. *$Pr^c_{DSky}(p') < Pr^c_{DSky}(p)$ iff: (i) $p \prec_c p'$ and (ii) $Pr(p') < Pr(p) \vee (Pr(p') \times (1 - Pr(p))) < Pr(p)$ [26].*

Definition 2.6. **Uncertain Dynamic Dominance (UD-Dominance) [26].** A probabilistic product $p \in \mathcal{P}$ UD-dominates another probabilistic product $p' \in \mathcal{P}$ w.r.t. a customer c, denoted by $p \prec^u_c p'$, iff the followings hold: (i) $p \prec_c p'$ and (ii) $Pr^c_{DSky}(p) \geq Pr^c_{DSky}(p')$.

Example 2.7. Consider the datasets given in Fig. 1. As $w_3 \prec_{c_1} w_6$ (see Ex. 2.2) and also, $Pr^{c_1}_{DSky}(w_3) > Pr^{c_1}_{DSky}(w_6)$ (see Ex. 2.4), w_3 UD-dominates w_6 w.r.t. c_1, i.e., $w_3 \prec^u_{c_1} w_6$.

Definition 2.8. **Uncertain Dynamic Skyline (UDS) [26].** Given a set of probabilistic products $\mathcal{P}$ and a customer c, the uncertain dynamic skyline of c, denoted by $UDS(c)$, consists of all products $p \in \mathcal{P}$ such that p is not UD-dominated by any other $p' \in \mathcal{P} \setminus p$, w.r.t. c. Mathematically, $UDS(c) = \{p \in \mathcal{P} | \nexists p' \in \mathcal{P} \setminus p : p' \prec^u_c p\}$.

Example 2.9. Consider the datasets given in Fig. 1. According to Definition 2.8, the uncertain dynamic skyline of the customer c_1, i.e., $UDS(c_1)$, consists of wines w_2 and w_3 as no other wines in $\mathcal{W}$ UD-dominates them w.r.t. c_1. Similarly, the $UDS(c_2)$ and $UDS(c_3)$ are $\{w_1, w_2\}$ and $\{w_3, w_5, w_6\}$, respectively.

Definition 2.10. **Favorite Probability [26].** Given a probabilistic product set $\mathcal{P}$, the *favorite probability* of a product p in view of a customer c, denoted by $Pr^c_{Fav}(p)$, is computed as given as follows:

$$Pr^c_{Fav}(p) = \begin{cases} \frac{Pr^c_{DSky}(p)}{\sum_{\forall p' \in UDS(c)} Pr^c_{DSky}(p')} & \text{if } p \in UDS(c) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

The **favorability rating** of a product p w.r.t. a customer set C, denoted by $Pr^C_{Fav}(p)$, is computed as follows:

$$Pr^C_{Fav}(p) = \sum_{\forall c \in C} Pr^c_{Fav}(p) = \sum_{\forall c \in C} \frac{Pr^c_{DSky}(p)}{\sum_{\forall p' \in UDS(c)} Pr^c_{DSky}(p')} \quad (3)$$

Example 2.11. Consider the datasets given in Fig. 1. According to Eq. 3, the favorability rating of w_1 w.r.t. the customer set C is $Pr^C_{Fav}(w_1) = \frac{0.00}{0.48+0.40} + \frac{0.90}{0.90+0.80} + \frac{0.00}{0.40+0.42+0.60} = 0.53$. Similarly, the favorability rating of w_2 w.r.t. C is $Pr^C_{Fav}(w_2) = \frac{0.48}{0.48+0.40} + \frac{0.90}{0.90+0.80} + \frac{0.00}{0.40+0.42+0.60} = 1.02$.

3 UNCERTAIN REVERSE SKYLINE

Here, we present a new skyline query, called *uncertain reverse skyline query* based on *UD-Dominance*[26].

Definition 3.1. **Uncertain Reverse Skyline (URS).** Given a set of probabilistic products $\mathcal{P}$, a set of customers C and a query product q, the uncertain reverse skyline of q, denoted by $URS(q)$, consists of all customers $c \in C$ such that q appears in $UDS(c)$, i.e., $q \in UDS(c)$. Mathematically, a customer $c \in C$ appears in $URS(q)$ iff $\nexists p \in \mathcal{P}$ such that: (a) $p \prec_c q$ and (b) $Pr^c_{DSky}(p) \geq Pr^c_{DSky}(q)$.

Example 3.2. Consider the datasets given in Fig. 1. According to Definition 3.1, the $URS(w_1)$ consists of c_2 only. The $URS(w_2)$ and $URS(w_3)$ are $\{c_1, c_2\}$ and $\{c_1, c_3\}$, respectively.

Unlike the *probabilistic reverse skyline* [14], [15], the proposed *uncertain reverse skyline* is *user friendly*, *stable* and *fair*. One does not need to provide the setting of threshold δ for computing the uncertain reverse skyline and it does not favor the query product over another one unless the query product strictly dominates the other one and the *dynamic skyline probability* of the query product is better than the other one. The *uncertain reverse skyline* always returns the same result as there is no threshold dependency.

Definition 3.3. **Influence.** The *influence set* of a probabilistic product $p \in \mathcal{P}$, denoted by $IS(p)$, consists of all customers $c \in C$ that appear in the uncertain reverse skyline of p, i.e, $IS(p) = URS(p)$. Given a set of probabilistic products $\mathcal{P}$ and the customer set C, the influence score of a probabilistic product p, denoted by $\tau(p)$, is measured by its favorability rating w.r.t. C, i.e., $\tau(p) = Pr^C_{Fav}(p)$.

Example 3.4. Consider the datasets given in Fig. 1. The influence score of wine product w_1 is $\tau(w_1) = Pr^C_{Fav}(w_1) = 0.53$ (easy to verify from Ex. 2.11). Similarly, the influence score of wine product w_2 is $\tau(w_2) = Pr^C_{Fav}(w_2) = 1.02$.

3.1 Complexity Analysis

A naive approach of computing the *influence score* of a probabilistic product $p \in \mathcal{P}$ like the one proposed by Zhou et al. [26] first computes the *uncertain dynamic skyline* of each customer $c \in C$, i.e., $UDS(c)$ and then, check whether the $UDS(c)$ includes the product p and then computes its *influence score* by following Eq. 3. However, this approach requires the computation of $|C|$ uncertain dynamic skylines, i.e., $UDS(c), \forall c \in C$. As the *uncertain dynamic skyline* query itself is computationally prohibitive, this naïve approach is not efficient enough to compute the influence score of a product p, i.e., $\tau(p)$. The following lemma guides how to efficiently compute $\tau(p)$ through the *uncertain reverse skyline* of p, i.e, $URS(p)$.

Lemma 3.5. $\tau(p) = Pr_{Fav}^{URS(p)}(p) = \sum_{\forall c \in URS(p)} Pr_{Fav}^{c}(p)$.

Proof. From Definition 3.3 and Eq. 3, we get:

$$\tau(p) = Pr_{Fav}^{C}(p) \quad = \sum_{\forall c \in C} \frac{Pr_{DSky}^{c}(p)}{\sum_{\forall p' \in UDS(c)} Pr_{DSky}^{c}(p')}$$

Now, we can divide the customers $c \in C$ in view of the product p into two groups: (a) the customers $c \in C$ that appear in the uncertain reverse skyline of p, i.e., $URS(p)$ and (b) the rest, i.e., $C \setminus URS(p)$. Therefore, we can rewrite the above as given as follows:

$$\begin{aligned}\tau(p) &= \sum_{\forall c \in URS(p)} \frac{Pr_{DSky}^{c}(p)}{\sum_{\forall p' \in UDS(c)} Pr_{DSky}^{c}(p')} \\ &+ \sum_{\forall c' \in \{C \setminus URS(p)\}} \frac{Pr_{DSky}^{c'}(p)}{\sum_{\forall p' \in UDS(c')} Pr_{DSky}^{c'}(p')} \\ &= \sum_{\forall c \in URS(p)} Pr_{Fav}^{c}(p) + \sum_{\forall c' \in \{C \setminus URS(p)\}} Pr_{Fav}^{c'}(p)\end{aligned}$$

According to Definition 3.1, a product p does not appear in the *uncertain dynamic skyline* of a customer c' if $c' \notin URS(p)$. Therefore, we get $Pr_{Fav}^{c'}(p) = 0, \forall c' \in \{C \setminus URS(p)\}$ as per Eq. 2 and the above can be rewritten as given as follows:

$$\tau(p) = \sum_{\forall c \in URS(p)} Pr_{Fav}^{c}(p) + 0 = Pr_{Fav}^{URS(p)}(p) \tag{4}$$

Hence, the lemma, i.e., $\tau(p) = Pr_{Fav}^{URS(p)}(p)$. □

From Lemma 3.5, we conclude that the efficiency of computing the *influence score* of a product depends merely on the efficiency of computing its *uncertain reverse skyline*, i.e., $URS(p)$. We present efficient pruning ideas and R-Tree data indexing based techniques for processing the *uncertain reverse skyline query* of a product in Section 4. As we experience voluminous product and customer data in most data retrieval systems these days, we also present a *parallel uncertain reverse skyline query evaluation technique* in Section 5, which outperforms its serial counterpart significantly.

