Skyline Query on Uncertain Datasets Revisited

Yikai Zhang, Hengzhao Ma, Xingxing Xiao, Xiangyu Gao, Xiao Pan, Dongjing Miao, and Jianzhong Li

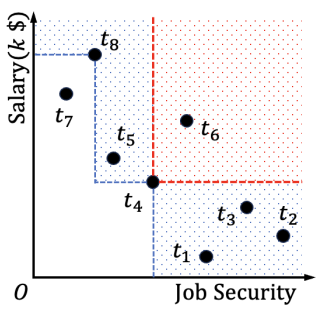
***Abstract*—With the rapid increase in the amount of uncertain data, probabilistic skyline computation on uncertain datasets has become an important research topic. To provide flexible control over the output size while ensuring the incomparability of the resulting tuples, the UC-SKY query has been recently introduced. It aims to identify a specified number of mutually non-dominated tuples with the highest probability of being included in the skyline. However, the computational complexity of the UC-SKY query remains unclear, and only an enumeration-based algorithm has been proposed to address UC-SKY queries. To fill this gap, we revisit the UC-SKY query in this paper. In 2D space, we design an exact algorithm for the UC-SKY query, indicating it belongs to P. We further improve the time and space complexity of this algorithm through grouping and pruning strategies. We formally prove that the UC-SKY query is NP-hard for arbitrary data dimension. In MD space, we present an efficient three-stage algorithm incorporating effective pruning strategies to exactly solve the UC-SKY query. As byproducts, we also introduce two approximation algorithms, which offer better time efficiency. Experimental results on real and synthetic datasets demonstrates the efficiency and scalability of our proposed algorithms.**

***Index Terms*—skyline query, uncertain datasets**

I. Introduction

SKYLINE queries aim to find all tuples in a dataset that are not dominated by any other tuple [1]. A tuple t dominates another tuple s if t is not worse than s in all dimensions and better than s in at least one dimension. The set of returned tuples is called the skyline of the dataset. Skyline queries are widely used in many real-world applications such as decision making, multi-criteria data analysis, and data mining. Due to privacy issues, noisy measurements, delayed data updates, etc., data uncertainty arises inherently in these applications. Hence, conducting skyline queries on uncertain datasets has received widespread studies in the past decade [2]–[11]. Generally, two categories of uncertain skyline queries have been proposed. The first one, called PSKY, aims to identify uncertain tuples that have a high probability of *being in the skyline* [2]–[6], [8], [10]. The second one, called USKY, focuses on finding a set of mutually non-dominated uncertain tuples that is most-likely to *be the skyline* [7], [11]. However, both of them have drawbacks, as demonstrated by the following examples.

Fig. 1 illustrates the concepts of PSKY and USKY using a job opportunities dataset. Each job is associated with a probability that particular job will be offered to the applicant [3].



Job	Probability of getting the offer
t_1	$p(t_1) = 0.8$
t_2	$p(t_2) = 0.2$
t_3	$p(t_3) = 0.3$
t_4	$p(t_4) = 0.9$
t_5	$p(t_5) = 1$
t_6	$p(t_6) = 0.5$
t_7	$p(t_7) = 0.5$
t_8	$p(t_8) = 0.4$

Fig. 1: An illustration of the two categories of uncertain skyline queries.

For each job t, PSKY defines its skyline probability $\Pr_{\text{psky}}(t)$ as the probability that t is offered to the applicant and none of jobs dominating t are offered [2]. Under the assumption of independence, for example, $\Pr_{\text{psky}}(t_4) = p(t_4) \times (1 - p(t_6))$. However, it is common that jobs with high $\Pr_{\text{psky}}(\cdot)$ identified by PSKY are mutually dominated. This can reduce the diversity of options available for further decision-making. In our example, the two jobs with the highest $\Pr_{\text{psky}}(\cdot)$ are t_5 and t_6, both with probability 0.5. But t_5 is dominated by t_6, which limits the range of viable choices despite high $\Pr_{\text{psky}}(\cdot)$.

Given a set of mutually non-dominated jobs S, USKY defines its skyline probability $\Pr_{\text{usky}}(S)$ as the probability that all jobs in S are offered to the applicant, and that other jobs not dominated by any job in S are not offered [7]. As an example, $\Pr_{\text{usky}}(\{t_4, t_8\}) = \prod_{i\in\{4,8\}} p(t_i) \times \prod_{j\in\{1,2,3,5,6\}}(1 - p(t_j))$. The goal of USKY is to find a set with the largest $\Pr_{\text{usky}}(\cdot)$. However, the size of the result returned by USKY is uncontrollable. As suggested by Hick's Law, controlling the size of the result presented to the user is essential for the further decision quality and the overall user experience [12], [13]. Dictating the result size is crucial also because of design considerations such as display size, device capabilities, connection speed, etc. What's more, $\Pr_{\text{usky}}(\cdot)$ of every job set can be quite small because it considers all jobs not dominated by any job in this set. Although the states of t_1, t_2, t_3 do not affect whether t_4 and t_8 are dominated, they still contribute to $\Pr_{\text{usky}}(\{t_4, t_8\})$.

The UC-SKY query has recently been introduced to bring together the strong points of PSKY and USKY, while avoiding their drawbacks [9]. Given an output size l, the UC-SKY query aims to identify a set of l mutually non-dominated uncertain tuples S that maximizes the probability of *being contained in the skyline*. Continuing with our example, the skyline probability $\Pr_{\text{uc-sky}}(S)$ of a set S is defined as the probability that all jobs in S are offered to the applicant, and that all other jobs that dominate some jobs in S are not offered. For example, $\Pr_{\text{uc-sky}}(\{t_4, t_8\}) = \prod_{i\in\{4,8\}} p(t_i) \times (1 - p(t_6))$. It is apparent

Xiangyu Gao is the corresponding author.

Yikai Zhang, Xiao Pan, and Xiangyu Gao are with the School of Information Science and Technology, Shijiazhuang Tiedao University, China. Email: 20224105@student.stdu.edu.cn, {smallpx, gaoxy}@stdu.edu.cn

Hengzhao Ma is with the Software College of Northeastern University, China. Email: mahz@swc.neu.edu.cn

Xingxing Xiao, Dongjing Miao, and Jianzhong Li are with the Faculty of Computing, Harbin Institute of Technology, China. Jianzhong Li is also with the Faculty of Computer Science and Control Engineering, Shenzhen University of Advanced Technology, China. Email: {xiaoxx, miaodongjing, lijzh}@hit.edu.cn

that the UC-SKY query is an output-size specified operator and tuples returned by the UC-SKY query are mutually non-dominated. Meanwhile, for every set S, $\Pr_{\text{uc-sky}}(S)$ is also greater than $\Pr_{\text{usky}}(S)$.

Despite the proposal of the UC-SKY query, the work in [9] faces two main issues that require further attention. First, the computational complexity of the UC-SKY query has not been fully explored. It remains unclear whether there is a condition under which UC-SKY queries can be solved in polynomial time. Second, only an enumeration-based algorithm is proposed to solve UC-SKY queries. While pruning strategies and a range-tree-like index are employed to reduce the data size and accelerate the computation of $\Pr_{\text{uc-sky}}(\cdot)$, respectively, the overall efficiency of the algorithm is still unsatisfactory.

In this paper, we revisit the UC-SKY query, providing both theoretical and practical insights to address above issues. By establishing a polynomial-time reduction from the minimum l-union problem [14], we prove that exactly solving UC-SKY queries is NP-hard. In 2D space, we design a polynomial-time exact algorithm based on dynamic programming, indicating that the UC-SKY query is in P for $d = 2$. We further improve the algorithm's time and space complexities by introducing grouping and pruning strategies. In multi-dimensional (MD) space, we propose an exact algorithm with three consecutive stages, i.e., the preprocessing stage, the dividing stage, and the branch-and-bound search stage. In the preprocessing stage, two approximation algorithms are proposed to compute a lower bound of the solution quality, and the input dataset is reduced to a smaller one based on this lower bound. In the dividing stage, the UC-SKY query is decomposed into multiple constrained subqueries, each applied to a much smaller dataset, using a non-trivial dividing strategy. Each subquery is then solved by performing a branch-and-bound search in the final stage, and the final result is selected among all subquery solutions. Although the algorithm has an exponentially increasing running time in the worst case, our experiments verify that its average performance is about two orders of magnitude higher than the state-of-the-art algorithms. Although the algorithm exhibits exponential time complexity in the worst case, our experimental results show that it outperforms the state-of-the-art algorithm by around two orders of magnitude on average.

The main contributions of this paper are listed below.

- We prove that the UC-SKY query is NP-hard.
- In 2D space, we propose an exact algorithm with time and space complexity $O(\max\{l, \log n\}n^2)$ and $O(n^2)$, where n is the number of uncertain tuples and l is the output size. We further optimize this algorithm using grouping and pruning strategies, reducing time and space complexity to $O(\max\{l, \log n\}n'm)$ and $O(nl)$, where n' and m are parameters smaller than n.
- In MD space, we present an efficient three-stage algorithm incorporating effective pruning strategies for exactly solving UC-SKY queries. In addition, we propose two approximation algorithms that offer better time efficiency at the cost of some solution quality.
- We introduce the UC-RSKY query, an extension of the UC-SKY query designed for serving personalization. We prove that the UC-RSKY query is NP-hard even when $d = 2$, and extend our algorithms to efficiently solve it.
- We conduct extensive experiments on both real and synthetic datasets to verify the efficiency and scalability of our algorithms.

II. Related Work

In recent years, significant research has been dedicated to addressing the problem of performing skyline queries on uncertain data. Pei et al. first studied skyline queries on uncertain data and introduced the PSKY query [2]. They proposed two algorithms to identify uncertain tuples whose skyline probabilities exceed a user-specified threshold p. However, it maybe challenging to specify an appropriate threshold, leading Atallah and Qi to investigate the problem of computing skyline probabilities for all instances [3]. They developed an $\tilde{O}(n^{2-1/(d+1)})$-time algorithm by combining two primary methods, i.e., the weighted dominance counting method and the grid method. They further improved the time complexity to $\tilde{O}(n^{2-1/d})$ by replacing the grid method with a more efficient sweeping method [4]. However, both algorithms are limited to 2D datasets due to a hidden factor that is exponential in the data dimension, which comes from the high dimensional weighted dominance counting algorithm. To address this limitation, Afshani et al. proposed a new algorithm based on a pre-order traversal of a modified KD-tree [8]. Leveraging the property of the KD-tree, they proved that the time complexity of their algorithm is $O(n^{2-1/d})$. More practically, Kim et al. introduced an in-memory Z-tree structure to reduce the number of dominance tests, which was experimentally demonstrated to be efficient [6]. To facilitate the control of the result size and accelerate computation, the problem of identifying uncertain tuples with top-k skyline probabilities was studied in [5], [15].

Recognizing that uncertain tuples with high skyline probabilities may dominate each other, the USKY query was introduced in [7], [11]. Liu et al. proved that when $d \geq 3$, the USKY query is NP-hard [7]. They proposed a search algorithm based on dynamic programming and introduced pruning and early termination strategies to further improve efficiency. They also designed several data reduction rules and data partition methods to pre-eliminate unnecessary tuples and reduce the input data size. Agrawal et al. proposed an $O(n \log n)$-time exact algorithm to solve the USKY query when $d = 2$ based on dynamic programming [11]. They also proved that when $d \geq 3$, the USKY query cannot be approximated within $2^{-O(m^{1-\delta})}$ in polynomial time for any $\delta > 0$, unless P = NP. And they also gave a 2^m-approximation algorithm.

