Noname manuscript No.
(will be inserted by the editor)

Efficient Algorithms for Uncertain Restricted Skyline Query Processing

Xiangyu Gao · Xingxing Xiao · Xiao Pan · Dongjing Miao · Jianzhong Li

Received: date / Accepted: date

Abstract With the rapid growth of uncertain data, query processing on uncertain data has become an important research area. Although considerable efforts have been devoted to answering certain types of queries on uncertain data, how to perform restricted skyline (rskyline) queries on uncertain data remains an open problem. To fill the gap, this paper studies the all rskyline probabilities (ARSP) problem, which aims to compute the probability of each uncertain tuple appearing in the rskyline, and the most-likely rskyline (MLRS) problem, which aims to identify a set of uncertain tuples with the highest probability of being the rskyline. We prove that no algorithm can solve the ARSP problem in strongly subquadratic time, unless the orthogonal vectors conjecture fails and the MLRS problem is NP-hard. When $\mathcal{F}$ is a set of linear scoring functions subject to a set of linear constraints on weights, we propose two efficient algorithms for solving the ARSP problem. For a special linear constraint, we further develop an algorithm with sublinear query time. For the MLRS problem, we first design a series of data reduction rules to reduce the input data size. Then, we propose two exact algorithms with different search strategies, as well as a local search based approximation algorithm to further improve the time efficiency. Experimental results show that these two problems provide complementary and comprehensive perspectives on rskylines of uncertain datasets, and demonstrate the effectiveness and efficiency of the proposed algorithms.

Keywords Uncertain data · Probabilistic Restricted Skyline · Query Processing

Xiangyu Gao
Shijiazhuang Tiedao University, Shijiazhuang, China
E-mail: gaoxy@stdu.edu.cn

Xingxing Xiao
Harbin Institute of Technology, Harbin, China
E-mail: xiaoxx@hit.edu.cn

Xiao Pan
Shijiazhuang Tiedao University, Shijiazhuang, China
E-mail: smallpx@stdu.edu.cn

Dongjing Miao
Harbin Institute of Technology, Harbin, China
E-mail: miaodongjing@hit.edu.cn

Jianzhong Li
Shenzhen University of Advanced Technology, Shenzhen, China
E-mail: lijz@siat.ac.cn

1 Introduction

The restricted skyline (rskyline) query is widely used in many applications such as multi-criteria decision making [13], personalized recommendation [28], data filtering [8, 34], etc. It extends the skyline query by additionally serving user preferences. Specifically, given a dataset of multidimensional tuples and a set of monotone scoring functions $\mathcal{F}$, the rskyline query retrieves the set of tuples that are not $\mathcal{F}$-dominated by any other tuple. A tuple t is said to $\mathcal{F}$-dominate another tuple s if t is never worse than s under all functions in $\mathcal{F}$. It has been shown that the rskyline query is effective in identifying attractive tuples according to $\mathcal{F}$ and reducing the result size without introducing any notable overhead compared to the skyline query [13]. Due to its effectiveness and wide applications, many algorithms have been proposed to perform rskyline queries on datasets where uncertainty is not involved [13–15, 28, 34]. However, data uncertainty is inherent in rskyline query applications, which is caused by factors such as measurement equipment limitations, privacy issues, data incompleteness, outdated data sources, and so on [27]. Below

are two application scenarios involving performing rskyline queries on uncertain datasets.

E-commerce Scenario: Probabilistic selling is a novel sales strategy in e-commerce [18]. Sellers create probabilistic products by setting the probability of getting any product from a predefined set of products. A typical example is car rentals on *Hotwire*[1]. This platform groups cars with varying horsepower (HP) and miles per gallon (MPG) by price and category, and then abstracts these groups into probabilistic cars. When a customer rents a probabilistic car, the platform will provide the customer with any car from the corresponding group based on the predefined probability distribution. All probabilistic cars essentially form an uncertain dataset for customers to make multi-criteria decisions. Suppose the score of a car is defined as the weighted sum of its attributes HP and MPG. As stated in [34], it is unrealistic to request customers to specify attribute weights with absolute accuracy. Instead, customers are able to specify general preferences like for saving money, MPG is more important than HP. This preference can be represented by the family $\mathcal{F} = \{\omega[1]\text{HP}+\omega[2]\text{MPG} \mid \omega[1] \leq \omega[2]\}$. Performing the rskyline query with such $\mathcal{F}$ on the uncertain dataset of probabilistic cars helps identify choices with high probabilities of getting a car with good fuel economy.