4 OUR APPROACH

This section presents our pruning ideas and the detail of uncertain reverse skyline query processing technique of a product.

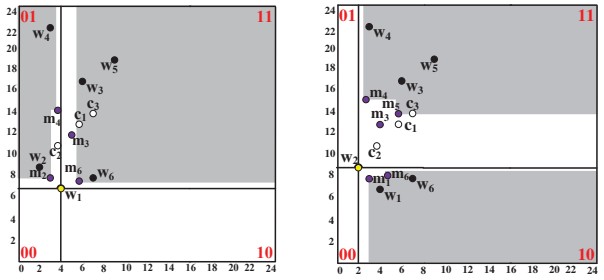


Figure 2: UD-Dominance Regions (UDRs) of (a) w_1 and (b) w_2

4.1 Pruning Ideas

Definition 4.1. **Orthant.** Given an object p and a query q, the orthant O of p w.r.t. q, denoted by $O_q(p)$, is computed as: $O_q^i(p) = 0$ *iff* $p^i \leq q^i$, otherwise $O_q^i(p) = 1$.

A d-dimensional query q has 2^d orthants in total, e.g., the orthants of w_1 and w_2 are shown as red-colored binary strings in Fig. 2(a) and Fig. 2(b), respectively.

Definition 4.2. **Midpoint.** The midpoint m of a product p w.r.t. a query q is computed as follows: $m^i = \frac{(p^i + q^i)}{2}, \forall i \in \{1, 2, ..., d\}$.

Example 4.3. Consider the datasets of wine products $\mathcal{W}$ as given in Fig. 1(a). The midpoint of w_6 w.r.t. the query product w_1 is $m_6 =< 55, 75 >$. Similarly the midpoints of w_2, w_3 and w_4 w.r.t. w_1 are $m_2 =< 30, 80 >$, $m_3 =< 50, 120 >$ and $m_4 =< 35, 145 >$, respectively. These midpoints are depicted in Fig. 2(a).

Lemma 4.4. *Assume m' is a midpoint of p' w.r.t. p and the followings hold: (i) $O_p(m') = O_p(c)$; (ii) $m' \prec_p c$; and (iii) $Pr(p) < Pr(p') \vee (Pr(p) \times (1 - Pr(p'))) < Pr(p')$. Then, we get $Pr_{DSky}^{c}(p) < Pr_{DSky}^{c}(p')$ and $c \notin URS(p)$.*

Proof. As m' is a midpoint of p' w.r.t. p, we get $p' \prec_c p \leftrightarrow m' \prec_p c$ iff $O_p(m') = O_p(c)$ [24]. This satisfies the conditions given for Lemma 2.5, i.e., $Pr_{DSky}^{c}(p) < Pr_{DSky}^{c}(p')$ if conditions (i)-(iii) hold. Now, we get $c \notin URS(p)$ according to Definition 3.1 as $p' \prec_c p$ and $Pr_{DSky}^{c}(p) < Pr_{DSky}^{c}(p')$. Hence, the lemma. □

Definition 4.5. **UD-Dominance Region (UDR).** Given a probabilistic product set $\mathcal{P}$ in a d-dimensional data space, a region is said to be a *UD-dominance region* of a product $p \in \mathcal{P}$, denoted by $UDR(p)$, for which $\forall c \in UDR(p)$, $\exists p' \in \mathcal{P}$ such that the followings hold: (i) $O_p(m') = O_p(c)$, (ii) $m' \prec_p c$ and (iii) $Pr_{DSky}^{c}(p) \leq Pr_{DSky}^{c}(p')$, where m' is the midpoint of p' w.r.t. p.

Example 4.6. Consider the datasets in Fig. 1. The UD-dominance regions of w_1 and w_2 are shown as gray patterned regions in Fig. 2(a) and Fig. 2(b), respectively. Here, the $UDR(w_1)$ is defined by the midpoints of w_2, w_4 and w_6 w.r.t. w_1. Similarly, the $UDR(w_2)$ is defined by the midpoints of w_1, w_4, w_5 and w_6 w.r.t. w_2.

Lemma 4.7. *A customer $c \in UDR(p)$ is not an uncertain reverse skyline of p, i.e., $c \notin URS(p)$ if $c \in UDR(p)$.*

Proof. Assume that $c \in UDR(p)$. According to the Definition 4.5, $\exists p' \in \mathcal{P}$ such that the midpoint of p' w.r.t. p dominates c w.r.t. p,

i.e., $p' \prec_c p$ (conditions (i)-(ii)) and also, $Pr^c_{DSky}(p) \leq Pr^c_{DSky}(p')$. Therefore, the $UDS(c)$ does not include p according to Definition 2.8, which implies $c \notin URS(p)$ according to Definition 3.1. □

Definition 4.8. **Uncertain Midpoint Skyline.** Given a set of probabilistic products $\mathcal{P}$, the uncertain midpoint skyline of a probabilistic query product q, denoted by $UMSL(q)$, consists of a minimal set of midpoints of the products $p \in \mathcal{P}$ that defines the *UD-dominance region* of q[5].

Lemma 4.9. *If there are two products $p \in \mathcal{P}$ and $p' \in \mathcal{P}$ such that the following holds: (i) $m \prec_q m'$ and (ii) $\forall c \in C$, $m' \prec_q c \rightarrow m \prec_q c$ and $Pr^c_{DSky}(p) > Pr^c_{DSky}(q)$, then $m' \notin UMSL(q)$ but $m \in UMSL(q)$, where m and m' are the midpoints of the products p and p' w.r.t. q, respectively.*

Proof. Assume that $\exists c \in C$ such that $m' \prec_q c$ and $Pr^c_{DSky}(p') > Pr^c_{DSky}(q)$, where m' is the midpoint of the product $p' \in \mathcal{P}$, but $m' \notin UMSL(q)$. This can not happen. Either $m' \in UMSL(q)$ or $\exists m \in UMSL(q)$ such that $m \prec_q m'$ and $Pr^c_{DSky}(p) > Pr^c_{DSky}(q)$, where m is the midpoint of a product $p \in \mathcal{P}$ and $p \neq p'$. For the former case, the $UMSL(q)$ is already correct as c will be pruned by m' from $URS(q)$. For the later case, we get $m \prec_q c$ as $m \prec_q m'$ and $m' \prec_q c$ (transitivity of dynamic dominance). Since $Pr^c_{DSky}(p) > Pr^c_{DSky}(q)$, the customer c can be pruned from $URS(q)$ by m even if $m' \notin UMSL(q)$. Hence, the lemma. □

Example 4.10. The uncertain midpoint skyline of w_1 consists of the midpoints of the products w_2, w_4 and w_6 w.r.t. w_1, i.e., $UMSL(w_1) = \{m_2, m_4, m_6\}$, where m_2, m_4, m_6 are the midpoints of w_2, w_4 and w_6 w.r.t. w_1. Similarly, the $UMSL(w_2)$ consists of the midpoints of the wine products w_1, w_4, w_5 and w_6 w.r.t. w_2.

4.2 Data Indexing

From Lemma 4.7 and Lemma 4.9, it is obvious that we need to compute the $UMSL(q)$ of a probabilistic product q to compute its uncertain reverse skyline. Thats is, the midpoints of the probabilistic products $p \in \mathcal{P}$ that defines the UD-domiance region of the query product q. This section presents an efficient approach to approximate the UD-dominance region of a probabilistic product by extending the R-Tree [8] based data indexing for probabilistic product databases, called PR-Tree, which can take advantage of Lemma 4.7 to compute its uncertain reverse skyline. The idea of PR-Tree is to augment each R-Tree node with the maximum and minimum probabilities of its children and store these probabilities in the tree node along with the links to its children. To construct the PR-Tree, we convert each product $p \in \mathcal{P}$ to its corresponding midpoint m and then, insert it in the tree. We also index the customer data by the general R-Tree, which is refereed as CR-Tree in this paper. We use R-Tree to denote either of the trees throughout this paper. In connection with computing the uncertain reverse skyline of a product q using R-Tree, we make the following statements.

- A midpoint m is said to have the same orthant as an R-Tree node n, denoted by $O_q(m) = O_q(n)$, if all 2^d corners of node n have the same orthant w.r.t. q as m does w.r.t. q.
- An object m dynamically dominates a node n w.r.t. a query object q, denoted by $m \prec_q n$, if all 2^d corners of n is dynamically dominated by m w.r.t. q.
- The tree nodes are accessed in order of their distances to the query product q.

[5] A member of $UMSL(q)$ can represent more than one product in $\mathcal{P}$ theoretically.

4.3 Query Processing

This section describes how to process the uncertain reverse skyline query and the influence (score) of a probabilistic product through its uncertain reverse skyline in detail.

4.3.1 Uncertain Reverse Skyline. While computing the uncertain reverse skyline of an arbitrary probabilistic query product q, we prune a PR-Tree node as per the following lemma.