However, as previously mentioned, the USKY query cannot control the output size, and the skyline probability of its result is typically small. To address this issue, Liu et al. introduced the UC-SKY query, which aims to identify a set of k uncertain tuples with the highest probability of being contained in the skyline [9]. They proposed an enumeration-based algorithm and employed pruning strategies, along with a range-tree-like index, to enhance efficiency. Our work resolves two main issues of the UC-SKY query. First, we conduct a comprehensive study of the computational complexity of

the UC-SKY query. We formally prove that the UC-SKY query belongs to P when $d = 2$ and is NP-hard for arbitrary data dimension. Second, we propose more efficient algorithms to solve UC-SKY queries both exactly and approximately. Our experimental results show that the proposed algorithms outperform the state-of-the-art algorithm by around two orders of magnitude on average.

In addition, the problem of conducting reverse skyline queries on uncertain datasets was explored in [16]. Zhang et al. addressed the challenge of performing skyline queries on uncertain data streams with a sliding window approach [17]. The authors of [18], [19] modeled data uncertainty using continuous probability distributions and proposed algorithms to solve skyline queries on such uncertain data. Zhang et al. studied uncertain skyline queries in a distributed environment [20] and Gavagsaz focused on uncertain skyline query processing on the Map-Reduce model [21].

III. Problem and Preliminaries

Let D be a d-dimensional dataset consisting of n tuples. Each tuple $t \in D$ has d numeric attributes, denoted by $t = (t[1], \cdots, t[d])$. Under the assumption that lower values are preferred than higher ones, a tuple t is said to *dominate* another tuple s, denoted by $t \prec s$, if $\forall 1 \leq i \leq d : t[i] \leq s[i]$, and $\exists 1 \leq j \leq d : t[j] < s[j]$. The *skyline* of D, denoted by $\mathrm{SKY}(D)$, is the subset of tuples that are not dominated by any other tuples in D, i.e., $\mathrm{SKY}(D) = \{t \in D \mid \forall s \in D : s \not\prec t\}$.

Following the uncertain setting used in [7], [11], we adopt the independent probability model to represent an uncertain dataset $\mathcal{D} = (T, p)$. Here T is the entire dataset consisting of n tuples, and $p : T \to (0, 1]$ is the function that determines the probability that the object or event represented by this tuple does occur. For each $t \in T$, we refer $p(t)$ as its *attendance probability*. This model assumes that all tuples' attendance probabilities are independent. Under the possible world semantics [22], a *possible world* D of $\mathcal{D}$ is a deterministic dataset which is a subset of T and the probability that D exists can be quantified as

$$\Pr(D) = \prod_{t \in D} p(t) \times \prod_{t \notin D} (1 - p(t)).$$

We use $D \sqsubseteq \mathcal{D}$ to denote that D is a possible world of $\mathcal{D}$. It is easy to verify that the total number of such possible world of $\mathcal{D}$ is $2^{|T|}$ and $\sum_{D \sqsubseteq \mathcal{D}} \Pr(D) = 1$.

Definition 1 (UC-SKY Probability [9]): Given an uncertain dataset $\mathcal{D} = (T, p)$, the U-Skyline containment (UC-SKY) probability of a set of mutually non-dominated tuples $S \subseteq T$ is defined as the accumulated existence probabilities of possible worlds that have S in their skyline, i.e.,

$$\Pr_{\text{uc-sky}}(S) = \sum_{D \sqsubseteq \mathcal{D}} \Pr(D) \times \mathbf{1}[S \subseteq \mathrm{SKY}(D)], \quad (1)$$

where $\mathbf{1}[\cdot]$ is the indicator function.

For any possible world $D \sqsubseteq \mathcal{D}$, $S \subseteq \mathrm{SKY}(D)$ if and only if all tuples in S appear in D and all tuples that dominate some tuples in S do not appear in D. Therefore, $\Pr_{\text{uc-sky}}(S)$ can be equivalently represented as

$$\Pr_{\text{uc-sky}}(S) = \prod_{t \in S} p(t) \times \prod_{t \in T \setminus S : \exists s \in S : t \prec s} (1 - p(t)). \quad (2)$$

The following lemma shows that the UC-SKY probability is anti-monotone.

Lemma 1: For any two sets of non-dominant tuples S_1, S_2, if $S_1 \subseteq S_2$, then $\Pr_{\text{uc-sky}}(S_1) \geq \Pr_{\text{uc-sky}}(S_2)$.

Proof: Since $S_1 \subseteq S_2$, we have $\{t \in T \setminus S_1 : \exists s \in S_1 : t \prec s\} \subseteq \{t \in T \setminus S_2 : \exists s \in S_2 : t \prec s\}$. Therefore, we know $\prod_{t \in S_1} p(t) \leq \prod_{t \in S_2} p(t)$, and $[\prod_{t \in T \setminus S_1 : \exists s \in S_1 : t \prec s} p(t) \leq \prod_{t \in T \setminus S_1 : \exists s \in S_1 : t \prec s} p(t)$. This completes the proof. ∎

Definition 2 (UC-SKY Query [9]): Given an uncertain dataset $\mathcal{D} = (T, p)$ and an integer l, the U-Skyline containment (UC-SKY) query aims to find a set of mutually non-dominant tuples S^* such that $|S^*| = l$ and $\Pr_{\text{uc-sky}}(S^*)$ is maximized, i.e.,

$$\text{UC-SKY}(\mathcal{D}, l) = S^* = \arg \max_{S \subseteq T : |S| = l} \Pr_{\text{uc-sky}}(S).$$

Hardness: We prove the UC-SKY query is NP-hard. Given a collection of sets $\mathcal{S} = \{S_1, \cdots, S_m\}$ and an integer l, the minimum l-union problem aims to find a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \geq l$ and the number of covered elements $|\bigcup_{S_i \in \mathcal{S}'}|$ is minimized. It was proved that the minimum l-union problem is NP-hard [14]. We first prove the Min-RSP problem is NP-hard by establishing a polynomial-time reduction from the minimum l-union problem.

Definition 3 (Min-RSP Problem): Given a set D of tuples and an integer l, the minimum representative skyline points (Min-RSP) problem aims to find a set $S \subseteq \mathrm{SKY}(D)$ with size l such that $|\{t \in D \mid \exists s \in S : s \prec t\}|$ is minimized.

Lemma 2: The Min-RSP problem is NP-hard.

Proof: Given a collection of sets $\mathcal{S} = \{S_1, \cdots, S_m\}$ and an integer l, the reduction constructs a set D of tuples and an integer l' as follows. Let $U = \bigcup_{S_i \in \mathcal{S}} S_i$ and $d = |U| + |\mathcal{S}|$. For ease of notation, we impose an arbitrary order on elements in U and denote the i-th element in U by u_i. For each element $u_i \in U$, a d-dimensional tuple t_i is inserted into D, where for $1 \leq j \leq d$, $t_i[j] = 0.5$ if $j = i$, otherwise $t_i[j] = 1$. For each set $S_i \in \mathcal{S}$, a d-dimensional tuple s_i is inserted into D. Here for $1 \leq j \leq |U|$, $s_i[j] = 0$ if $u_j \in S_i$, otherwise $s_i[j] = 1$. And for $|U| + 1 \leq j \leq d$, $s_i[j] = 1$ if $j = i + |U|$, otherwise $s_i[j] = 0$. Then, simply let $l' = l$. It is straight to verify that a element u_i is covered by a set S_j if and only if u_i is dominated by s_j. And the last $|\mathcal{S}|$ coordinates guarantee that $\mathrm{SKY}(D) = \{s_i \mid S_i \in \mathcal{S}\}$. This indicates that for any subset $\mathcal{S}' = \{S_{i_1}, \cdots, S_{i_l}\} \subseteq \mathcal{S}$ with size l, let $S = \{s_{i_1}, \cdots, s_{i_l}\} \subseteq \mathrm{SKY}(D)$, $\bigcup_{S_i \in \mathcal{S}'} = \{t \in D \mid \exists s \in S : s \prec t\}$, which completes the proof. ∎

Then, we establish a polynomial-time reduction from Min-RSP problem to UC-SKY query to prove the latter is NP-hard.

Theorem 1: The UC-SKY query is NP-hard.

Proof: Given a set D of tuples and an integer l, the reduction constructs an uncertain dataset $\mathcal{D} = (T, p)$ and an integer l' as follows. For ease of notation, an arbitrary order on tuples in D is imposed and the i-th tuple in D is denoted by t_i. First, for each tuple $t_i \in D$, a tuple s_i is inserted in T,

Algorithm 1: Basic DP Algorithm

Input: uncertain dataset $\mathcal{D} = (T, p)$, integer l
Output: UC-SKY$(\mathcal{D}, l)$

```
Sort T ← {t_1, ··· , t_n} in lexicographic order;
Construct a 2-dimensional range-tree 𝒯 on T;
foreach i ← 1 to n do
    foreach j ← 1 to i − 1 s.t. t_j ⊀ t_i do
        γ(t_i, t_j) ← (1 − p(t_i))^{-1} ×
          RangeQuery(𝒯, (t_j[1], t_i[1]], [0, t_i[2]]);
foreach k ← 1 to l do
    foreach i ← 1 to n do
        if k = 1 then
            Y(t_i, 1) ← t_i;
            cost(t_i, 1) ← p(t_i) × (1 − p(t_i))^{-1} ×
              RangeQuery(𝒯, [0, t_i[1]], [0, t_i[2]]);
        else
            Y(t_i, k) ← arg max_{t_j ∈ T_{i−1} : t_j ⊀ t_i} {γ(t_i, t_j) ×
              cost(t_j, k − 1)};
            cost(t_i, k) ← p(t_i) × γ(t_i, Y(t_i, k)) ×
              cost(Y(t_i, k), k − 1);
t_{i*} ← arg max_{t_i ∈ T} cost(t_i, l);
UC-SKY(𝒟, l) ← ∅;
foreach k ← l to 1 do
    UC-SKY(𝒟, l) ← UC-SKY(𝒟, l) ∪ t_{i*};
    t_{i*} ← Y(t_{i*}, k);
return UC-SKY(𝒟, l);
```

where for each $1 \le j \le d$, $s_i[j] = 1/t_i[j]$. And $p(s_i) = 1$ if $t_i \in \text{SKY}(D)$, otherwise, $p(s_i) = \frac{1}{3^n}$. Then, let $l' = l$. We first claim that only tuples in $T' = \{s_i \mid t_i \in \text{SKY}(D)\}$ exist in $S^* = \text{UC-SKY}(\mathcal{D}, l)$. Without loss of generality, suppose $a = |S^* \cap (T \setminus T')| > 0$, we have

$$\begin{aligned}\Pr_{\text{uc-sky}}(S^*) &= \left(\frac{1}{3^n}\right)^a \times \left(1 - \frac{1}{3^n}\right)^m \\ &\le \frac{1}{3^n} < \left(1 - \frac{1}{3^n}\right)^n \le \Pr_{\text{uc-sky}}(S'),\end{aligned}$$

where $m = |\{s_i \in T \mid \exists s \in S^*, s_i \prec s\}|$ and S' is an arbitrary subset of T' with size l. Since the optimal solution S^* only contains tuples in T', $\Pr_{\text{uc-sky}}(S^*) = \left(1 - \frac{1}{3^n}\right)^m$. Hence, an S^* with fewer $m = |\{s_i \in T \mid \exists s \in S^*, s_i \prec s\}|$ will lead to a higher $\Pr_{\text{uc-sky}}(S^*)$. According to the reduction, for any two tuples $t_i, t_j \in D$, $t_i \prec t_j$ if and only if $s_j \prec s_i$. Therefore, a set $S \subseteq \text{SKY}(D)$ with the least dominated tuples corresponds to the set $S^* \subseteq T'$ with the highest UC-SKY probability. ■

IV. Algorithms in 2D

In this section, we focus on 2D case. We first propose an exact algorithm based on dynamic programming, and then improve it with grouping and pruning strategies.