Prediction Service: With the rapid development of machine learning, prediction services are commonly provided in fields such as finance [49], disease control [52], healthcare [35], etc. For instance, given historical data of stock market, algorithms like [12, 49] can predict the price (P) and trading volume (TR) of a stock. The prediction is usually associated with a confidence value, i.e., the probability. All predictions constitute an uncertain dataset. Financial institutions can use this dataset to provide personalized investment recommendations to clients. A client's investment preference can be estimated by having the investor make pairwise comparisons between a set of stocks [25, 41]. Suppose that the client prefers stock $A(100, 200)$ over stock $B(150, 100)$, this induces the family $\mathcal{F} = \{\omega[1]\text{P} + \omega[2]\text{TR} \mid 100\text{P} + 200\text{TR} \geq 150\text{P}+100\text{TR}\}$. Performing the rskyline query with such $\mathcal{F}$ on this uncertain dataset can retrieve stocks with high probabilities of appealing to the client.

To our knowledge, no work has been done to address the problem of conducting rskyline queries on uncertain datasets to date. Existing research on querying uncertain data focused on different tasks, such as skyline queries [5, 38, 47, 48, 51], top-k queries [22, 42, 44], etc. These works only considered uncertainty in data but not in user preferences. And, as stated in [33, 34], it is hardly realistic to determine the precise scoring function of a user as required by top-k queries, and skyline queries lack personalization. Since uncertainty in data is typically modeled by representing each uncertain tuple with a discrete probability distribution over a set of possible instances [2, 5, 6, 22, 26, 37, 38], an alternative approach is to convert an uncertain dataset into a certain one by representing each attribute of an uncertain tuple with an aggregate function, for example, the weighted sum of attributes of its instances, and then perform rskyline queries on the certain dataset. However, as verified by our experiments, such aggregated values lose important distribution information, and uncertain tuples with equal aggregated values but different instances will be treated equally.

To fill the gap, we adopt the possible world semantic [1] and define the rskyline probability of an uncertain tuple and a set of uncertain tuples, respectively. The former captures how likely an uncertain tuple is to be included in the rskyline of a possible world, while the latter reflects the likelihood that a set of uncertain tuples is exactly the rskyline of a possible world. They provide complementary perspectives on rskylines of uncertain datasets. We then formalize the all rskyline probabilities (ARSP) problem to compute rskyline probabilities for all uncertain tuples, and the most-likely rskyline (MLRS) problem to identify the set of uncertain tuples with the largest rskyline probability.

The work most related to these two problems is the research on the all skyline probabilities (ASP) problem [2, 5, 6, 26] and the most-likely skyline (MLS) problem [4, 30]. Our work extends these works by additionally considering all possible scoring functions of a user in $\mathcal{F}$, which also makes the problems more complicated. We begin by investigating the hardness of solving the ARSP problem and the MLRS problem. By establishing a fine-grained reduction from the orthogonal vector problem [11], we prove that no algorithm can solve the ARSP problem within strongly subquadratic time. We also prove that for any $d \geq 2$, the MLRS problem is NP-hard and cannot be approximated within $2^{-O(m^{1-\delta})}$ in polynomial time for any $\delta > 0$ unless P = NP.

Next, we focus on designing efficient algorithms to solve these two problems. Since linear scoring functions are commonly used in practice [16], and user preferences can be effectively captured through constraints on the weights in the linear scoring functions, we propose algorithms for the ARSP problem when $\mathcal{F}$ is a set of linear scoring functions subject to a set of linear constraints on weights. By mapping instances into the score space, we reduce the ARSP problem to the ASP problem. This leads to a near-optimal algorithm. To further improve practical efficiency, we try to perform only necessary mappings, and design an algorithm with bet-

[1] `www.hotwire.com/car-rentals/`

ter expected time complexity based on the branch-and-bound paradigm. Under the ratio constraints [28], we utilize preprocessing techniques to enhance query time. We establish a Turing reduction from the ARSP problem to the half-space reporting problem [3] based on the newly proposed efficient $\mathcal{F}$-dominance test method, which yields a primitive algorithm. Subsequently, we introduce the multi-level strategy and the data-shifting strategy to achieve sublinear query time.