Lemma 4.11. *A PR-Tree node n is* pruned *if $\exists m' \in \mathcal{M}'$ such that (i) $O_q(m') = O_q(n)$, (ii) $m' \prec_q n$ and (iii) $Pr(q) < Pr(p') \vee Pr(q) \times (1 - Pr(p')) < Pr(p')$, where $\mathcal{M}'$ is the set of midpoints of the products $\mathcal{P}' \subseteq \mathcal{P}$ accessed so far in the PR-Tree while computing $UMSL(q)$ for $URS(q)$ and m' is the midpoint of the product $p' \in \mathcal{P}'$.*

Proof. As all 2^d corners of node n has the same orthant w.r.t. q as m does w.r.t. q (condition (i)) and any $m \in n$ is bounded by the corners of n, m must have the same orthant w.r.t. q as m' does. Also, as m' dynamically dominates n w.r.t. q and $m \in n$ is bounded by the corners of n, m' also dynamically dominates m w.r.t. q, i.e., $m' \prec_q m$. Therefore, $\forall c \in C$, if $m \prec_q c$ and $Pr^c_{DSky}(p) \geq Pr^c_{DSky}(q)$, we also get $m' \prec_q c$ and $Pr^c_{DSky}(p') \geq Pr^c_{DSky}(q)$ (condition (iii)), which implies n can be *pruned*. Hence, the lemma. □

While computing the uncertain reverse skyline of a product q, we prune a CR-Tree node as per the following lemma.

Lemma 4.12. *A CR-Tree node n is* pruned *if $\exists m \in UMSL(q)$ such that (i) $O_q(m) = O_q(n)$ and (ii) $m \prec_q n$.*

Proof. As all 2^d corners of node n has the same orthant w.r.t. q as m does w.r.t. q (condition (i)) and any customer $c \in n$ is bounded by the corners of n, the customer c must have the same orthant w.r.t. q as m does. Also, as m dynamically dominates n w.r.t. q (condition (ii)) and $c \in n$ is bounded by the corners of n, m dynamically dominates c w.r.t. q, i.e., $m \prec_q c$. Therefore, $\exists p \in \mathcal{P}$ such that $p \prec_c q$, where p is the corresponding product of the midpoint m and $Pr^c_{DSky}(p) \geq Pr^c_{DSky}(q)$ as $m \in UMSL(q)$, which implies node n can be *pruned*. Hence, the lemma. □

The steps of computing the uncertain reverse skyline of a product q, i.e., $URS(q)$, with R-Trees are listed as follows:

(1) Firstly, we convert the products $p \in \mathcal{P}$ into their midpoints m w.r.t. q and index them into a PR-Tree.
(2) We initialize $UMSL(q)$ to an empty set. Then, we retrieve the children of the root node of the PR-Tree and insert them into a mean-heap $\mathcal{H}^{\mathcal{P}}_q$. We repeatedly retrieve the front entry E from $\mathcal{H}^{\mathcal{P}}_q$ until $\mathcal{H}^{\mathcal{P}}_q$ becomes empty and do the following: ignore E iff $\exists m \in UMSL(q)$ such that (i) $O_q(m) = O_q(E)$ and (ii) $m \prec_q E$ (Lemma 4.11), otherwise, insert its children into $\mathcal{H}^{\mathcal{P}}_q$ if E is a non-leaf node, else add the midpoint m contained in E into $UMSL(q)$ iff $Pr(q) < Pr(p) \vee (Pr(q) \times (1 - Pr(p))) < Pr(p)$, where p is the corresponding product of the midpoint m in $\mathcal{P}$.

Algorithm 1: Uncertain Reverse Skyline

```
   Input  : q: query, P: products, C: customers
   Output : URS(q): uncertain reverse skyline of q
1  begin
2      M ←convertProductsToMidpoints(P);                    // convert to midpoints
3      root^P ←constructPRTree(M);                          // create PR-Tree
4      UMSL(q) ← ∅;                                         // initialization
5      H_q^P ← insert(H_q^P, children(root^P));             // mean heap
6      while H_q^P ≠ ∅ do
7          E ←retrieveFront(H_q^P);
8          if ∃m ∈ UMSL(q) : O_q(E) = O_q(m) and m ≺_q E then
9              continue;                                    // prune PR-Tree node as per Lemma 4.11
10         else if !E.isLeaf() then
11             H_q^P ← insert(H_q^P, children(E));          // non-leaf PR-Tree node
12         else if Pr(q) < Pr(E) ∨ (Pr(q) × (1 − Pr(E))) < Pr(E) then
13             UMSL(q) ← add(UMSL(q), E);                   // UMSL member
14     URS_can(q) ← ∅;                                      // initialization
15     root^C ←constructCRTree(C);                          // create CR-Tree
16     H_q^C ← insert(H_q^C, children(root^C));             // mean heap
17     while H_q^C ≠ ∅ do
18         E ←retrieveFront(H_q^C);
19         if ∃m ∈ UMSL(q) : O_q(m) = O_q(E) and m ≺_q E then
20             continue;                                    // prune CR-Tree node as per Lemma 4.12
21         else if !E.isLeaf() then
22             H_q^C ← insert(H_q^C, children(E));          // non-leaf CR-Tree node
23         else
24             URS_can(q) ← add(URS_can(q), E);             // candidate of URS(q)
25     URS(q) ← verifyAndRefine(URS_can(q));                // verify and finalize URS(q)
```

(3) We index the customer data into a CR-Tree and initialize the candidate uncertain reverse skyline $URS_{can}(q)$ to an empty set. Then, we retrieve the children of the root node of the CR-Tree and insert them into a mean-heap $\mathcal{H}_q^C$. We repeatedly retrieve the front entry E from $\mathcal{H}_q^C$ until $\mathcal{H}_q^C$ becomes empty and do the following: ignore E iff $\exists m \in UMSL(q)$ such that (i) $O_q(m) = O_q(E)$ and $m \prec_q E$ (Lemma 4.12), otherwise, insert its children into $\mathcal{H}_q^C$ if E is a non-leaf node, else add the c contained in E into the candidate uncertain reverse skyline set $URS_{can}(q)$.

(4) Lastly, we verify and refine $URS_{can}(q)$ to finalize $URS(q)$ as follows: we run a window query bwteen q and each customer $c \in URS_{can}(q)$ in PR-Tree centered at c, denoted by $win(c, q)$, and add the customer c to $URS(q)$ iff $\nexists p \in win(c, q)$ such that $Pr^c_{DSky}(p) \geq Pr^c_{DSky}(q)$.

The above steps are pseudocoded in Algorithm 1.

Lemma 4.13. *Algorithm 1 computes accurately the uncertain reverse skyline of an arbitrary probabilistic product q.*

Proof. The computation of the uncertain reverse skyline of q, i.e., $URS(q)$, starts scanning the products $p \in \mathcal{P}$, then converting them into their corresponding midpoints w.r.t. q and thereafter, inserting them into the PR-Tree as given in lines 2-3. Then, we initialize $UMSL(q)$ to $\emptyset$ and insert the children of PR-Tree root into the min-heap $\mathcal{H}_q^{\mathcal{P}}$ in lines 4-5. The lines 6-13 repeatedly retrieve the front entry E of $\mathcal{H}_q^{\mathcal{P}}$ until $\mathcal{H}_q^{\mathcal{P}}$ is empty and prune E (PR-Tree node) as per Lemma 4.11, otherwise, insert the children of E into $\mathcal{H}_q^{\mathcal{P}}$ if E is an internal node, else add the midpoint m contained in E (leaf node) into the $UMSL(q)$ only if $Pr(q) < Pr(p) \vee Pr(q) \times (1 - Pr(p)) < Pr(p)$ to make sure that if $\exists c \in C$ such that $p \prec_c q$ and $Pr^c_{DSky}(p) > Pr^c_{DSky}(q)$ hold, c can be pruned by m as per Lemma 4.7, where p is the corresponding product of m in $\mathcal{P}$. As the entries (PR-Tree nodes) in $\mathcal{H}_q^{\mathcal{P}}$ are accessed in order of their distances to q, the $UMSL(q)$ computed in lines 6-13 is minimal and correct. Now, we initialize $URS_{can}(q)$ to $\emptyset$, constrcut CR-Tree of the customers C and insert the children of the CR-Tree root into the min-heap $\mathcal{H}_q^C$ in lines 14-16. The lines 17-24 repeatedly retrieve the front entry E of $\mathcal{H}_q^C$ until $\mathcal{H}_q^C$ is empty and prune E (CR-Tree node) as per Lemma 4.12, otherwise, insert the children of E into $\mathcal{H}_q^C$ if E is an internal node, else add the customer c contained in E (leaf node) into the $URS_{can}(q)$. Lastly, we verify and refine $URS_{can}(q)$ to finalize $URS(q)$ as per Definition 3.1 in line 25. This is because the products in $win(c, q)$ result are the only products in $\mathcal{P}$ that dynamically dominate q w.r.t. the customer c and may jointly prune the customer c from the $URS(q)$. Hence, the lemma. □

Algorithm 2: Influence Score

```
   Input  : q: query, P: products, C: customers
   Output : τ(q): influence score of q
1  begin
2      URS(q) ←computeURS(q, P, C);                         // Algorithm 1
3      τ(q) ← 0;                                            // initialization
4      for each c ∈ URS(q) do
5          UDS(c) ←computeUDS(c, P ∪ q);                    // Approach in [26]
6          score ← 0; // initialization
7          for each p ∈ UDS(c) do
8              score ← score + Pr^c_DSky(p);                // denominator of Eq. 2
9          τ(q) ← τ(q) + Pr^c_DSky(q) / score;              // Eq. 4
```

4.3.2 Influence Score. As per Lemma 3.5, we need to compute the dynamic skyline probability of each product $p \in UDS(c)$ for each $c \in URS(q)$ to compute the influence score $\tau(q)$ of the query product q. To achieve this, we first compute the uncertain reverse skyline of q, i.e., $URS(q)$ by Algorithm 1. Then, we compute the dynamic skyline probability of each product $p \in UDS(c)$ for each $c \in URS(q)$ as per the approach proposed in [26]. This idea is pseudocoded in Algorithm 2. Though, we adopt the approach proposed in [26] for computing the dynamic skyline probability in Algorithm 2, there is a significant difference between our approach and the approach proposed in [26] for computing $\tau(q)$. The approach proposed in [26] computes the $UDS(c)$ of each customer $c \in C$ irrespective of whether c is in $URS(q)$ or not to compute $\tau(q)$, which we don't do in our approach. Therefore, our approach is more efficient than the naïve approach proposed in [26] for computing the influence score $\tau(q)$ of an arbitrary query product q in probabilistic databases.