A. Basic Dynamic Programming Algorithm

Suppose tuples in T are lexicographically sorted, i.e., tuples in T are first sorted in ascending order based on $t[1]$, then tuples with the same $t[1]$ are sorted in ascending order based on $t[2]$. Let t_i denote the i-th tuple in T and $T_i = \{t_1, \cdots, t_i\}$ $(1 \le i \le n)$. It is easy to verify that tuples in T dominating t_i all belong to T_{i-1}. For each $1 \le k \le l$, $X(t_i, k)$ is defined as a subset of T_i such that (1) $|X(t_i, k)| = k$, (2) $t_i \in X(t_i, k)$, and (3) $\Pr_{\text{uc-sky}}(X(t_i, k))$ is maximized. When $k = 1$, apparently,

$$\begin{cases} X(t_i, 1) = \{t_i\}, \\ \Pr_{\text{uc-sky}}(X(t_i, 1)) = p(t_i) \times \prod\limits_{t_j \in T_{i-1} : t_j \prec t_i} (1 - p(t_j)). \end{cases} \quad (3)$$

Then, for each $t_j \in T_{i-1}$ such that $t_j \not\prec t_i$, let $\Delta(t_i, t_j) = \{t \in T_i \mid t \prec t_i \wedge t \not\prec t_j\}$ and $\gamma(t_i, t_j) = \prod_{t \in \Delta(t_i, t_j)} (1 - p(t))$. An example is illustrated in Fig. 2, where $\Delta(t_6, t_5) = \{t_4\}$ and $\gamma(t_6, t_5) = 1 - p(t_4)$. When $2 \le k \le l$, it is derived that

$$\begin{cases} X(t_i, k) = \{t_i\} \cup X(t_{j^*}, k-1), \\ \Pr_{\text{uc-sky}}(X(t_i, k)) = p(t_i)\gamma(t_i, t_{j^*})\Pr_{\text{uc-sky}}(X(t_{j^*}, k-1)), \end{cases} \quad (4)$$

where

$$t_{j^*} = \arg \max_{t_j \in T_{i-1} : t_j \not\prec t_i} \{\gamma(t_i, t_j) \times \Pr_{\text{uc-sky}}(X(t_j, k-1))\}. \quad (5)$$

Clearly,

$$\text{UC-SKY}(\mathcal{D}, l) = \arg \max_{X(t_i, l)(1 \le i \le n)} \Pr_{\text{uc-sky}}(X(t_i, l)). \quad (6)$$

According to formula (3)-(6), a basic dynamic programming algorithm called BasicDP is proposed. The pseudo-code of BasicDP is shown in Algorithm 1. BasicDP first sorts T in lexicographic order and constructs a 2-dimensional range tree $\mathcal{T}$ on T. Then, for all pairs of tuples t_i, t_j such that $t_j \in T_{i-1}$ and $t_j \not\prec t_i$, BasicDP precomputes and stores $\gamma(t_i, t_j)$ by performing a range query $(t_j[1], t_i[1]] \times [0, t_i[2]]$ on $\mathcal{T}$. After that, BasicDP keeps two arrays to record the states during the computation: $cost(t_i, k)$ for $\Pr_{\text{uc-sky}}(X(t_i, k))$, and $Y(t_i, k)$ for t_{j^*} defined in formula (5), i.e., tuple with the largest index in $X(t_i, k) \setminus \{t_i\}$. When $k = 1$, BasicDP initializes $Y(t_i, 1)$ as t_i and computes $cost(t_i, 1)$ by performing a range query $[0, t_i[1]] \times [0, t_i[2]]$ on $\mathcal{T}$ for each tuple $t_i \in T$. Note that since t_i also lies in the queried range, the returned value is multiplied by $(1 - p(t_i))^{-1}$ to exclude the impact of t_i. When $k \ge 2$, BasicDP iterates over T_{i-1} to find the ideal tuple t_{j^*} guided by formula (5), and then compute $Y(t_i, k)$ and $cost(t_i, k)$ according to formula (4). Finally, the UC-SKY probability of the optimal solution is given by the maximum $cost(t_i, l)$ over

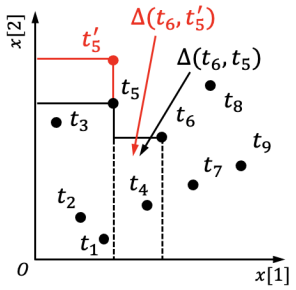


Fig. 2: Illustrating $\Delta(t_i, t_j)$

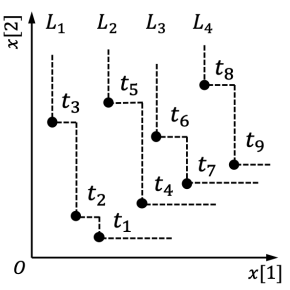


Fig. 3: Maximal layers

all $t_i \in T$, and the corresponding set is obtained by recursively traversing the array Y.

Complexity Analysis: Sorting T in lexicographical order and constructing the 2-dimensional range tree $\mathcal{T}$ on T consume $O(n \log n)$ time. Using a variant of fractional cascading [23], range queries on $\mathcal{T}$ can be performed in $O(\log n)$ time. Therefore, precomputing all $\gamma(t_i, t_j)$ and initializing $cost(t_i, 1)$ for each $t_i \in T$ takes $O(n^2 \log n)$ and $O(n \log n)$ time, respectively. For each $k \geq 2$, it takes linear time to find the ideal t_{j^*} by iterating over T_{i-1} for each $t_i \in T$, while other operations take constant time. Hence, the time complexity of BASICDP is $O(n \log n + n^2 \log n + ln^2) = O(\max\{l, \log n\} n^2)$. The space consumption of the 2-dimensional range tree $\mathcal{T}$ is $O(n)$. And storing $\gamma(t_i, t_j)$ for all pairs of tuples t_i, t_j such that $t_j \not\prec t_i$ and $t_j \in T_{i-1}$ takes $O(n^2)$ space. The two arrays $cost(t_i, k)$ and $Y(t_i, k)$ occupy $O(ln)$ space in total. Thus, the space complexity of BASICDP is $O(n^2 + ln) = O(n^2)$.

B. Grouping and Pruning Optimizations

In this part, we further improve BASICDP based on two critical observations. First, when precomputing $\gamma(t_i, t_j)$ for each tuple $t_i \in T$, range queries performed in line 5 of Algorithm 1 are identical for tuples with the same $t_j[1]$. Lemma 3 formally declares this observation.

Lemma 3: For each tuple $t_i \in T$, let t_{j_1}, t_{j_2} be two arbitrary tuples in T_{i-1} such that $t_{j_1} \neq t_{j_2}$ and $t_{j_1}, t_{j_2} \not\prec t_i$. If $t_{j_1}[1] = t_{j_2}[1]$, then $\gamma(t_i, t_{j_1}) = \gamma(t_i, t_{j_2})$.

Proof: Since all tuples t in $\Delta(t_i, t_j) = \{t \in T_i \mid t \prec t_i \wedge t \not\prec t_j\}$ satisfy that $t_j[1] < t[1] \leq t_i[1]$, $0 \leq t[2] \leq t_i[2]$, and $t \neq t_i$, we have $\Delta(t_i, t_{j_1}) = \Delta(t_i, t_{j_2})$. Thus, $\gamma(t_i, t_{j_1}) = \gamma(t_i, t_{j_2})$. ■

Let $X = \{x_1, \cdots, x_m\}$ be the projection of tuples in T on $t[1]$-axis sorted in ascending order. Partition T into m groups $T_{x_1}, \cdots, T_{x_m}$, where $T_{x_i} = \{t \in T \mid t[1] = x_i\}$. Recall that when T is sorted in lexicographical order, for each $1 \leq i \leq n$, t_i denotes the i-th tuple in T and T_i is the set of the first i tuples in T. Suppose $t_i \in T_{x_q}$, i.e., $t_i[1] = x_q$. Every $t_j \in T_{i-1}$ such that $t_j \not\prec t_i$ belongs to $\bigcup_{x_p \in X : x_p < x_q} T_{x_p}$. For each group T_{x_p} with $x_p < x_q$, let $\gamma(t_i, x_p) = \gamma(t_i, t' = (x_p, t_i[2]))$, where t' is a newly created tuple. From Lemma 3, it is concluded that for each tuple $t_j \in T_{x_p}$ such that $t_j \not\prec t_i$, $\gamma(t_i, t_j) = \gamma(t_i, x_p)$. For example, consider tuples t_5 and t'_5 in Fig. 2, since $t_5[1] = t'_5[1]$, we have $\gamma(t_6, t_5) = \gamma(t_6, t'_5) = \gamma(t_6, t_5[1])$. Accordingly, let

$$t_{x_p, j^*} = \arg \max_{t_j \in T_{x_p} : t_j \not\prec t_i} \Pr_{\text{uc-sky}}(X(t_j, k-1)), \quad (7)$$

formula (5) can be rewritten as

$$t_{j^*} = \arg \max_{t_{x_p, j^*} (x_p < x_q)} \gamma(t_i, x_p) \times \Pr_{\text{uc-sky}}(X(t_{x_p, j^*}, k-1)). \quad (8)$$

Formula (8) indicates that if T is partitioned into groups, only tuples with the largest $\Pr_{\text{uc-sky}}(X(t_j, k-1))$ in each group need to be considered when looking for the ideal t_{j^*}.

Second, let S^* be the optimal solution, formula (6) implies that $\Pr_{\text{uc-sky}}(S^*) \geq cost(t_i, l)$ for each $t_i \in T$. Thus, if array $cost$ is filled horizontally, i.e., the next tuple will be processed only after $cost(t, 1), \cdots, cost(t, l)$ of the currently processed tuple t are all computed, instead of vertically as in BASICDP, a lower bound lb on $\Pr_{\text{uc-sky}}(S^*)$ can be obtained very early by maintaining the maximum value of $cost(t_i, l)$ computed so far. Utilizing lb, computations of some tuples can be pruned according to the following lemma.

Algorithm 2: Group DP Algorithm with Pruning

Input: uncertain dataset $\mathcal{D} = (T, p)$, integer l
Output: UC-SKY$(\mathcal{D}, l)$

```
1  X = {x_1, ..., x_m} ← projection of T on t[1]-axis;
2  Partition T into T_{x_1}, ..., T_{x_m};
3  foreach x_q ∈ X do
4      Sort T_{x_q} in descending order based on t[2];
5  Construct a 2-dimensional range-tree T on T;
6  lb ← 0;
7  foreach x_q ∈ X do
8      foreach t ∈ T_{x_q} do
9          Compute cost(t,1) as Algorithm 1 does;
10         if cost(t,1) < lb then T_{x_q} ← T_{x_q} \ {t};
11         else
12             foreach x_p ∈ X : x_p < x_q ∧ |T_{x_p}| > 0 do
13                 γ(t, x_p) ← (1 − p(t))^{-1} ×
                     RangeQuery(T, (x_p, t[1]], [0, t[2]]);
14                 j_{x_p} ← LowerBound(T_{x_p}, t[2]);
15                 foreach k ← 2 to l do
16                     t_{x_p,j*} ← A_{x_p}[k − 1][j_{x_p}];
17                     if p(t) × γ(t, x_p) × cost(t_{x_p,j*}, k −
                         1) > cost(t,k) then
18                         Y(t,k) ← t_{x_p,j*};
19                         cost(t,k) ← p(t) × γ(t, x_p) ×
                             cost(t_{x_p,j*}, k − 1);
20             Assume t is the i-th tuple in T_{x_q};
21             foreach k ← 1 to l do
22                 if cost(t,k) > cost(A_{x_q}[k][i − 1], k)
                     then A_{x_q}[k][i] ← t ;
23                 else A_{x_q}[k][i] ← A_{x_q}[k][i − 1];
24             lb ← max{lb, cost(t,l)};
25 Collect UC-SKY(D, l) as Algorithm 1 does;
26 return UC-SKY(D, l);
```

Lemma 4: Given an uncertain dataset $\mathcal{D} = (T, p)$, an integer l, and a lower bound lb, a tuple t_i can be safely discarded if $cost(t_i, 1) = p(t_i) \times \prod_{t_j \in T_i : t_j \prec t_i} (1 - p(t_j)) < lb$.