Given the hardness of the MLRS problem, we resort to search strategies for its solution. First, we propose data reduction rules to reduce the input data size, thereby shrinking the search space. We then introduce a baseline algorithm by adapting the state-of-the-art algorithm for the MLS problem [30]. Since $\mathcal{F}$ is taken into account, the adaption is non-trivial. However, the baseline is inefficient as its breadth-first search strategy and loose pruning conditions force it to store a large number of candidates in memory. To address this, we propose a new best-first search algorithm equipped with more effective pruning strategies. Experiments show that it outperforms the baseline by around two orders of magnitude. We also design a polynomial-time approximation algorithm based on local search. The main idea is to first find an approximate result and then refine its using two local search operations. Although without strong quality guarantee in theory, experiments show that it can find a near-optimal solution in most cases.

The main contributions of this paper are summarized as follows.

(1) We propose the ARSP problem to compute the probability of each uncertain tuple appearing in the rskyline, and the MLRS problem to identify a set of uncertain tuples with the highest probability of being the rskyline.
(2) We prove that the ARSP problem cannot be solve in $O(n^{2-\delta})$ time for any constant $\delta > 0$, unless the orthogonal vectors conjecture fails.
(3) We prove that for $d \geq 2$, the MLRS problem is NP-hard and cannot be approximated within $2^{-O(m^{1-\delta})}$ in polynomial time for any $\delta > 0$, unless P = NP.
(4) When $\mathcal{F}$ is a set of linear scoring functions subject to a set of linear constraints on weights, we provide two algorithm with time complexity $O(d'dn + n^{2-1/d'})$ and expected time complexity $O(mn \log n)$ to solve the ARSP problem, where m and n are the number of uncertain tuples and instances, respectively, and d' is the dimensionality of the mapped score space. We also introduce an algorithm specialized for ratio constraints with polynomial preprocessing time and $O(2^{d-1} \log n + n)$ query time.
(5) We design two exact algorithms with time complexity $O(m'^2\gamma^{m'})$ and $O(m'\gamma^{m'})$ for solving the MLRS problem, where m' is the data size after data reduction and $1 < \gamma \leq 2$. We also propose an $O(m'^4)$ time $\tau^l \times 2^{-m'}$-approximation algorithm, where τ is a user-specified threshold and $0 \leq l \leq m' \log_\tau 2$ is the number of local search operations it performs.
(6) We conduct extensive experiments on synthetic and real datasets to verify the efficiency and effectiveness of the proposed algorithms.

The preliminary work of this paper appeared in [21]. As an extension, this paper studies the MLRS problem as detailed in Section 4. And additional experiments are conducted to verify the efficiency and effectiveness of the algorithms proposed for the MLRS problem in Section 5.2.3, 5.2.4, and 5.3.2.

2 Problem Definition and Hardness

2.1 Restricted Skyline

Let D be a d-dimensional dataset consisting of n tuples. Each tuple $t \in D$ has d numeric attributes, denoted by $t = (t[1], \cdots, t[d])$. W.l.o.g., we assume that lower values are preferred to higher ones. Given a scoring function $f : \mathbb{R}^d \to \mathbb{R}^+$, the value $f(t[1], \cdots, t[d])$ is called the score of t under f, also written as $f(t)$. Function f is called monotone if for any two tuples t and s, it holds that $f(t) \leq f(s)$ if $\forall 1 \leq i \leq d : t[i] \leq s[i]$. Let $\mathcal{F}$ be a set of monotone scoring functions, a tuple t $\mathcal{F}$-dominates another tuple $s \neq t$, denoted by $t \prec_{\mathcal{F}} s$, if $\forall f \in \mathcal{F} : f(t) \leq f(s)$ [13]. The restricted skyline (rskyline) of D with respect to $\mathcal{F}$ is the set of tuples that are not $\mathcal{F}$-dominated by any other tuple, i.e., $\text{RSKY}(D, \mathcal{F}) = \{t \in D \mid \nexists s \in D : s \prec_{\mathcal{F}} t\}$ [13].