4.3.3 Optimization. Assume that n_{far} is the farthest and n_{near} is the nearest corner of a node n w.r.t. q as shown by the green-colored bulleted points in Fig. 3. If n is a PR-Tree node, also assume that $Pr(n_{far}) = min\{Pr(p), \forall m \in n\}$ and $Pr(n_{near}) = max\{Pr(p), \forall m \in n\}$, where p is the product of the midpoint m. The following lemma guides how to prune a PR-Tree node by comparing it with another PR-Tree node while computing the uncertain midpoint skyline of an arbitrary query product q, i.e., $UMSL(q)$.

Lemma 4.14. *A PR-Tree node n' can be pruned if $\exists n \in PR-Tree$ such that (i) $O_q(n) = O_q(n')$, (ii) $n_{far} \prec_q n'_{near}$ and (iii) $Pr(q) < Pr(n_{far}) \vee Pr(q) \times (1 - Pr(n_{far})) < Pr(n_{far})$.*

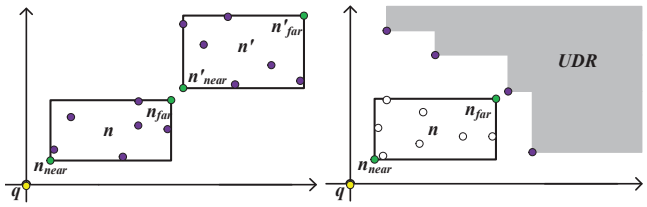


Figure 3: Optimization (a) a node prunes another node in PR-Tree node for $UMSL(q)$ and (b) adding each customers in a CR-Tree node into $URS(q)$, where purple-colored bulleted points represent uncertain midpoint skyline points of q

Proof. Assume that $\exists m' \in n'$ and $\exists c \in C$ such that $m' \prec_q c$ and $Pr^c_{DSky}(p') > Pr^c_{DSky}(q)$, where p' is the corresponding product in $\mathcal{P}$ of the midpoint m', i.e., $c \notin URS(q)$. Now, there must exist a midpoint $m \in n$ such that $m \prec_q c$ because of conditions (i) and (ii) as follows: $m \prec_q n_{far} \wedge n_{far} \prec_q n'_{near} \wedge n'_{near} \prec_q m' \wedge m' \prec_q c \rightarrow m \prec_q c$ (transitivity of dominance). Now, $Pr(q) < Pr(p) \vee Pr(q) \times (1 - Pr(p)) < Pr(p)$ because of condition (iii), where p is the corresponding product in $\mathcal{P}$ of the midpoint m, which implies $Pr^c_{DSky}(p) > Pr^c_{DSky}(q)$. Therefore, we can still prune the customer c by $m \in n$ even if we prune n'. Hence, the lemma. □

Lemma 4.15. *The customers c in a CR-Tree node n can be safely added to $URS_{can}(q)$ if $\nexists m \in UMSL(q)$ such that the followings hold: (i) $O_q(m) = O_q(n)$ and (ii) $m \prec_q n_{far}$.*

Proof. Assume that $\exists c \in n$ and conditions (i)-(ii) are true, but $c \notin URS_{can}(q)$. We prove that $URS_{can}(q)$ is potentially incorrect. As $\nexists m \in UMSL(q)$ such that $m \prec_q n_{far}$ and c is bounded within the region of node n, we get $m \not\prec_q c$. Therefore, c could be in the final uncertain reverse skyline of q. Hence, the lemma. □

The above optimization heuristics, i.e., Lemma 4.14 and Lemma 4.15 are pseudocoded in Algorithm 3. The difference between Algorithm 1 and Algorithm 3 is that Algorithm 3 applies PR-Tree node to node pruning on $\mathcal{H}^{\mathcal{P}}_q$ after inserting the children of an entry E into $\mathcal{H}^{\mathcal{P}}_q$ while computing $UMSL(q)$ (lines 10-12) and adds all c of a CR-Tree non-leaf node E into $URS_{can}(q)$ if the conditions in Lemma 4.15 are satisfied without inserting the children into $\mathcal{H}^{C}_q$ (lines 22-23). The optimization of influence score computation in Algorithm 2 is done by replacing Algorithm 1 with Algorithm 3 in line 2 for computing the uncertain reverse skyline of q.

5 PARALLEL APPROACH

This section presents an efficient approach of computing the uncertain reverse skyline and the influence score of a product by parallelizing their evaluations for today's data intensive systems involving millions of customer objects.

5.1 Computing Environment

We assume a simplified computing environment for evaluating uncertain reverse skyline queries in parallel in which a master processor, denoted by $\mathcal{T}_0$, is responsible for coordinating and managing

Algorithm 3: Optimized Uncertain Reverse Skyline

```
   Input  : q: query, P: products, C: customers
   Output : URS(q): uncertain reverse skyline of q
1  begin
2      M ←converProductsToMidpoints(P);                    // convert to midpoints
3      root^P ←constructPRTree(M);                          // create PR-Tree
4      UMSL(q) ← ∅;                                         // initialization
5      H^P_q ← insert(H^P_q, children(root^P));             // mean heap
6      while H^P_q ≠ ∅ do
7          E ←retrieveFront(H^P_q);
8          if ∃m ∈ UMSL(q) : O_q(E) = O_q(m) and m ≺_q E then
9              continue;                                    // prune PR-Tree node as per Lemma 4.11
10         else if !E.isLeaf() then
11             H^P_q ← insert(H^P_q, children(E));          // non-leaf PR-Tree node
12             H^P_q ←applyNodeToNodePruning(H^P_q);        // Lemma 4.14
13         else if Pr(q) < Pr(E) ∨ (Pr(q) × (1 − Pr(E))) < Pr(E) then
14             UMSL(q) ← add(UMSL(q), E);                   // UMSL member
15     URS_can(q) ← ∅;                                      // initialization
16     root^C ←constructCRTree(C);                          // create CR-Tree
17     H^C_q ← insert(H^C_q, children(root^C));             // mean heap
18     while H^C_q ≠ ∅ do
19         E ←retrieveFront(H^C_q);
20         if ∃m ∈ UMSL(q) : O_q(m) = O_q(E) and m ≺_q E then
21             continue;                                    // prune CR-Tree node as per Lemma 4.12
22         else if ∄m ∈ UMSL(q) : O_q(m) = O_q(E) and m ≺_q E_far then
23             URS_can(q) ← add(URS_can(q), customers(E));  // Lemma 4.15
24         else
25             H^C_q ← insert(H^C_q, children(E));          // non-leaf CR-Tree node
26     URS(q) ← verifyAndRefine(URS_can(q));                // verify and finalize URS(q)
```

the independent tasks carried out by the worker processors, denoted by $\{\mathcal{T}_j\}$. A worker processor $\mathcal{T}_j$ receives input data from the master and the task type, finishes the task accordingly and sends the processed result back to the master processor. The master processor may pre-process the input data before sending them to the workers. The master processor $\mathcal{T}_0$ finalizes the result in one or more rounds. We also assume that the communications and synchronizations between the master processor and the worker processors are integral part of this environment, and the computing powers of all worker processors are the same.

5.2 Parallel Uncertain Reverse Skyline

The parallel steps of computing the uncertain reverse skyline of a probabilistic product q, i.e., $URS(q)$, are listed as follows:

(1) In the first round, the master divides $\mathcal{P}$ into chunks $\mathcal{P}_j \subset \mathcal{P}$ (such that $\cup\mathcal{P}_j = \mathcal{P}$) and sends these chunks $\mathcal{P}_j$ and the query product q to its workers.
(2) A worker processor converts the products $p \in \mathcal{P}_j$ into their midpoints m w.r.t. q and index them into its local PR-Tree. Then, the worker computes the local uncertain midpoint skyline $UMSL_j$ by following the same technique as given in Step (2) in Section 4.3.1.
(3) Then, the master does the followings: (i) collects all local $UMSL_j$s from its workers and inserts them into a min heap $\mathcal{H}^{\mathcal{P}}_q$; (ii) initializes $UMSL(q)$ to $\emptyset$ and (iii) repeatedly retrieves the front entry m from $\mathcal{H}^{\mathcal{P}}_q$ until it becomes empty and does the following: adds m to $UMSL(q)$ if $\nexists m' \in UMSL(q)$ such that: $O_q(m') = O_q(m)$ and $m' \prec_q m$, otherwise ignores m.