Proof: Let S be a solution containing t_i. Since the UC-SKY probability is anti-monotonicity, $\Pr_{\text{uc-sky}}(S) \leq \Pr_{\text{uc-sky}}(t_i) = p(t_i) \times \prod_{t_j \in T_i : t_j \prec t_i} (1 - p(t_j))$. Thus, we have $\Pr_{\text{uc-sky}}(S) < lb$. Hence, S is not the optimal solution. ■

With these two observations, an improved algorithm called GROUPDP is proposed, whose pseudo-code is shown in Algorithm 2. GROUPDP first partitions T into groups $T_{x_1}, \cdots, T_{x_m}$ and sorts tuples in each group in descending order based on $t[2]$. Then, GROUPDP constructs a 2-dimensional range tree $\mathcal{T}$ on T and initializes the lower bound lb to zero. After that, GROUPDP starts to fill arrays $cost$ and Y horizontally. In spe-

cific, for each tuple t in each group T_{x_q}, GROUPDP initializes $cost(t,1)$ and $Y(t,1)$ as BASICDP does. If $cost(t,1) < lb$, then t is removed from T_{x_q} based on Lemma 4. Otherwise, GROUPDP iterates over non-empty groups T_{x_p} with $x_p < x_q$ to update $cost(t,2), \cdots, cost(t,l)$ according to formula (8). Concretely, for each such T_{x_p}, GROUPDP computes $\gamma(t, x_p)$ by performing a range query on $\mathcal{T}$, and locates the index j_{x_p} of the last tuple t' in T_{x_p} such that $t'[2] > t[2]$, i.e.,

$$j_{x_p} = \arg \max_{1 \le j \le |T_{x_p}|} T_{x_p}[j][2] > t[2] \wedge T_{x_p}[j+1][2] \le t[2],$$

with a binary search on T_{x_p}. Since all tuples $t' \in T_{x_p}$ have $t'[1] = x_p < x_q = t[1]$, the indexes of all tuples in T_{x_p} that do not dominate t are not greater than j_{x_p}. Using j_{x_p}, the tuple t_{x_p,j^*} in group T_{x_p} used to update $cost(t,k)$ is retrieved from array $A_{x_p}[k-1]$ for each $2 \le k \le l$, where for $1 \le i \le |T_{x_p}|$,

$$A_{x_p}[k][i] = \arg \max_{t \in \{T_{x_p}[1], \cdots, T_{x_p}[i]\}} \Pr_{\text{uc-sky}}(X(t,k)).$$

When $cost(t,2), \cdots, cost(t,l)$ are computed, say t is the i-th tuple in T_{x_q}, GROUPDP sets $A_{x_q}[k][i]$ according to $A_{x_q}[k][i-1]$ $(1 \le k \le l)$, and updates lb to $\max\{lb, cost(t,l)\}$. Finally, GROUPDP collects the result as BASICDP does.

Complexity Analysis: Partitioning T into groups, sorting all groups, and constructing the range tree all take $O(n \log n)$ time. For each tuple t, a range query is performed to compute $cost(t,1)$, which takes $O(n \log n)$ time in total. If a tuple t is not pruned, say $t \in T_{x_q}$, for each non-empty groups T_{x_p} with $x_p < x_q$, performing a range query on $\mathcal{T}$ to compute $\gamma(t, x_p)$ and performing a binary search on T_{x_p} to locate j_{x_p} both take $O(\log n)$ time. Then, for each $2 \le k \le l$, it takes constant time to retrieve t_{x_p,j^*} from $A_{x_p}[k]$ and update $cost(t,k)$. This uses a total of $O(m(\log n + l))$ time. After processing t, updating each array $A_{x_q}[k]$ $(1 \le k \le l)$ takes constant time. Therefore, the time complexity of GROUPDP is $O(n \log n + n'm(\log n + l))$, where n' is the number of tuples not discarded.

The horizontal filling of *cost* avoids the space consumption for pre-recording all $\gamma(t, x_p)$. Now, they can be computed on demand. And arrays $A_{x_1}, \cdots, A_{x_m}$ use a total of $O(nl)$ space. Thus, the space complexity of GROUPDP reduces to $O(nl)$.

V. ALGORITHMS IN MD

The UC-SKY query becomes more challenging in MD due to its NP-hardness. To solve the UC-SKY query efficiently, we propose an exact algorithm with effective pruning strategies. As byproducts, two approximation algorithms are also obtained, which achieve better time efficiency at the expense of a certain degree of solution quality.

The proposed exact algorithm is called PDBB. It runs in three consecutive stages, namely the preprocessing stage, the dividing stage, and the branch-and-bound search stage. In the preprocessing stage, PDBB reduces the input dataset to a smaller one based on proposed data reduction rules. In the dividing stage, PDBB uses a non-trivial dividing strategy to divide the query on the reduced dataset into multiple constrained subqueries on much smaller datasets. It is ensured that the optimal solution is included in the solutions of these subqueries. In the branch-and-bound search stage, PDBB solves each subquery by performing a branch-and-bound search to find a constrained optimal solution, and returns the one with the largest UC-SKY probability as the final result.

A. Preprocessing Stage

For ease of notation, for each tuple $t \in T$, let $D_{\prec}(t) = \{s \in T \mid s \prec t\}$ and $D_{\succ}(t) = \{s \in T \mid s \succ t\}$ hereafter. We call a set S consisting of l mutually non-dominated tuples a candidate set or a valid solution. A tuple t cannot participate in any valid solution if there are more than $n-l+1$ tuples in $D(t) = D_{\prec}(t) \cup D_{\succ}(t)$. Thus, all such tuples will be discarded as stated in Lemma 5.

Lemma 5: Given an uncertain dataset $\mathcal{D} = (T, p)$ and an integer l, a tuple t can be safely discarded if $|D(t)| > n-l+1$.

However, it is observed that when the input size l is small, the effectiveness of this rule is quite low. Thus, a lower bound lb on the UC-SKY probability of the optimal solution is also used to further reduce the dataset. As stated by Lemma 6, a tuple t will be discarded if the upper bound on the UC-SKY probability of the best solution including t is less than lb. It is worth noting that, unlike Lemma 4, Lemma 6 derives a tighter upper bound based on dominance relationships between tuples.

Lemma 6: Given an uncertain dataset $\mathcal{D} = (T, p)$, an integer l, and a lower bound lb, a tuple t can be safely discarded if $\beta(t) = p(t) \times \prod_{s \in D_{\prec}(t)}(1-p(s)) \times \prod_{s \in \Delta(t)} p(s) < lb$, where $\Delta(t)$ is the set consisting of $l-1$ tuples in $T \setminus (D(t) \cup \{t\})$ with highest probabilities.

Proof: Let S be a solution containing t. According to formula (2), $\Pr_{\text{uc-sky}}(S) = \prod_{t \in S} p(t) \times \prod_{s \in \cup_{t \in S} D_{\prec}(t)}(1 - p(s))$. Since tuples in S are mutually non-dominated, we have $\prod_{t \in S} p(t) \le p(t) \times \prod_{s \in \Delta(t)} p(s)$. Moreover, since $D_{\prec}(t) \subseteq \cup_{t \in S} D_{\prec}(t)$, $\prod_{s \in \cup_{t \in S} D_{\prec}(t)}(1-p(s)) \le \prod_{s \in D_{\prec}(t)}(1-p(s))$. Hence, $\Pr_{\text{uc-sky}}(S) < lb$, i.e., S is not the optimal solution. ■

1) Methods for lb computation: Two methods are proposed for computing the lower bound lb. The first, called SUBSKY, is a fast heuristic, while the second, called GREEDY, typically produces a larger lower bound at the cost of higher computational time. To be specific, SUBSKY first computes the skyline of the tuples in T whose attendance probability exceeds 1/2. If the resulting skyline contains fewer than l tuples, SUBSKY sets lb to zero. Otherwise, SUBSKY sets lb to the UC-SKY probability of the candidate set, which consists of the l tuples with the highest probabilities from the resulting skyline. The pseudo-code for SUBSKY is provided in Algorithm 3, and Lemma 7 proves the approximation ratio of SUBSKY.

Lemma 7: Given an uncertain dataset $\mathcal{D} = (T, p)$ and an integer l, let $S^* = \text{UC-SKY}(\mathcal{D}, l)$ and lb be the lower bound returned by Algorithm 3, if $lb > 0$, then $lb \ge 2^{-n} \times \Pr_{\text{uc-sky}}(S^*)$, where n is the number of tuples in T.

Proof: According to Algorithm 3 and formula (2), if $lb > 0$, we have $lb = \Pr_{\text{uc-sky}}(S) = \prod_{t \in S} p(t) \times \prod_{s \in \cup_{t \in S} D_{\prec}(t)}(1 - p(s)) > 1/2^{|S|} \times 1/2^{|\cup_{t \in S} D_{\prec}(t)|}$ as $\forall t \in T', p(t) > 1/2$ and $\forall t \in T \setminus T', 1-p(t) > 1/2$. Since $\Pr_{\text{uc-sky}}(S^*) \le 1$ and $|S| + |\cup_{t \in S} D_{\prec}(t)| \le n$, it is derived that $lb \ge 2^{-n} \times \Pr_{\text{uc-sky}}(S^*)$. ■

GREEDY computes lb in a greedy manner. However, it is observed that just greedily selecting the tuple t that achieves the

Algorithm 3: Sub-Skyline Algorithm for lb

Input: uncertain dataset $\mathcal{D} = (T, p)$, integer l
Output: lb

$T' \leftarrow \{t \in T \mid p(t) > 1/2\}$;
Compute $\mathrm{SKY}(T')$;
if $|\mathrm{SKY}(T')| \geq l$ **then**
  $S \leftarrow$ collect tuples in $\mathrm{SKY}(T')$ with top-l probabilities;
  $lb \leftarrow \Pr_{\text{uc-sky}}(S)$;
else $lb \leftarrow 0$;
return lb;

largest marginal gain $\log \Pr_{\text{uc-sky}}(S \cup \{t\}) - \log \Pr_{\text{uc-sky}}(S)$ may fail to find a valid solution. On the other hand, greedily selecting the tuple t with the fewest tuples in $D(t)$, in an attempt to ensure a valid solution with high probability, can lead to poor solution quality. To address this trade-off, GREEDY introduces a parameter α to balance these two factors. Specifically, GREEDY varies α from 0 to 1 in steps of ε, and at each step greedily selects the tuple t that maximizes the function $f(t) = (1-\alpha) \times \log p(t) \prod_{s \in D_{\prec}(t)}(1-p(s)) - \alpha \times |D(t)|$. To efficiently identify the tuple that maximizes $f(t)$, GREEDY uses a modifiable max-heap H sorted by $f(t)$. The heap H consists of two arrays. The first array serves as a normal heap and the second array maps the identifier of a tuple to its position in the first array. Since no tuples share the same identifier, H supports efficient updates to $f(t)$ for any tuple t in the heap. When there are fewer than l tuples in S and H is not empty, the top tuple t in H is selected and added to S. Then, all tuples in $D(t)$ are removed from H since they cannot be added to S anymore. Due to the removal of tuples in $D(t)$, $f(w)$ of some tuples w are also updated accordingly as shown in line 11-16 of Algorithm 4.

Algorithm 5 outlines the details of the preprocessing stage. Initially, SUBSKY is invoked to compute lb, and a coarse data reduction is performed on the dataset according to Lemma 4 Then, after updating lb with GREEDY, the dataset undergoes further fine-grained reduction based on Lemma 5 and 6. A queue Q is initialized to record tuples that can be discarded. Each tuple t that satisfies the conditions in these two lemmas is pushed into Q. While Q is not empty, for each t popped from Q, t is removed from T. After removing t, since both $|D(s)|$ and $|T|$ decrease by one for each $s \in D(t)$, the relation between $|D(s)|$ and $|T| - l + 1$ does not change. Similarly, $\beta(s)$ for each $s \in D(t)$ also remains unchanged. Thus, only tuples in $T \setminus N(t)$ are checked for collecting tuples that newly meet the conditions.

Complexity Analysis: In Algorithm 3, the computation of $\mathrm{SKY}(T')$ and the identification of S require at most $O(n^2)$ time. Computing $\Pr_{\text{uc-sky}}(t)$ for all tuples $t \in T$ also takes at most $O(n^2)$ time. Hence, the coarse data reduction takes at most $O(n^2)$ time. Suppose $D_{\prec}(t)$ and $D_{\succ}(t)$ are precomputed for each tuple $t \in T$ in $O(n^2)$ time. In Algorithm 4, when a tuple t is added to S, tuples in $D(t)$ are removed from H and $f(w)$ of every $w \in D(s)$ for each $s \in D(t)$ are updated. Since

Algorithm 4: Greedy Algorithm for lb

Input: uncertain dataset $\mathcal{D} = (T, p)$, integer l, step ε
Output: lb

1 $lb \leftarrow 0$; $\alpha \leftarrow 0$;
2 **while** $lb = 0$ **and** $\alpha \leq 1$ **do**
3 $S \leftarrow \emptyset$;
4 Initialize a modifiable max-heap H;
5 **foreach** $t \in T$ **do**
6 $f(t) \leftarrow (1-\alpha) \times \log p(t) \prod_{s \in D_{\prec}(t)}(1-p(s)) - \alpha \times |D(t)|$;
7 $H.push(\{f(t), t\})$
8 **while** $|S| < l$ **and** $H \neq \emptyset$ **do**
9 $t \leftarrow H.top()$; $S \leftarrow S \cup \{t\}$; $H.pop()$;
10 Remove tuples in $D(t)$ from H;
11 **foreach** $s \in D(t)$ **do**
12 **foreach** $w \in D(s)$ **s.t.** $w \in H$ **do**
13 Add α to $f(w)$ in H;
14 **foreach** $s \in D_{\prec}(t)$ **do**
15 **foreach** $w \in D_{\succ}(s)$ **s.t.** $w \in H$ **do**
16 Reduce $f(w)$ in H by $(1-\alpha) \times \log(1-p(s))$;
17 **if** $|S| < l$ **then** $lb \leftarrow 0$;
18 **else** $lb \leftarrow \Pr_{\text{uc-sky}}(S)$;
19 $\alpha \leftarrow \alpha + \varepsilon$;
20 **return** lb;

$f(w)$ of at most n tuples w will be affected when a tuple $s \in D(t)$ is removed, up to $O(n^2)$ updates will occur in H at each step. Then, with the fact that the modifiable heap H can update $f(t)$ of a tuple t and then readjust itself in $O(\log n)$ time, the time complexity of Algorithms 4 is $O(1/\varepsilon n^2 \log n))$. After that, it takes $O(n^2 \log n)$ time to sort tuples in $T \setminus (D(t) \cup \{t\})$ in descending order of their probabilities for each tuple $t \in T$. With this information, iteratively collecting tuples satisfying the conditions in Lemma 5 and 6 into Q needs at most $O(n^2)$ time. Therefore, the overall time complexity of Algorithm 5 is $O(1/\varepsilon n^2 \log n)$.

B. Dividing Stage

In this stage, the UC-SKY query being processed is further divided based on the following observation. That is the tuple set T can be partitioned into *maximal layers*, where the first layer L_1 is the skyline of T, and for each $i > 1$, the i-th layer L_i is the skyline of $T \setminus \bigcup_{j=1}^{i-1} L_j$. By definition, tuples in each layer L_i are mutually non-dominated. Suppose there are τ maximal layers in T, any candidate set S consists of τ subsets $S_1 \subseteq L_1, \cdots, S_\tau \subseteq L_\tau$ such that $\sum_{i=1}^{\tau} |S_i| = l$. Note that some of them maybe empty. Set S_i with the smallest index among non-empty ones is called the pivot set of S. For example, the tuple set T shown in Fig. 3 is partitioned into four maximal layers L_1, L_2, L_3, and L_4. And when $l = 3$, $S = \{t_5, t_6, t_7\}$ consists of two non-empty subsets $S_2 = \{t_5\} \subseteq L_2$ and $S_3 = \{t_6, t_7\} \subseteq L_3$, where S_2 is the pivot set of S.

Algorithm 5: Preprocessing Stage

Input: uncertain dataset $\mathcal{D} = (T, p)$, integer l, step ε
Output: reduced dataset $\mathcal{D} = (T, p)$

1 $lb \leftarrow \text{SKY}(\mathcal{D}, l)$;
2 **foreach** $t \in T$ **do**
3 **if** $\Pr_{\text{uc-sky}}(t) < lb$ **then** Remove t from T ;
4 $lb \leftarrow \max\{lb, \text{GREEDY}(\mathcal{D}, l, \varepsilon)\}$;
5 Initialize a queue Q;
6 **foreach** $t \in T$ **do**
7 **if** $|D(t)| > |T| - l + 1$ **or** $\beta(t) < lb$ **then**
8 $Q.push(t)$;
9 **while** $Q \neq \emptyset$ **do**
10 $t \leftarrow Q.pop()$;
11 Remove t from T;
12 **foreach** $s \in T \setminus D(t)$ **do**
13 **if** $|D(s)| > |T| - l + 1$ **then** $Q.push(s)$;
14 **else**
15 Update $\Delta(s)$;
16 **if** $\beta(s) < lb$ **then** $Q.push(s)$;

Accordingly, the UC-SKY query on $\mathcal{D}$ is divided into subqueries by enumerating pivot sets S_p of all candidate sets. Given a pivot set S_p, the subquery aims to find the candidate set S with the largest UC-SKY probability among all candidate sets which have S_p as the pivot set. Algorithm 6 presents the formal procedure of the dividing stage. Suppose the algorithm now reaches the i-th layer, i.e., $L = L_i$. When an empty set is enumerated, it recursively calls itself at L_{i+1} (line 4-7). Otherwise, it solves the subquery exactly by calling the search algorithm introduced later (line 9-15). The optimal solution S^* is returned as the one with the largest UC-SKY probability among all candidate sets returned by formed subqueries. This is because let S_p^* be the pivot set of S^*, S^* will be obtained via solving the subquery given S_p^*. Note that the pruning strategy is also used in Algorithm 6. According to Lemma 8 and the anti-monotonicity of the UC-SKY probability, the enumeration will stop immediately once the product of the top j highest probabilities of tuples in L is less than the lower bound lb.

Lemma 8: Let $t_1, \cdots, t_j$ be tuples in L with top j highest probabilities and lb be the lower bound. If $\prod_{i=1}^{j} p(t_i) < lb$, then for any set $S \subseteq L$ with $|S| \geq j$, $\Pr_{\text{uc-sky}}(S) < lb$.

Proof: According to formula (2), we know $\Pr_{\text{uc-sky}}(S) \leq \prod_{t \in S} p(t) \leq \prod_{i=1}^{|S|} p(t_i)$. Moreover, since $|S| \geq j$, we have $\prod_{i=1}^{|S|} p(t_i) \leq \prod_{i=1}^{j} p(t_i) < lb$. ∎

C. Branch-and-Bound Search Stage

Suppose that the dividing stage reaches the maximal layer L_i. When a non-empty subset $S_q \subseteq L_i$ of size j is enumerated, BBSEARCH is invoked to find the candidate set S with the largest UC-SKY probability under the condition that S_p is the pivot set of S. The pseudo-code of BBSEARCH is presented in Algorithm 7. BBSEARCH adopts the *branch-and-bound* framework, and a branch B in BBSEARCH is represented as a triple $(\mathcal{D} = (T, p), S, k)$. Specifically,

Algorithm 6: Dividing Stage
DIVIDE$(\mathcal{D} = (T, p), l, lb)$

Input: uncertain dataset $\mathcal{D} = (T, p)$, integer l, lower bound lb
Output: UC-SKY$(\mathcal{D}, l)$

1 $L \leftarrow \text{SKY}(T)$; $T \leftarrow T \setminus L$; $p \leftarrow 1$;
2 Sort $L \leftarrow \{t_1, \cdots, t_{|L|}\}$ in descending order of $p(t)$;
3 **foreach** $j \leftarrow 0$ **to** $\min\{|L|, l\}$ **do**
4 **if** $j = 0$ **and** $|T| \geq l$ **then**
5 $S \leftarrow \text{DIVIDE}(\mathcal{D} = (T, p), l, lb)$;
6 **if** $\Pr_{\text{uc-sky}}(S) > lb$ **then**
7 $S^* \leftarrow S$; $lb \leftarrow \Pr_{\text{uc-sky}}(S)$;
8 **else**
9 $p \leftarrow p \times p(t_j)$;
10 **if** $p < lb$ **then**
11 **foreach** $S_p \subseteq L$ **s.t.** $|S_p| = j$ **do**
12 $S \leftarrow \text{BBSEARCH}(\mathcal{D} = (T \setminus \bigcup_{t \in S_p} D_{\succ}(t), p), S_p, l - j, lb)$;
13 **if** $\Pr_{\text{uc-sky}}(S) > lb$ **then**
14 $S^* \leftarrow S$; $lb \leftarrow \Pr_{\text{uc-sky}}(S)$;
15 **else break** ;
16 **return** S^*;

- S is a candidate set satisfying that $|S| \leq l$ and S_p is the pivot set of S,
- T contains tuples that can be used to further expend S,
- k is the number of tuples that should be further added to S, i.e., $|S| + k = l$.

Given a pivot set $S_p \subseteq L_i$, BBSEARCH starts from the branch $(\mathcal{D} = (T = \bigcup_{p=i+1}^{\tau} L_p \setminus \bigcup_{t \in S_p} D_{\succ}(t), p), S = S_p, k = l - j)$. To facilitate subsequent computations, for each tuple $t \in T$, a temporary set $D'_{\prec}(t)$ is initialized to record tuples in $D_{\prec}(t)$ that not dominate any tuple in S_p. Moreover, t is associated with the marginal gain $r(t)$ of adding t to S. Formally, $r(t) = \Pr_{\text{uc-sky}}(S \cup \{t\}) / \Pr_{\text{uc-sky}}(S) = p(t) \times \prod_{s \in D'_{\prec}(t)} (1 - p(s))$. Let $\{t_1, \cdots, t_{|T|}\}$ be tuples in T sorted in descending order of $r(t)$. BBSEARCH iteratively creates $|T|$ sub-branches, where the i-th branch expends S with t_i, and excludes $t_1, \cdots, t_{i-1}$ from S. Specifically, in each iteration, the current best tuple t is removed from T, and the new candidate set S' is set to $S \cup \{t\}$. The set T' including tuples for further expending S' is generated by removing tuples in $D_{\succ}(t)$ from T. Also, the marginal gains of tuples in T' are updated by eliminating the impact of each tuple $s \in D'_{\prec}(t)$ on each tuple $w \in D_{\succ}(s)$.

During the recursive search process, some pruning techniques can also be applied. Let $B = (\mathcal{D} = (T, P), S, k)$ denote a branch, and lb be the lower bound on the UC-SKY probability of the optima. First, branch B can be pruned if there are fewer than k tuples in T for further expending S. Second, branch B can be pruned if $\Pr_{\text{uc-sky}}(S) < lb$ since $\Pr_{\text{uc-sky}}(S)$ is an upper bound on the UC-SKY probability of any superset of S.