Algorithm 4: Parallel Uncertain Reverse Skyline

```
   Input  : q: query, P: products, C: customers
   Output : URS(q): uncertain reverse skyline of q
1  begin
2      T_0 :partitionProductData(P);                          // partition product data
3      T_0 : parallel for each P_j do
4          sendQuery(T_j, q);                                 // send query product q to T_j
5          sendProducts(T_j, P_j);                            // send product subset P_j to T_j
6          T_j : M_j ←convertProductsToMidpoints(P_j);        // midpoints
7          T_j : root_j^P ←constructPRTree(M_j);              // create local PR-Tree
8          UMSL_j ← T_j : localMidpointSkyline(q, root_j^P);  // local UMSL
9      T_0 : UMSL(q) ←globalMidpointSkyline(q, ∪UMSL_j);      // global UMSL(q)
10     T_0 :partitionCustomerData(C);                         // partition customer data
11     T_0 : parallel for each C_j do
12         sendGlobalMidpointSkyline(T_j, UMSL(q));           // send global UMSL(q)
13         sendCustomers(T_j, C_j);                           // send customer subset C_j to T_j
14         T_j : root_j^C ←constructCRTree(C_j);              // create local CR-Tree
15         URS_can_j ← T_j : localURS_can(q, root_j^C, UMSL(q)); // local URS_can
16     T_0 : URS_can(q) ← ∪URS_can_j;                         // global URS_can(q)
17     T_0 : parallel for each c ∈ URS_can(q) do
18         sendCustomer(T_j, c);                              // send customer c to T_j
19         win_j(c, q) ← T_j : localWindow(c, q, root_j^P);   // compute local win(c, q)
20     T_0 : win(c, q) ← ∪win_j(c, q);                        // global win(c, q)
21     T_0 : URS(q) ← ∅;                                      // initialization
22     T_0 : parallel for each c ∈ URS_can(q) do
23         if ∄p ∈ win(c, q) : Pr^c_DSky(p) ≥ Pr^c_DSky(q) then
24             URS(q) ← add(URS(q), c);                       // c is a member of URS(q)
```

(4) In the second round, the master divides C into chunks $C_j \subset C$ (such that $\cup C_j = C$) and sends these chunks C_j and the global $UMSL(q)$ to its workers.
(5) A worker processor indexes C_j into its local CR-Tree. Then, the worker computes the local candidate reverse skyline URS_{can_j} by following the same technique as given in Step (3) in Section 4.3.1
(6) The master collects all local URS_{can_j}s from its workers into the global $URS_{can}(q)$.
(7) In the third round, the master asks all workers to process $win(c, q)$ for each $c \in URS_{can}(q)$ to finalize $URS(q)$ as given in Step (4) in Section 4.3.1.

The above steps are pseudocoded in Algorithm 4 as explained below. The master $\mathcal{T}_0$ partitions the product data $\mathcal{P}$ equally for the workers in line 2. The master then sends the query product q and the partitioned data $\mathcal{P}_j$ to the corresponding worker processor $\mathcal{T}_j$ in lines 4-5. In lines 6-8, the worker $\mathcal{T}_j$ converts $\mathcal{P}_j$ into the corresponding midpoints $\mathcal{M}_j$, constructs the local PR-Tree $root_j^{\mathcal{P}}$ and computes the local uncertain midpoint skyline $UMSL_j$ by calling *localMidpointSkyline(q, $root_j^{\mathcal{P}}$)* method which implements Step (2). Once computed, $\mathcal{T}_j$ sends the local $UMSL_j$ to the master $\mathcal{T}_0$ in line 8. The master $\mathcal{T}_0$ computes the global uncertain midpoint skyline $UMSL(q)$ by calling *globalMidpointSkyline(q, $\cup UMSL_j$)* method which implements Step (3) in line 9. The master processor $\mathcal{T}_0$ now partitions the customer data C equally for the workers in line 10 and then, sends the global $UMSL(q)$ and C_j to the corresponding worker $\mathcal{T}_j$ in lines 12-13. The worker $\mathcal{T}_j$ constructs the local CR-Tree $root_j^C$ and computes the local URS_{can_j} by calling method *localURS$_{can}$(q, $root_j^C$, UMSL(q))* which implements step (5) in lines 14-15. The local URS_{can_j} are accumulated by the master $\mathcal{T}_0$ into the global $URS_{can}(q)$ in line 16. Then, the master call the workers to compute the local $win_j(c, q)$ for each cutomer $c \in URS_{can}(q)$ in lines 17-19. The local $win_j(c, q)$s are collected into the global $win(c, q)$, and $win(c, q)$ result are used to verify and refine $URS_{can}(q)$ to finalize $URS(q)$ in lines 20-24 of Algorithm 4.

Lemma 5.1. *The Algorithm 4 accurately computes the uncertain reverse skyline of an arbitrary query product q.*

Proof. Firstly, we prove that the global uncertain midpoint skyline, i.e., $UMSL(q)$ computed by Algorithm 4 is correct. The local midpoint skyline $UMSL_j$ of q is correct for the partition $\mathcal{P}_i$ as we prove for $\mathcal{P}$ in Algorithm 1. Now, Algorithm 4 computes the global $UMSL(q)$ by accumulating the local $UMSL_j$s into the mean heap $\mathcal{H}_q^{\mathcal{P}}$ and thereafter, accessing the midpoints in $\mathcal{H}_q^{\mathcal{P}}$ in order of their distances to q. A midpoint m is added to the global $UMSL(q)$ iff it's filtering capability cannot be achieved by another midpoint m already existing in $UMSL(q)$. Therefore, the global $UMSL(q)$ can filter the customers $c \in C$ that would be filtered by local $UMSL_j$s, i.e., the global $UMSL(q)$ is correct and minimal. Then, the worker computes the local URS_{can_j} for the customer set $c \in C_j$ based on the global $UMSL(q)$ as we compute $URS_{can}(q)$ for C in Algorithm 1. As the selection of customers in the uncertain reverse skyline set of q are mutually independent, the global $URS_{can}(q)$ accumulated in the master is correct. Finally, the master computes the local window query result $win_j(c, q)$ for each $c \in URS_{can}(q)$ in each partition $\mathcal{P}_j$, which are accumulated by the master into the global $win(c, q)$ to finalize $URS(q)$. Hence, the lemma. □

5.3 Parallel Influence Score

This section presents an approach for computing the influence score of an arbitrary query product q in parallel. Our approach is significantly different from the approach proposed in [26]. The approach in [26] computes the favorite probability $Pr^c_{Fav}(q)$ by executing the uncertain dynamic skyline query of each $c \in C$ in different processing nodes without partitioning $\mathcal{P}$. In our approach, we partition not only C, but also $\mathcal{P}$, and execute the uncertain dynamic skyline query only for $c \in URS(q)$, not for each $c \in C$ as suggested in Lemma 3.5. Our approach is described below.

Firstly, we compute $URS(q)$ by calling Algorithm 4. Then, each worker constructs the PR-Tree on $\mathcal{P}_j$ without converting it to midpoints. Then, we compute two sets of products UDS_j and UDS_{can_j} for each customer $c \in URS(q)$ on each partition $\mathcal{P}_j$ locally by following the same technique described in [26]. Once the local UDS_j and UDS_{can_j} product sets are calculated, we accumulate them into the sets UDS and UDS_{can} in the master. We move a product p' from UDS to UDS_{can} iff $\exists p \in UDS$ such that $p \neq p'$ and $p \prec_c p'$. We also update UDS_{can} by ignoring all products $p' \in UDS_{can}$ iff $\exists p \in UDS_{can}$ such that $p \neq p'$ and $p \prec^u_c p'$.

Once the UDS and UDS_{can} product sets are computed for each $c \in URS(q)$, we update the dynamic skyline probabilities of the UDS_{can}[6] product set in parallel. To achieve this, firstly we compute the dominating points for each $p \in UDS_{can}$ on each partition $\mathcal{P}_j$ by running a window query for it locally. Once done for each

[6]The dynamic skyline probability of a $p \in UDS^c$ is $Pr(p)$ i.e., $Pr^c_{DSky(p)} = Pr(p)$, as $\nexists p' \in \mathcal{P}$ such that $p' \prec_c p$.