Algorithm 7: Branch-and-Bound Search Stage BBSEARCH($\mathcal{D} = (T, p), S_p, k, lb$)

Input: uncertain dataset $\mathcal{D} = (T, p)$, pivot set S_p, integer k, lower bound lb
Output: optimal solution S with S_p as the pivot set

foreach $t \in T$ **do**
  $D'_{\prec}(t) \leftarrow D_{\prec}(t) \setminus \bigcup_{w \in S_p} D_{\prec}(w)$;
  $r(t) \leftarrow p(t) \times \prod_{s \in D'_{\prec}(t)} (1 - p(s))$;
  Replace t with $(t, r(t))$;
BBS-RES($\mathcal{D}, S_p, k, lb$);
Procedure BBS-RES($\mathcal{D} = (T, p), S, k, lb$)
  if $k = 0$ **and** $\Pr_{\text{uc-sky}}(S) > lb$ **then**
    $lb \leftarrow \Pr_{\text{uc-sky}}(S)$; **return** S;
  if $|T| < k$ **or** $\Pr_{\text{uc-sky}}(S) < lb$ **then return**;
  $t_1, \cdots, t_k \leftarrow k$-tuples in T with top-k highest $p(t)$;
  if $\Pr_{\text{uc-sky}}(S) \times \prod_{i=1}^{k} p(t_i) < lb$ **then return**;
  while $|T| \geq k$ **do**
    $t \leftarrow \arg\max_{t \in T} r(t)$;
    $T \leftarrow T \setminus \{t\}$;
    $S' \leftarrow S \cup \{t\}$;
    $\Pr_{\text{uc-sky}}(S') \leftarrow \Pr_{\text{uc-sky}}(S) \times r(t)$;
    $T' \leftarrow T \setminus D_{\succ}(t)$;
    foreach $s \in D'_{\prec}(t)$ **do**
      foreach $w \in D_{\succ}(s)$ **s.t.** $w \in T'$ **do**
        $r(w) \leftarrow r(w)/(1 - p(s))$;
    BBSEARCH($\mathcal{D} = (T', p), S', k - 1, lb$);

Third, let $t_1, \cdots, t_k$ be the k tuples in T with the top-k highest $p(t)$, branch B can be pruned if $\Pr_{\text{uc-sky}}(S) \times \prod_{i=1}^{k} p(t_i) < lb$, which also estimates the upper bound on the UC-SKY probability of any superset of S.

VI. The UC-RSKY Query

Personalization is a rigid requirement for decision support. To extend the skyline query for serving the specific preferences of an individual user, restricted skyline (rskyline for short) query is recently proposed in [24]. In this section, we analogously extend the UC-SKY query.

Given a scoring function $f : \mathbb{R}^d \to \mathbb{R}^+$, the value $f(t[1], \cdots, t[d])$ is called the score of tuple t, also written as $f(t)$. The function f is said to be *monotone* if for any tuple t, s it holds that $f(t) \leq f(s)$ if $\forall 1 \leq i \leq d : t[i] \leq s[i]$. Given a set of monotone scoring functions $\mathcal{F}$, a tuple t is said to *$\mathcal{F}$-dominate* another tuple s, denoted by $t \prec_{\mathcal{F}} s$, if $\forall f \in \mathcal{F} : f(t) \leq f(s)$ [24]. Given a d-dimensional dataset D,the rskyline of D with respect to $\mathcal{F}$, denoted by $\text{RSKY}(D, \mathcal{F})$, is the subset of tuples that are not $\mathcal{F}$-dominated by any other tuples in D, i.e., $\text{RSKY}(D, \mathcal{F}) = \{t \in D \mid \forall s \in D : s \not\prec_{\mathcal{F}} t\}$ [24].

Definition 4 (UC-RSKY Probability): Given an uncertain dataset $\mathcal{D} = (T, p)$ and a set of monotone scoring functions $\mathcal{F}$, the U-RSkyline containment (UC-RSKY) probability of a set of mutually non-$\mathcal{F}$-dominated tuples $S \subseteq T$ is defined as the accumulated existence probabilities of possible worlds that have S in their rskyline with respect to $\mathcal{F}$, i.e.,

$$\Pr_{\text{uc-rsky}}(S, \mathcal{F}) = \sum_{D \sqsubseteq \mathcal{D}} \Pr(D) \times \mathbf{1}[S \subseteq \text{RSKY}(D, \mathcal{F})], \quad (9)$$

where $\mathbf{1}[\cdot]$ is the indicator function.

Definition 5 (UC-RSKY Query): Given an uncertain dataset $\mathcal{D} = (T, p)$, a set of monotone scoring functions $\mathcal{F}$, and an integer l, the U-RSkyline containment (UC-RSKY) query aims to find a set of mutually non-$\mathcal{F}$-dominant tuples S^* such that $|S^*| = l$ and $\Pr_{\text{uc-rsky}}(S^*, \mathcal{F})$ is maximized, i.e.,

$$\text{UC-RSKY}(\mathcal{D}, \mathcal{F}, l) = S^* = \arg \max_{S \subseteq T : |S| = l} \Pr_{\text{uc-rsky}}(S, \mathcal{F}).$$

Theorem 2: The UC-RSKY query is NP-hard when $d = 2$.

Proof: We prove this theorem by establishing a polynomial-time reduction from the UC-SKY query to the UC-RSKY query with $d = 2$. Given a d-dimensional uncertain dataset $\mathcal{D} = (T = \{t_1, \cdots, t_n\}, p)$ and an integer l, we construct a 2-dimensional uncertain dataset $\mathcal{D}' = (T', p')$, a set of monotone scoring functions $\mathcal{F} = \{f_1(\cdot), \cdots, f_d(\cdot)\}$, and an integer l' as follows. Let $T' = \{x_i = (i/(n+1), (n+1-i)/(n+1) \mid i \in \{1, \cdots, n\}\}$, and $p'(x_i) = p(t_i)$. Then, for each $1 \leq i \leq d$, a piece-wise function $f_i(\cdot)$ is inserted into $\mathcal{F}$, where

$$f_i(x) \begin{cases} t_j[i] & \exists 1 \leq j \leq n, x = x_j, \\ \min_{1 \leq j \leq n} t_j[i] & \text{else if } x[2] \leq 1 - x[1], \\ \max_{1 \leq j \leq n} t_j[i] & \text{otherwise.} \end{cases}$$

For any $1 \leq i \leq d$, $f_i(\cdot)$ is monotone since for any x_1 and x_2 if $x_1[1] \leq x_2[1]$ and $x_1[2] \leq x_2[2]$, then $f_i(x_1) \leq f_i(x_2)$. Finally, we set $l' = l$. Thus, we obtain an input instance to the UC-RSKY query when $d = 2$. It is noticed that for any two tuples $x_p, x_q \in T'$, $x_p \prec_{\mathcal{F}} x_q$ if and only if $f_i(x_p) = t_p[i] \leq f_i(x_q) = t_q[i]$ holds for all $1 \leq i \leq d$, i.e., $t_p \prec t_q$. Therefore, for each possible $D \sqsubseteq \mathcal{D}$, $S \subseteq \text{SKY}(D)$ if and only if $S' = \{x_i \mid t_i \in S\} \subseteq \text{RSKY}(D', \mathcal{F}) = \{x_i \mid t_i \in \text{SKY}(D)\}$, where $D' = \{x_i \mid t_i \in D\} \sqsubseteq \mathcal{D}'$. According to formula (1), it is concluded that $\Pr_{\text{uc-sky}}(S) = \Pr_{\text{uc-rsky}}(S', \mathcal{F})$ for any $S \subseteq T$ and $S' = \{x_i \mid t_i \in S\} \subseteq T'$. Thus, finding a l-tuple subset of T with the largest UC-SKY probability is equivalent to finding a l-tuple subset of T' with the largest UC-RSKY probability under $\mathcal{F}$. This completes the proof. ■

Based on Theorem 2, exact algorithms proposed in Section IV can only be used in some special cases. For example, as stated in [25], when $\mathcal{F}$ is the set of linear scoring functions subject to a set of linear constraints on weights, the rskyline query with $d = 2$ can be converted to the skyline query with $d = 2$. In such case, the UC-RSKY query with $d = 2$ can also be converted to the UC-SKY query with $d = 2$ using the same conversion, and thus can be exactly solved in polynomial time. As for algorithms proposed in Section V, all of them can be easily adapted for solving the UC-RSKY query.

VII. Experiments

A. Experimental Settings

Datasets: Both real and synthetic datasets are used in our experiments. IIP[1] is a 2D dataset containing 19,664 iceberg

[1]https://nsidc.org/data/g00807/versions/1

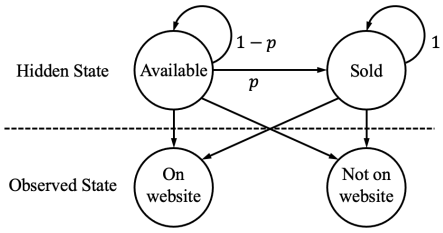


Fig. 4: HMM for generating vehicles' probabilities.

sighting records. Each record in IIP has a confidence level according to the source of sighting, including R/V (radar and visual), VIS (visual only), and RAD (radar only). Similar to [26], we generate the attendance probability of each record by converting these three confidence levels to probabilities 0.8, 0.7, and 0.6 respectively. CAR[2] is a 4D dataset with 323,433 tuples, each of which records the condition of a used car. Following [7], the available probability of each car is inferred with a hidden Markov model (HMM) shown in Fig. 4 based on the date when it is posted on the website. The transition graph assumes an independent selling probability for each car and once a car is sold, it shall not return to the available state. NBA[3] is an 8D dataset that contains 23,160 statistics for NBA players. The probability of each record is assigned using a uniform distribution in the interval [0, 1].

IND, ANTI, and CORR are generated with the same procedure described in [7]. These synthetic datasets represent typical distributions in multi-criteria data analysis and are generated using the benchmark data generator [1]. The probability of each tuple is generated uniformly in the interval $[\alpha-\delta, \alpha+\delta]$, where α is the specified probability center and δ represents one side width, i.e., $\delta = \min(\alpha, 1-\alpha)$. This guarantees that the generated probabilities are always between $[0, 1]$.

Algorithms. We implement the following algorithms in C++ and the source code is available at [27]

- HRT is the state-of-the-art algorithm proposed in [9].
- BASICDP is the 2D exact DP algorithm in Algorithm 1.
- GROUPDP is the 2D exact DP algorithm with grouping and pruning optimizations in Algorithm 2.
- SUBSKY is the approximation algorithm in Algorithm 3.
- GREEDY is the greedy algorithm in Algorithm 4. Step ε is set to 0.01 for preprocessing, and to 0.001 otherwise.
- PDBB is the MD exact algorithm, and its three stages are shown in Algorithm 5, 6, and 7, respectively.

All experiments are conducted on a machine with a 3.5-GHz Intel(R) Core(TM) i9-10920X CPU, 256GB main memory, and 1TB hard disk running CentOS 7.

B. 2D Experimental Results

Fig. 5 to 16 show the performance of the proposed algorithms on 2D synthetic datasets and IIP. The default values for data size n and probability center α of synthetic datasets, and output size l are set to $n = 10^4$, $\alpha = 0.5$, and $l = 6$. The running time limit (INF) is set as an hour. The colored lines show the running time and the bars show the approximation ratio of the two approximation algorithms. Note that some results are omitted since the corresponding algorithms cannot obtain any solution. For example, due to huge memory consumption, BASICDP cannot finish when $n \geq 10^5$ on synthetic datasets, and there are fewer than l tuples in the skyline found by SUBSKY when $n = 40\%$ and 60% on IIP.