Algorithm 5: Parallel Influence Score

```
Input  : q: query, P: products, C: customers
Output : τ(q): influence score of q
1  begin
2    URS(q) ←parallelURS(q);                                   // Algorithm 4
     // partitioned data P_j is already sent to workers as part of Algorithm 4
3    T_0 : parallel for each P_j do
4      T_j : root_j^P ←constructPRTree(P_j);                   // local PR-Tree on P_j
5    T_0 : parallel for each c ∈ URS(q) do
6      T_0 : parallel for each T_j do
7        T_j: localUDS(c, root_j^P);  // compute local UDS and UDS_can points
8        UDS_j^c ← T_j: sendLocalUDSPoints();                  // local UDS points
9        UDS_can_j^c ← T_j: sendLocalUDScanPoints(); // local UDS_can points
10   T_0 : parallel for each c ∈ URS(q) do
11     T_j ← selectAnyAvailableWorker({T_j});                  // select a free worker
12     T_j: globalUDS(c, (∪UDS_j^c) ∪ q, ∪UDS_can_j^c); // compute global UDS and
         UDS_can points
13     UDS^c ← T_j: sendGlobalUDSPoints();                     // global UDS points
14     UDS_can^c ← T_j: sendGlobalUDScanPoints();   // global UDS_can points
15   T_0 : parallel for each c ∈ URS(q) do
16     T_0 : parallel for each T_j do
17       sendUDScanPoints( T_j, UDS_can^c);    // send UDS_can points to T_j
18       T_j:dominatingPointSet(UDS_can^c, root_j^P);   // dominating products
19       UDS_can Dom_j^c ← T_j:sendLocalDomPoints();
20   τ(q) ← 0;                                                 // initialization
21   parallel for each c ∈ URS(q) do
22     updateDSkyProbs(UDS_can^c, ∪UDS_can Dom_j^c);   // update DSky probs
23     Pr_Fav^c(q) ←computeFavProb(UDS^c, UDS_can^c);          // Eq. 2
24     τ(q) ← τ(q) + Pr_Fav^c(q);                  // update τ(q) as per Eq. 4
```

partition, we update the dynamic skyline probabilities of the products UDS_{can} by their dominating products to refine it and finally, compute the favorite probability $Pr^c_{Fav}(q)$ of each $c \in URS(q)$ in the master. Once the favorite probabilities are computed, the influence score $\tau(q)$ of the query product q is computed by following Eq. 4. The above parallel steps are pseudocoded in Algorithm 5.

Lemma 5.2. *Algorithm 5 accurately computes the influence score of an arbitrary query product q in parallel.*

Proof. Here, we prove that we accurately compute UDS and UDS_{can} product sets for each customer $c \in URS(q)$ in Algorithm 5. The local UDS_j and UDS_{can_j} product sets are computed by following the same the technique as described in [26]. Once these sets are computed locally, we accumulated them in the master for further refinement. The refinement ensures that UDS set includes only non-dominating products for a customer $c \in URS(q)$. Similarly, the UDS_{can} set includes only products that are not UD-dominated by any other products. Finally, the algorithm computes the dominating products for each product $p \in UDS_{can}$ w.r.t. the customer c by executing window query on each partition $\mathcal{P}_j$ w.r.t. c and p. The discovery of these dominating products in each partition are independent from one partition to another. Therefore, the final UDS and UDS_{can} (along with the dominating products of each product $p \in UDS_{can}$) product sets are accurate. Hence, the lemma. □

5.4 Optimization

An optimized version of Algorithm 4 can be achieved by applying Lemma 4.14 and Lemma 4.15 while computing the local $UMSL$ and URS of q, respectively, as we apply these lemmas in Algorithm 3. An optimized version of Algorithm 5 can also be achieved by executing optimized version of Algorithm 4 while computing the URS of the query product q in line 2.

Table 1: Settings of parameters

Parameter	Values
Tested Datasets	Real (CarDB), Synthetic (UN, CO, AC)
Data Cardinality	2K, 4K, 6K, 8K, 10K, 100K, 1M, 3M, 5M, 7M, 10M
Dimensionality	2D, 3D, 4D, 5D, 6D
No. of Threads	1 ~ 15 (1 thread per processor)
MAX #entries in R-Tree Nodes	20, 30, 40, 50, 60 data objects

6 EXPERIMENTS

This section compares the efficiencies of different approaches for evaluating the uncertain reverse skyline queries and computing the influence score of an arbitrary product in probabilistic databases.

6.1 Datasets, Queries and Environment

Datasets. We evaluate the efficiency of our pruning ideas and techniques for processing the uncertain reverse skyline queries using real CarDB[7] data which consists of 2×10^5 car objects. The CarDB is a six-dimensional dataset with attributes: *make*, *model*, *year*, *price*, *mileage* and *location*. We consider only the three numerical attributes *year*, *price* and *mileage* in our experiments after normalizing them into the range $[0, 1]$. We randomly select half of the car objects as products and the rest as the customer preferences. We also assign random probabilities to the car objects. The synthetic data experiments include datasets: uniform (UN), correlated (CO) and anti-correlated (AC), consisting of varying number of products, customers and dimensions. The cardinalities of the synthetic datasets range from 2K to 10M. The dimensionality (d) of the datasets varies from 2 to 6.

Test Queries. The test queries are generated (synthetic) and selected (CarDB) randomly by following the distribution of the respective datasets. Again, the query products are assigned with random probabilities.

Computing Environment. We develop our algorithms in Java and execute them in Swinburne HPC system[8] with 1~15 processors and maximum 60GB main memory, where the parallel computing environment (master-worker) is simulated with Java multithreading and LOCK-based synchronization.

The above parameters are summarized in Table 1.

6.2 Tested Algorithms

To compare the efficiency of evaluating uncertain reverse skyline queries, we tested the following algorithms: Serial URS (SER-URS) - Algorithm 1, Optimized URS (OPT-URS) - Algorithm 3, Parallel URS (PAR-URS) - Algorithm 4 and Optimized Parallel URS (PAR-URS*) - Optimized Algorithm 4. The naïve algorithm in [26] and its parallel version are called Naïve-URS and Naïve-PAR-URS, respectively. To improve the performance of naïve algorithm in [26], we do not update the dynamic skyline probabilities of the products that appear in the UDS_{can} set if q is not dominated by any $p \in \mathcal{P}$. To compare the efficiency of computing the influence score of a probabilistic product, we tested the efficiencies of the following algorithms: Serial Influence Score (SER-IS) - Algorithm 2, Optimized Influence Score (OPT-IS) - Optimized Algorithm 2, Parallel Influence Score (PAR-IS) - Algorithm 5 and Optimized Parallel Influence Score (PAR-IS*) -

[7] https://autos.yahoo.com/
[8] http://www.astronomy.swin.edu.au/supercomputing/

Table 2: Effect of customer cardinality on the efficiency of evaluating URS queries by different approaches

Cardinality	CarDB (millisecs)			UN (millisecs)			CO (millisecs)			AC (millisecs)		
	SER-URS	OPT-URS	Naïve-URS	SER-URS	OPT-URS	Naïve-URS	SER-URS	OPT-URS	Naïve-URS	SER-URS	OPT-URS	Naïve-URS
Customer(2K)	3017	2990	143803	2927	2991	140145	3684	2940	118851	3402	3246	139054
Customer(4K)	3067	3123	281937	3084	3029	251026	3251	3046	238909	3399	3672	259967
Customer(6K)	3162	3136	419895	3233	3355	380060	3166	2913	337296	3402	3679	356604
Customer(8K)	3302	3288	597125	3186	3278	524370	3109	3106	457902	3443	3696	465955
Customer(10K)	3303	3246	749371	3468	3257	617057	3230	3222	545728	3837	4100	578158
Customer(100K)	5077	5196	not executed	4510	4756	not executed	4657	5134	not executed	5201	5167	not executed

Table 3: Effect of customer cardinality on the efficiency of computing influence scores by different approaches

Cardinality	CarDB (millisecs)			UN (millisecs)			CO (millisecs)			AC (millisecs)		
	SER-IS	OPT-IS	Naïve-IS	SER-IS	OPT-IS	Naïve-IS	SER-IS	OPT-IS	Naïve-IS	SER-IS	OPT-IS	Naïve-IS
Customer(2K)	5144	5149	1350344	2909	2907	550090	2797	2815	473691	2980	2829	507864
Customer(4K)	8438	8472	2636079	3067	2962	1288985	2872	2888	988031	3091	2978	1005732
Customer(6K)	11748	11516	3915923	6051	6011	1609840	2958	2920	1536300	3045	3015	1440399
Customer(8K)	11953	11923	5671686	6075	5998	2135613	2974	2911	2109738	3111	3207	1915065
Customer(10K)	12262	12054	5143220	5969	5930	3027367	2976	3116	2668434	3172	3157	2273715
Customer(100K)	13578	14116	not executed	10595	11701	not executed	9838	10173	not executed	9311	8430	not executed

Optimized Algorithm 5. The naïve algorithm in [26] and its parallel version are called Naïve-IS and Naïve-PAR-IS, respectively.

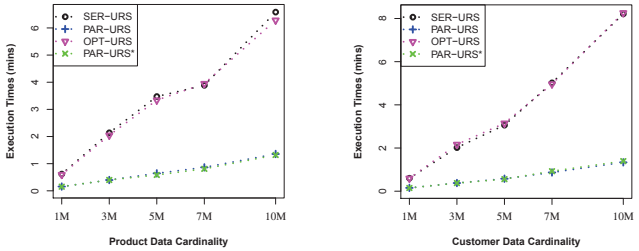
6.3 Efficiency Study

This section studies the efficiency of our proposed algorithms by comparing the execution times with the naïve approach proposed in [26] from the following perspectives.

6.3.1 Effect of data cardinalities. Here, we examine the effect of data cardinality (#customers) on the efficiency of processing uncertain reverse skyline queries and computing influence score of a probabilistic product by different approaches on the tested datasets. We set $|\mathcal{P}|$ = 100K, d = 2 and vary $|C|$ from 2K to 100K. We also set MAX #entries in a R-Tree node to 50. We run a number of queries and the results of evaluating a uncertain reverse skyline query and computing the influence score of a probabilistic product on average are shown in Table 2 and Table 3, respectively. It is evident that the naïve approach [26] is not scalable, whereas our approaches are scalable and can finish their executions within seconds even for 100K customers (naïve approach [26] is not executed as it takes hours to finish). We see that the speed-ups achieved by our approach over the naïve approach [26] are hugely significant.

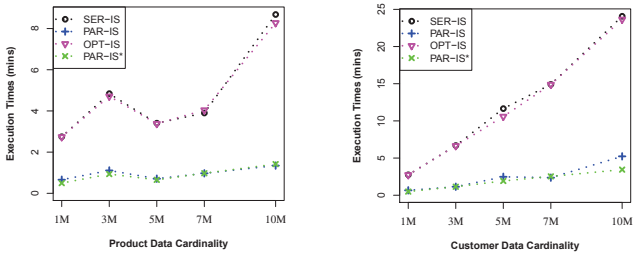
To justify the scalability of our approaches for millions of data objects, we perform another two experiments in UN dataset. For the first experiment, we set $|C|$ = 1M and vary $|\mathcal{P}|$ from 1M to 10M. For the second experiment, we set $|\mathcal{P}|$ = 1M and vary $|C|$ from 1M to 10M. For both experiments, we also set d = 2 and MAX #entries in a R-Tree node to 50. Finally, we run a number of queries and the results of evaluating a uncertain reverse skyline query and computing the influence score of a probabilistic product on average are shown in Fig. 4 and Fig. 5, respectively. We observe that our approaches can finish their executions within few minutes for millions of data objects.

6.3.2 Effect of data dimensions. Here, we examine the effect of data dimensionality on the efficiency of processing uncertain reverse skyline queries and computing the influence scores of probabilistic products by different approaches on CarDB two-dimensional (2D) and three-dimensional (3D) datasets. We set $|\mathcal{P}|$ = 100K, $|C|$=10K, #threads to 5 and 15 for PAR-URS, PAR-URS*, Naïve-PAR-URS, PAR-IS, PAR-IS* and Naïve-PAR-IS, and the MAX #entries in a R-Tree node to 50. We run a number of queries and the results of processing uncertain reverse skyline of a query and computing the influence score of a probabilistic product on average are shown in Fig. 6 and Fig. 7, respectively. We observe that the naïve approach[26] takes minutes to finish its execution in 3D data even with 15 threads (processors). The execution times get more worse for increased customer cardinality and dimensionality. On the other hand, all of our proposed approaches scale very well and finish their executions within seconds. We also perform another experiment in



(a) (b)

Figure 4: Effect of data cardinality on the efficiency of processing URS queries in UN dataset: (a) Product Cardinality and (b) Customer Cardinality



(a) (b)

Figure 5: Effect of data cardinality on the efficiency of computing influence scores in UN dataset: (a) Product Cardinality and (b) Customer Cardinality

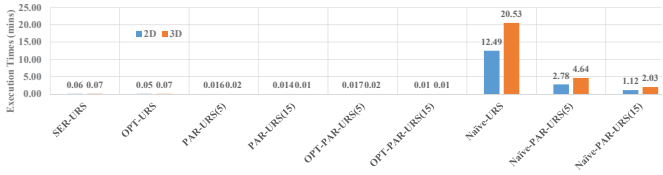


Figure 6: Effect of dimensions on the efficiency of processing reverse skyline queries in CarDB dataset

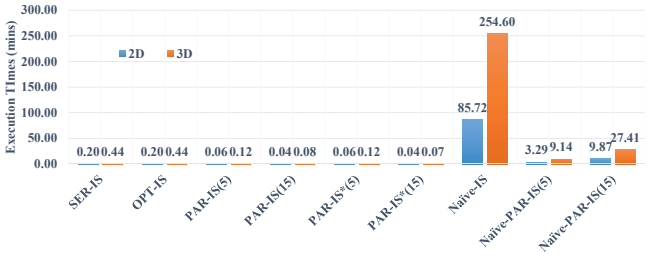


Figure 7: Effect of dimensions on the efficiency of computing influence scores in CarDB dataset

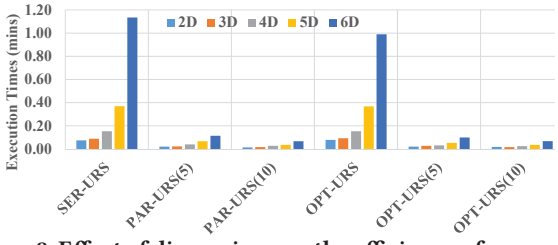
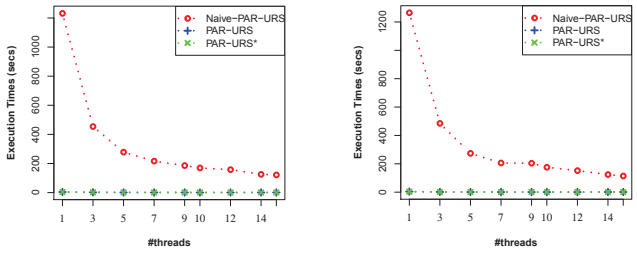
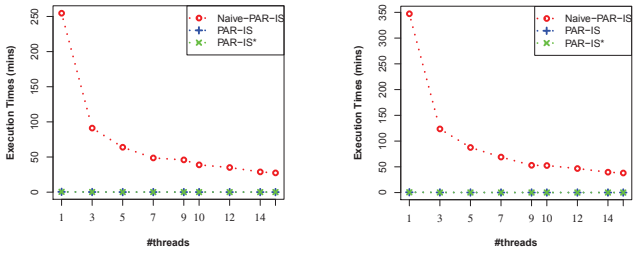


Figure 8: Effect of dimensions on the efficiency of processing reverse skyline queries in UN dataset



(a) CarDB (b) UN

Figure 9: Effect of #threads on the efficiency of URS queries: (a) CarDB and (b) UN datasets

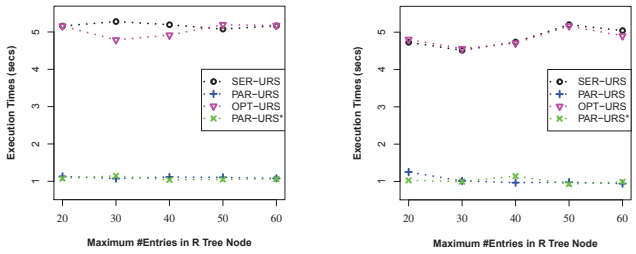


(a) CarDB (b) UN

Figure 10: Effect of #threads on the efficiency of influence scores in: (a) CarDB and (b) UN datasets

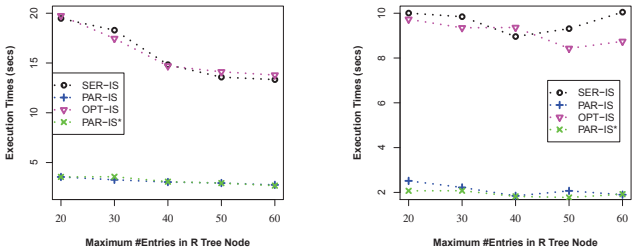
higher dimensions for UN dataset with varying d from 2 to 6 for testing the efficiency of evaluating the uncertain reverse skyline of a query. For this experiment, we set $|\mathcal{P}|$ and $|C|$ to 100K, and the MAX #entries in a R-Tree node to 50. The results are shown in Fig. 8. We observe that all of our approaches can finish their executions within 2 minutes. Therefore, we claim that our approaches are scalable even in higher dimensions.

6.3.3 Effect of threads. Here, we examine the effect of #threads on the efficiency of processing uncertain reverse skyline queries and computing the influence scores of probabilistic products in parallel by different approaches on CarDB and UN datasets. We set



(a) CarDB (b) AC

Figure 11: Effect of R-Tree #entries on the efficiency of URS queries: (a) CarDB and (b) AC datasets



(a) CarDB (b) AC

Figure 12: Effect of R-Tree #entries on the efficiency of influence scores in: (a) CarDB and (b) AC datasets

$|\mathcal{P}|$ = 100K, $|C|$=10K, d = 3, MAX #entries in a R-Tree node to 50 and vary #threads from 1 to 15. We run a number of queries and the results of evaluating an uncertain reverse skyline query and computing the influence score of a probabilistic product on average for different #threads are shown in Fig. 9 and Fig. 10, respectively. It is evident that the naïve approach[26] is not scalable even if we increase the #threads, whereas our approaches are scalable and can finish their executions within seconds with less #threads.