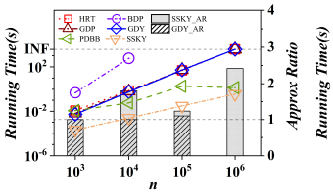


Fig. 5: IND, 2D: vary n

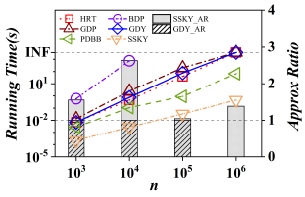


Fig. 6: ANTI, 2D: vary n

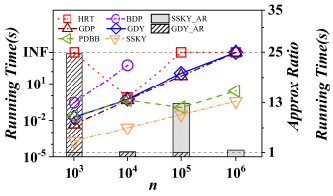


Fig. 7: CORR, 2D: vary n

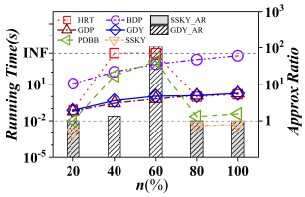


Fig. 8: IIP, 2D: vary n

1) The impact of data size n: We vary n from 10^3 to 10^6 for synthetic datasets and from 20% to 100% for IIP. Fig. 5 to 8 show the results for the four datasets. Overall, the running time of all algorithms increases as n grows. SUBSKY runs fastest due to its simplicity. However, sometimes it may find solutions of poor quality, or even fail to find feasible solutions. BASICDP is slowest since it requires quadratic time and does not involve any pruning strategy. GROUPDP consistently outperforms BASICDP, which demonstrates the effectiveness of the proposed optimization strategies. Generally, GROUPDP also outperforms HRT and is more stable. This is because tuple pruning and search are performed separately in HRT, and the performance of its search process is mainly determined by the data size after tuple pruning, while GROUPDP performs tuple pruning during the computation process. Although PDBB also performs tuple pruning and search separately, it remains relatively stable because alternative methods are used in its preprocessing stage to compute lb for tuple pruning (see the difference between PDBB and HRT when $n = 10^3$ on CORR). Note that PDBB outperforms HRT by around two orders of magnitude due to its more effective pruning and search strategies. The performance of PDBB is influenced by the solution quality of SUBSKY and GREEDY. Its running time increases only when both find poor solutions (see $n = 40\%$ and 60% on IIP). GREEDY performs similarly to GROUPDP. It can find a near-optimal solution in most cases, thus ensuring the efficiency of PDBB.

2) The impact of output size l: l is varied from 2 to 10, and the results are presented in Fig. 9 to 12. The running time of BASICDP, GROUPDP, GREEDY, and SUBSKY is barely impacted by changes in l since their time complexities are primarily determined by n. Note that the running time of GROUPDP on ANTI increases due to the fact that a large number of tuples are pruned, which causes the increasing value

[2]https://data.world/data-society/used-cars-data
[3]https://www.nba.com/stats/

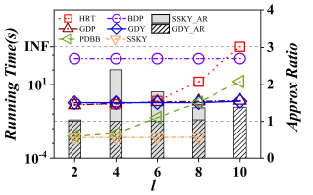


Fig. 9: IND, 2D: vary l

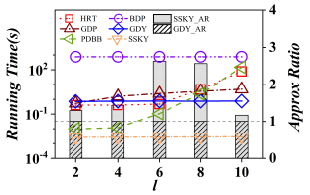


Fig. 10: ANTI, 2D: vary l

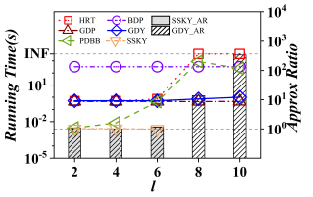


Fig. 11: CORR, 2D: vary l

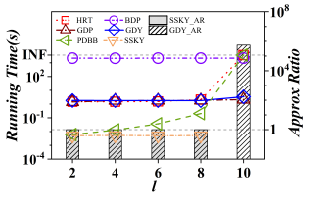


Fig. 12: IIP, 2D: vary l

of l to have an effect on its performance. In contrast, PDBB and HRT are more sensitive to changes in l, as both rely on tuple pruning and candidate set search. Consequently, their running time scales proportional with l due to the expanding search space, i.e., the larger possible candidate sets. Note that PDBB shows better scalability compared to HRT. HRT cannot finish within the limit time when l is large. Furthermore, as l increases, SUBSKY faces increasing difficulty in finding an approximate solution because the size of the skyline it constructs remains the same. This also makes it harder for GREEDY to find a solution with a good approximation ratio.

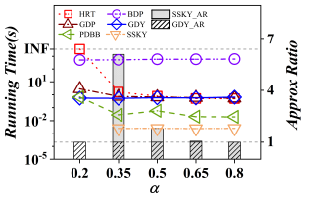


Fig. 13: IND, 2D: vary α

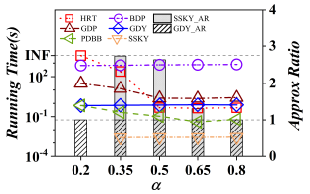


Fig. 14: ANTI, 2D: vary α

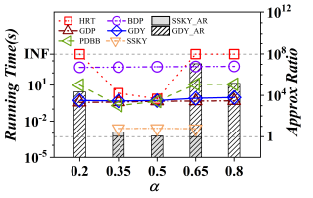


Fig. 15: CORR, 2D: vary α

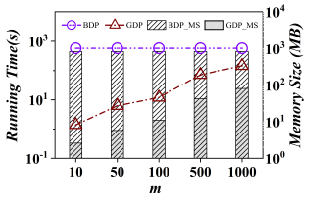


Fig. 16: IND, 2D: vary m

3) The impact of probability center α: We vary $\alpha \in \{0.2, 0.35, 0.5, 0.65, 0.8\}$ for the synthetic datasets. The results are shown in Fig. 13 to 15. Typically, the UC-SKY probability of the candidate set grows with the increase of α. For instance, on IND with $\alpha = 0.2$, $\Pr_{\text{uc-sky}}(S^*) = 0.516 \times 10^{-3}$, while on IND with $\alpha = 0.8$, and $\Pr_{\text{uc-sky}}(S^*) = 0.271$. Therefore, the larger α, the more pruning will be triggered in HRT, GROUPDP, and PDBB, causing their running time to decrease as α increases. In contrast, since BASICDP, GREEDY, and SUBSKY do not involve any pruning strategy, their running time remains stable with respect to α. Note that SUBSKY always fails in datasets with $\alpha = 0.2$ since there is no uncertain tuple with attendance probability higher than 0.5. The solution quality of GREEDY remains optimal on IND and ANTI, but fluctuates on CORR. This is because tuples exhibit the strongest dominant ability in CORR, thus limiting the selection space of GREEDY.

4) The effectiveness of GROUPDP: To further verify the effectiveness of the grouping strategy used in GROUPDP, we reorganize IND by dividing uncertain tuples into m groups and unifying the first coordinates of all tuples in each group, and disable the pruning strategy in GROUPDP. Fig. 16 shows the results with m varying from 10 to 10^3. Note that compared to BASICDP, GROUPDP always runs faster and uses less memory. While, the benefit of the grouping strategy decreases as m increases, which is consistent with the theoretical analysis.

C. MD Experimental Results

Fig. 21 to 35 show the experimental results of proposed algorithms on MD datasets. The default values for data size n, data dimension d, and probability center α of synthetic datasets, and output size l are set to $n = 10^4$, $d = 6$, $\alpha = 0.5$, and $l = 6$. The running time limit (INF) is set as an hour.

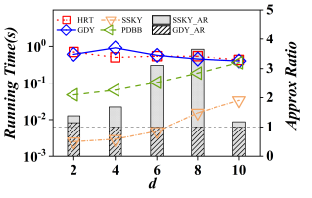


Fig. 17: IND, MD: vary d

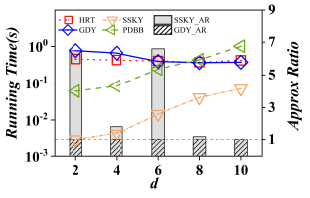


Fig. 18: ANTI, MD: vary d

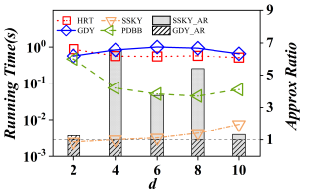


Fig. 19: CORR, MD: vary d

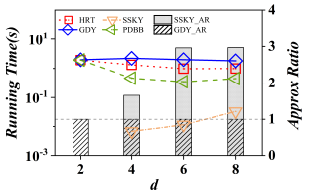


Fig. 20: NBA, 8D: vary d

1) The impact of data dimension d: We vary d from 2 to 10 for synthetic datasets and from 2 to 8 for NBA. Fig. 17 to 20 show the results for the four datasets. Due to the computation of skyline, SUBSKY takes slightly more time as d grows. The running time of HRT and GREEDY is less sensitive to the increase in data dimension. For HRT, two factors contribute to this phenomenon. First, as d increases, the dataset becomes sparser, which results in a higher UC-SKY probability of the candidate set, leading to the pruning of more tuples. Second, the time required to enumerate a set also increases with the growing dimension. For PDBB, because both SUBSKY and GREEDY are used in its preprocessing stage to compute lb, its running time is influenced by theirs, causing it to run slightly slower as d increases. It is worth noting that the performance gap between PDBB and SUBSKY narrows, indicating the effectiveness of the search strategies employed in the final two stages of PDBB. GREEDY always finds the optimal solution with varying d, while SUBSKY can find an approximate solution with a not bad ratio in a very short time.

2) The impact of data size n: We vary n from 10^3 to 10^6 for synthetic datasets and from 20% to 100% for CAR and

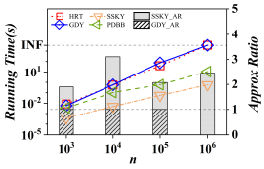


Fig. 21: IND, MD: vary n

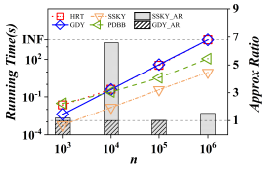


Fig. 22: ANTI, MD: vary n

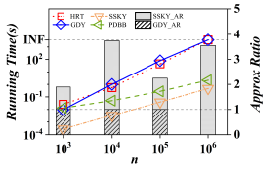


Fig. 23: CORR, MD: vary n

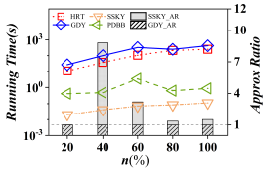


Fig. 24: CAR, 4D: vary n

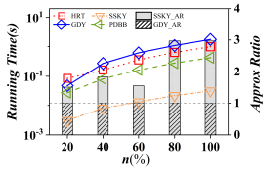


Fig. 25: NBA, 8D: vary n

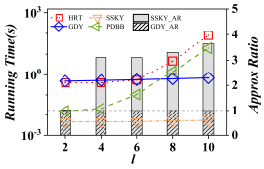


Fig. 26: IND, MD: vary l

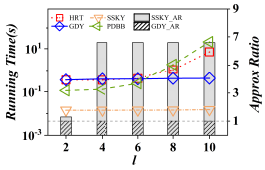


Fig. 27: ANTI, MD: vary l

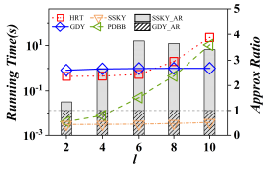


Fig. 28: CORR, MD: vary l

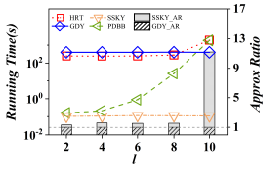


Fig. 29: CAR, 4D: vary l

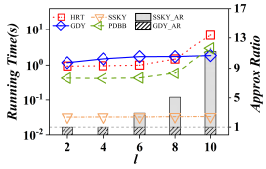


Fig. 30: NBA, 8D: vary l

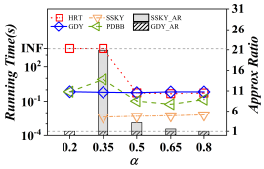


Fig. 31: IND, MD: vary α

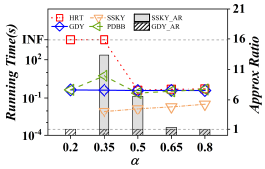


Fig. 32: ANTI, MD: vary α

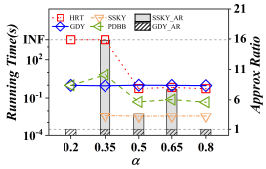


Fig. 33: CORR, MD: vary α

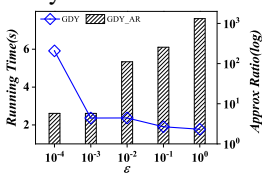


Fig. 34: IIP, 2D: vary ε ($l = 10$)

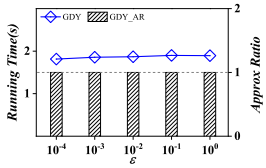


Fig. 35: NBA, 8D: vary ε ($l = 10$)

NBA. The results are shown in Fig. 21 to 25. The relative performance of all algorithms is generally consistent with the results on 2D datasets. SUBSKY shows a great advantage in terms of time efficiency while also having a reasonable approximation ratio. As n increases, the running time of HRT and GREEDY grows significantly. PDBB runs faster and is less sensitive to the increase in data volume due to its effective methods for lb computation, divide-and-conquer strategy, and efficient searching strategy. Notably, on synthetic datasets, both HRT and GREEDY fail to complete within the time limit when $n = 10^6$, whereas PDBB finishes within 100 seconds across all datasets. Despite the increased running time, GREEDY consistently achieves the best approximation ratio.