6.3.4 Effect of R-Tree parameters. Here, we examine the effect of R-Tree parameters (MAX #entries in a R-Tree node) on the efficiency of processing uncertain reverse skyline queries and computing the influence scores of probabilistic products by different approaches on CarDB and AC datasets. Here, we set $|\mathcal{P}|$ = 100K, $|C|$=100K, #threads to 10 for PAR-URS and PAR-URS*, d = 2 and vary MAX #entries in a R-Tree node from 20 to 60. We run a number of queries and the results of evaluating an uncertain reverse skyline query and computing the influence score of a probabilistic product on average are shown in Fig. 11 and Fig. 12, respectively. We observe that efficiency improves in general in SER-URS and OPT-URS with the increased MAX #entries in a R-Tree node. However, we observe an exception in their parallel evaluations. We also observe that the efficiencies of different approaches improve if we increase the MAX #entries in a R-Tree node in general except for SER-IS in AC dataset. We believe that the efficiency depends on many factors including data distribution in different threads (processors) and #threads, not only on the MAX #entries in a R-Tree node.

6.4 Summary

We experimentally demonstrate (prove theoretically in Section 3.1) that the naïve approach [26] is not scalable for computing the influence score of a probabilistic product. The computation of the influence score of a probabilistic product through uncertain reverse

skyline in uncertain data is scalable for millions of customer and product data objects, and can finish executions within few minutes.

7 RELATED WORK

Reverse Skyline Queries and Related Studies. Dellis et al. [5] are the first to present *reverse skyline* query to the database community. Later, Wu et al. [24] propose an efficient approach for computing the influence of a product through its reverse skyline, where the influence set consists of the member of the reverse skyline query results. Then, [6] proposes an approach for evaluating reverse skyline queries with non-metric similarity measures. Wang et al. [22] propose an energy efficient approach for evaluating reverse skyline queries over wireless sensor networks. Arvanitis et al. [2] extend this idea for computing the k-most attractive candidates from a given set of products that maximizes the size of their joint influence set (score). Islam et al. [12] propose an approach to answer how to turn up a given customer into the reverse skyline query result of an arbitrary query product. Recently, Islam et al. [10] present an approach for computing the k-most promising products, which assigns equal probabilities to the products appearing in the dynamic skyline of a customer and selects a subset of given products to maximize their joint probabilistic influence score. All of the above works are in certain data settings. Lian et al. [14], [15] extend the idea of reverse skyline query in uncertain data. However, the *probabilistic reverse skylines* proposed in [14], [15] lack friendliness, stability and fairness as per [26]. Zhou et al. [26] propose *uncertain dynamic skyline* and an approach to compute top-k favorite probabilistic products through uncertain dynamic skyline. However, the approach in [26] is not efficient as discussed in Section 3.1. This paper presents uncertain reverse skyline query to efficiently evaluate the influence of an arbitrary probabilistic product in uncertain data settings. Unlike [14], [15], the proposed *uncertain reverse skyline* is user friendly, stable and fair.

Parallelizing Reverse Skyline Queries. Though there exist many works on parallelizing the standard skyline queries ([9], [16], [1], [20], [3], [25] for survey), there are only few works devoted to parallelizing the reverse skyline queries. Park et al. [19] propose an approach for parallelizing both dynamic and reverse skyline queries in MapReduce by inventing a novel quad-tree based data indexing. Later, the authors extend their quad-tree based data indexing in [18] for evaluating probabilistic dynamic and reverse skylines. Recently, Islam et al. [11] propose an advancement of the quad-tree based data indexing proposed in [19] for evaluating the dynamic skyline, monochromatic and bichromatic reverse skylines in parallel. Here, we propose an efficient approach for parallelizing the computation of uncertain reverse skyline query result and the influence score of an arbitrary probabilistic product using R-Tree. Our approach for computing the influence score of a probabilistic product is significantly different from the one proposed in [26]. Here, we only compute the dynamic skyline probabilities of the products that appear in the uncertain dynamic skyline of the customers existing in the uncertain reverse skyline of the query product, not for all customers in the dataset.

8 CONCLUSION

This paper presents a novel skyline query, called uncertain reverse skyline, for measuring the influence of an arbitrary probabilistic product in uncertain data settings. We propose efficient pruning ideas and techniques for processing the uncertain reverse skyline and the influence score of a query product in probabilistic databases using R-Tree. We also present a parallel approach for evaluating the uncertain reverse skyline query and the influence score of a probabilistic product, which outperforms its serial counterpart. We conduct experiments with both real and synthetic datasets and compare our results with the existing baseline approach to demonstrate the efficiency of our approach.

9 ACKNOWLEDGMENT

The research of C. Liu and T. Anwar is supported by the ARC discovery projects DP160102412 and DP170104747.

REFERENCES

[1] Foto N. Afrati, Paraschos Koutris, Dan Suciu, and Jeffrey D. Ullman. 2015. Parallel Skyline Queries. *Theory Comput. Syst.* 57, 4 (2015), 1008–1037.
[2] Anastasios Arvanitis, Antonios Deligiannakis, and Yannis Vassiliou. 2012. Efficient influence-based processing of market research queries. In *CIKM*. 1193–1202.
[3] Kenneth S. Bøgh, Sean Chester, and Ira Assent. 2015. Work-Efficient Parallel Skyline Computation for the GPU. *PVLDB* 8, 9 (2015), 962–973.
[4] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator. In *ICDE*. 421–430.
[5] Evangelos Dellis and Bernhard Seeger. 2007. Efficient Computation of Reverse Skyline Queries. In *VLDB*. 291–302.
[6] Prasad M. Deshpande and Deepak Padmanabhan. 2011. Efficient reverse skyline retrieval with arbitrary non-metric similarity measures. In *EDBT*. 319–330.
[7] Scott Fay and Jinhong Xie. 2008. Probabilistic Goods: A Creative Way of Selling Products and Services. *Marketing Science* 27, 4 (2008), 674–690.
[8] Antonin Guttman. 1984. R-trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*. 47–57.
[9] Katja Hose and Akrivi Vlachou. 2012. A survey of skyline processing in highly distributed environments. *VLDB J.* 21, 3 (2012), 359–384.
[10] Md. Saiful Islam and Chengfei Liu. 2016. Know your customer: computing k-most promising products for targeted marketing. *The VLDB Journal* 25, 4 (2016), 545–570.
[11] Md. Saiful Islam, Chengfei Liu, Wenny Rahayu, and Tarque Anwar. 2016. Q+Tree: An Efficient Quad Tree based Data Indexing for Parallelizing Dynamic and Reverse Skylines. In *CIKM*. 1291–1300.
[12] Md. Saiful Islam, Rui Zhou, and Chengfei Liu. 2013. On answering why-not questions in reverse skyline queries. In *ICDE*. 973–984.
[13] Cuiping Li, Beng Chin Ooi, Anthony K. H. Tung, and Shan Wang. 2006. DADA: a data cube for dominant relationship analysis. In *SIGMOD*. 659–670.
[14] Xiang Lian and Lei Chen. 2008. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD*. 213–226.
[15] Xiang Lian and Lei Chen. 2010. Reverse skyline search in uncertain databases. *ACM Trans. Database Syst.* 35, 1 (2010), 3:1–3:49.
[16] Kasper Mullesgaard, Jens Laurits Pederseny, Hua Lu, and Yongluan Zhou. 2014. Efficient Skyline Computation in MapReduce. In *EDBT*. 37–48.
[17] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2003. An Optimal and Progressive Algorithm for Skyline Queries. In *SIGMOD*. 467–478.
[18] Yoonjae Park, Jun-Ki Min, and Kyuseok Shim. 2015. Processing of Probabilistic Skyline Queries Using MapReduce. *PVLDB* 8, 12 (2015), 1406–1417.
[19] Yoonjae Park, Jun-Ki Min, and Kyuseok Shim. 2013. Parallel Computation of Skyline and Reverse Skyline Queries Using MapReduce. *PVLDB* 6, 14 (2013), 2002–2013.
[20] Dimitris Pertesis and Christos Doulkeridis. 2015. Efficient skyline query processing in SpatialHadoop. *Information Systems* 54 (2015), 325–335.
[21] Mehdi Sharifzadeh and Cyrus Shahabi. 2006. The Spatial Skyline Queries. In *VLDB*. 751–762.
[22] Guoren Wang, Junchang Xin, Lei Chen, and Yunhao Liu. 2012. Energy-Efficient Reverse Skyline Query Processing over Wireless Sensor Networks. *IEEE Trans. Knowl. Data Eng.* 24, 7 (2012), 1259–1275.
[23] Tianyi Wu, Dong Xin, Qiaozhu Mei, and Jiawei Han. 2009. Promotion Analysis in Multi-Dimensional Space. *PVLDB* 2, 1 (2009), 109–120.
[24] Xiaobing Wu, Yufei Tao, Raymond Chi-Wing Wong, Ling Ding, and Jeffrey Xu Yu. 2009. Finding the influence set through skylines. In *EDBT*. 1030–1041.
[25] Ji Zhang, Xunfei Jiang, Wei-Shinn Ku, and Xiao Qin. 2016. Efficient Parallel Skyline Evaluation using MapReduce. *IEEE Trans. Parallel Distrib. Syst.* 27, 7 (2016), 1996–2009.
[26] Xu Zhou, Kenli Li, Guoqing Xiao, Yantao Zhou, and Keqin Li. 2016. Top k Favorite Probabilistic Products Queries. *IEEE Trans. Knowl. Data Eng.* 28, 10 (2016), 2808–2821.