3) The impact of output size l: l is varied from 2 to 10, and the results are shown in Fig. 26 to 30. Now, all algorithms can complete within the time limit. In other words, as the data dimension grows, the value of l that all algorithms can handle also becomes larger. As mentioned before, this is because the increase in data dimension makes the data sparse, which makes the UC-SKY probability of the candidate set larger, so more tuples are pruned. For example, on IND, when $d = 2$, 52 tuples are left after pruning in PDBB, while when $d = 6$, only 17 tuples remain for processing.

4) The impact of probability center α: We vary $\alpha \in \{0.2, 0.35, 0.5, 0.65, 0.8\}$ for the synthetic datasets. Similar results as for 2D datasets can be observed in Fig. 31 to 33. PDBB and HRT run faster as α increases. However, HRT fails on datasets with smaller α since fewer tuples are pruned. PDBB can also finishes within 100 seconds across all datasets. Meanwhile, due to the sparsity of high-dimensional data, GREEDY achieves the best approximation ratio on CORR now.

5) The impact of step ε in GREEDY*:* ε is varied from 10^{-4} to 1 to assess its effect on the performance of GREEDY. Fig. 34 and 35 present results on IIP and NBA, where l is set to 10. On low-dimensional datasets, ε significantly influences the performance of GREEDY. Generally, as ε increases, its running time and solution quality degrade sharply, with the solution quality experiencing a particularly dramatic decline. Thus, selecting a suitable l is crucial for balancing time efficiency and solution quality. However, on high-dimensional datasets, the impact of ε is minimal when l is small.

VIII. CONCLUSION

In this paper, we revisit the UC-SKY query, which aims to identify a set of l mutually non-dominated uncertain tuples with the highest probability of being contained in the skyline. We first prove that the UC-SKY query is NP-hard. Then, we propose an exact algorithm for the UC-SKY query on 2D data and further optimize it with grouping and pruning strategies. Moreover, we present an exact three-stage algorithm incorporating effective pruning strategies to solve UC-SKY queries on MD data. As byproducts, we introduce two approximation algorithms that offer better time efficiency at the cost of some solution quality. Finally, we conduct extensive experiments over real and synthetic datasets to demonstrate the efficacy and efficiency of our proposed algorithms.

ACKNOWLEDGMENT

This work was partially supported by the National Natural Science Foundation of China grant 62372138, the Natural Science Foundation of Heilongjiang Province of China grant HSF20230095, and the Natural Science Foundation of Hebei Province of China grant F2024210042.

REFERENCES

[1] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings 17th international conference on data engineering*. IEEE, 2001, pp. 421–430.

[2] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C. Kanne, W. Klas, and E. J. Neuhold, Eds. ACM, 2007, pp. 15–26. [Online]. Available: http://www.vldb.org/conf/2007/papers/research/p15-pei.pdf

[3] M. J. Atallah and Y. Qi, "Computing all skyline probabilities for uncertain data," in *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA*, J. Paredaens and J. Su, Eds. ACM, 2009, pp. 279–287. [Online]. Available: https://doi.org/10.1145/1559795.1559837

[4] M. J. Atallah, Y. Qi, and H. Yuan, "Asymptotically efficient algorithms for skyline probabilities of uncertain data," *ACM Trans. Database Syst.*, vol. 36, no. 2, pp. 12:1–12:28, 2011. [Online]. Available: https://doi.org/10.1145/1966385.1966390

[5] Y. Zhang, W. Zhang, X. Lin, B. Jiang, and J. Pei, "Ranking uncertain sky: The probabilistic top-k skyline operator," *Inf. Syst.*, vol. 36, no. 5, pp. 898–915, 2011. [Online]. Available: https://doi.org/10.1016/j.is.2011.03.008

[6] D. Kim, H. Im, and S. Park, "Computing exact skyline probabilities for uncertain databases," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 12, pp. 2113–2126, 2012. [Online]. Available: https://doi.org/10.1109/TKDE.2011.164

[7] X. Liu, D. Yang, M. Ye, and W. Lee, "U-skyline: A new skyline query for uncertain databases," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 945–960, 2013. [Online]. Available: https://doi.org/10.1109/TKDE.2012.33

[8] P. Afshani, P. K. Agarwal, L. Arge, K. G. Larsen, and J. M. Phillips, "(approximate) uncertain skylines," *Theory Comput. Syst.*, vol. 52, no. 3, pp. 342–366, 2013. [Online]. Available: https://doi.org/10.1007/s00224-012-9382-7

[9] J. Liu, H. Zhang, L. Xiong, H. Li, and J. Luo, "Finding probabilistic k-skyline sets on uncertain data," in *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*, J. Bailey, A. Moffat, C. C. Aggarwal, M. de Rijke, R. Kumar, V. Murdock, T. K. Sellis, and J. X. Yu, Eds. ACM, 2015, pp. 1511–1520. [Online]. Available: https://doi.org/10.1145/2806416.2806452

[10] Z. Yang, K. Li, X. Zhou, J. Mei, and Y. Gao, "Top k probabilistic skyline queries on uncertain data," *Neurocomputing*, vol. 317, pp. 1–14, 2018. [Online]. Available: https://doi.org/10.1016/j.neucom.2018.03.052

[11] A. Agrawal, Y. Li, J. Xue, and R. Janardan, "The most-likely skyline problem for stochastic points," *Comput. Geom.*, vol. 88, p. 101609, 2020. [Online]. Available: https://doi.org/10.1016/j.comgeo.2020.101609

[12] W. E. Hick, "On the rate of gain of information," *Quarterly Journal of experimental psychology*, vol. 4, no. 1, pp. 11–26, 1952.

[13] R. Hyman, "Stimulus information as a determinant of reaction time." *Journal of experimental psychology*, vol. 45, no. 3, p. 188, 1953.

[14] S. A. Vinterbo, "A note on the hardness of the k-ambiguity problem," *Harvard Med. School, Boston, MA, USA, Tech. Rep. DSG*, 2002.

[15] Z. Yang, K. Li, X. Zhou, J. Mei, and Y. Gao, "Top k probabilistic skyline queries on uncertain data," *Neurocomputing*, vol. 317, pp. 1–14, 2018.

[16] X. Lian and L. Chen, "Reverse skyline search in uncertain databases," *ACM Transactions on Database Systems (TODS)*, vol. 35, no. 1, pp. 1–49, 2008.

[17] W. Zhang, X. Lin, Y. Zhang, W. Wang, and J. X. Yu, "Probabilistic skyline operator over sliding windows," in *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 2009, pp. 1060–1071.

[18] C. Böhm, F. Fiedler, A. Oswald, C. Plant, and B. Wackersreuther, "Probabilistic skyline queries," in *Proceedings of the 18th ACM conference on Information and knowledge management*, 2009, pp. 651–660.

[19] N. H. M. Saad, H. Ibrahim, F. Sidi, R. Yaakob, and A. A. Alwan, "Efficient skyline computation on uncertain dimensions," *IEEE Access*, vol. 9, pp. 96 975–96 994, 2021.

[20] K. Zhang, J. Wang, M. Wang, and X. Han, "Probabilistic skyline computation on vertically distributed uncertain data," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 154–163.

[21] E. Gavagsaz, "Parallel computation of probabilistic skyline queries using mapreduce," *The Journal of Supercomputing*, vol. 77, no. 1, pp. 418–444, 2021.

[22] S. Abiteboul, P. C. Kanellakis, and G. Grahne, "On the representation and querying of sets of possible worlds," in *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, CA, USA, May 27-29, 1987*, U. Dayal and I. L. Traiger, Eds. ACM Press, 1987, pp. 34–48. [Online]. Available: https://doi.org/10.1145/38713.38724

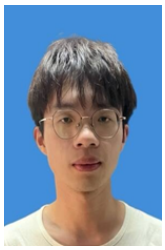
[23] G. S. Lueker, "A data structure for orthogonal range queries," in *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*. IEEE, 1978, pp. 28–34.

[24] P. Ciaccia and D. Martinenghi, "Reconciling skyline and ranking queries," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1454–1465, 2017. [Online]. Available: http://www.vldb.org/pvldb/vol10/p1454-martinenghi.pdf

[25] X. Gao, J. Li, and D. Miao, "Computing all restricted skyline probabilities on uncertain datasets," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 4773–4786.

[26] C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin, "Sliding-window top-k queries on uncertain streams," *The VLDB Journal*, vol. 19, pp. 411–435, 2010.

[27] X. Gao. (2025, January) Source code. [Online]. Available: https://github.com/gaoxy914/UC-SKY



Yikai Zhang is currently working toward the bachelor's degree in computer science and technology with the Shijiazhuang Tiedao University. His main research intersects include uncertain data management and analysis.

Hengzhao Ma received the doctoral degree from the Harbin Institute of Technology. He is currently working as a Lecturer at Software College of Northeastern University. His research interests include computational theory, algorithm design and analysis, and vector database.

Xingxing Xiao received the bachelor's degree in computer science and technology from the Harbin Institute of Technology. He is currently working toward the PhD degree in computer science and technology with the Harbin Institute of Technology. His main research interests include sampling and multi-criteria decision-making.

Xiangyu Gao received the PhD degree in computer science and technology from the Harbin Institute of Technology. He is currently working as a Lecturer at the School of Information Science and Technology, Shijiazhuang Tiedao University. His main research interests include massive data intensive computing and uncertain data mining.

Xiao Pan is a professor with the School of Information Science and Technology, Shijiazhuang Tiedao University. Her research interests include data management on moving objects and location privacy protection.

Dongjing Miao is a professor with the Faculty of Computing, Harbin Institute of Technology. His research interests include database, algorithm and complexity. He has published more than 40 papers in refereed journals and conference proceedings, such as VLDB Journal, IEEE Transactions on Knowledge and Data Engineering, VLDB, ICDE, SIGMOD.

Jianzhong Li is a professor with the Faculty of Computing, Harbin Institute of Technology. He worked with the Department of Computer Science, Lawrence Berkeley National Laboratory, as a scientist, from 1986 to 1987 and from 1992 to 1993. He was also a visiting professor with the University of Minnesota at Minneapolis, Minnesota. His research interests include massive data intensive computing and wireless sensor networks. He has published more than 200 papers in refereed journals and conference proceedings, such as the VLDB Journal, Algorithmica, IEEE Transactions on Knowledge and Data Engineering, SIGMOD, SIGKDD, VLDB, ICDE,and INFOCOM.