2.2 Uncertain Restricted Skyline

Let $\mathcal{D}$ denote a d-dimensional uncertain dataset consisting of m uncertain tuples. Each uncertain tuple $T_i \in \mathcal{D}$ is a discrete probability distribution over the d-dimensional data space. In other words, the sample space of T_i is a set of points $\{t_{i,1}, \cdots, t_{i,n_i}\}$ in the d-dimensional data space. Each point $t_{i,j}$ is called an instance of T_i and T_i has probability $p(t_{i,j})$ of occuring as $t_{i,j}$. We also use T_i to denote the set of its instances $\{t_{i,1}, \cdots, t_{i,n_i}\}$ and write $t \in T_i$ to mean that t is an instance of T_i. For any uncertain tuple T_i, we assume $\sum_{t \in T_i} p(t) \leq 1$ and T_i can only take one instance at a time. Let $I = \cup_{i=1}^m T_i$ denote the set of all instances and $n = |I| = \sum_{i=1}^m n_i$. To cope with datasets of large scale, we use a spatial R-tree index to organize I.

Similar to previous work [2,5,6,29,30,38], we adopt the possible world semantic [1] and assume that uncertain tuples are independent of each other. The uncertain dataset $\mathcal{D}$ is interpreted as a probability distribution over a set of datasets $D \sqsubseteq \mathcal{D}$ obtained by sampling each uncertain tuple T_i. The probability of observing the possible world D is $\Pr(D) = \prod_{i=1}^{m} p(D \cap T_i)$. Based on this, we define the rskyline probability of an uncertain tuple and a set of uncertain tuples, respectively.

Definition 1 (RSkyline Probability of an Uncertain Tuple) Given an uncertain dataset $\mathcal{D}$ and a set of monotone scoring functions $\mathcal{F}$, the rskyline probability of an instance $t \in T_i$ is defined as the accumulated possible world probabilities of all possible worlds that have t in their rskylines with respect to $\mathcal{F}$, i.e.,

$$\Pr_{\text{rsky}}(t) = \sum_{D \sqsubseteq \mathcal{D}} \Pr(D) \times \mathbf{1}[t \in \text{RSKY}(D, \mathcal{F})] \tag{1}$$

where $\mathbf{1}[\cdot]$ is the indicator function. The rskyline probability of an uncertain tuple T_i, denoted by $\Pr_{\text{rsky}}(T_i)$, is defined as the sum of rskyline probabilities of all its instances, i.e., $\Pr_{\text{rsky}}(T_i) = \sum_{t \in T_i} \Pr_{\text{rsky}}(t)$.

Definition 2 (RSkyline Probability of a Set of Uncertain Tuples) Given an uncertain dataset $\mathcal{D}$ and a set of monotone scoring functions $\mathcal{F}$, the rskyline probability of a set of instances $S \subseteq I$ is defined as the sum of existence probabilities of possible worlds that have S as their rskylines with respect to $\mathcal{F}$, i.e.,

$$\Pr_{\text{rsky}}(S) = \sum_{D \sqsubseteq \mathcal{D}} \Pr(D) \times \mathbf{1}[S = \text{RSKY}(D, \mathcal{F})], \tag{2}$$

where $\mathbf{1}[\cdot]$ is the indicator function.The rskyline probability of a set of uncertain tuples $\mathcal{S} \subseteq \mathcal{D}$, denoted by $\Pr_{\text{rsky}}(\mathcal{S})$, is defined as the sum of rskyline probabilities of its possible worlds with size $|\mathcal{S}|$, i.e., $\Pr_{\text{rsky}}(\mathcal{S}) = \sum_{S \sqsubseteq \mathcal{S} \wedge |S| = |\mathcal{S}|} \Pr_{\text{rsky}}(S)$[2].

The two problems studied in this paper are formally defined as follows.

Problem 1 : All RSkyline Probabilities (ARSP)
Input: an uncertain dataset $\mathcal{D} = \{T_1, \cdots, T_m\}$, a set of monotone scoring functions $\mathcal{F}$.
Output: rskyline probabilities of all instances in $I = \bigcup_{i=1}^{m} T_i$, i.e., $\text{ARSP} = \{(t, \Pr_{\text{rsky}}(t)) \mid t \in I\}$.

Problem 2 : Most-likely RSkyline (MLRS)
Input: an uncertain dataset $\mathcal{D} = \{T_1, \cdots, T_m\}$, a set of monotone scoring functions $\mathcal{F}$.
Output: a set of uncertain tuples $\mathcal{S}^* \subseteq \mathcal{D}$ with the largest rskyline probability, i.e.,

$$\text{MLRS}(\mathcal{D}, \mathcal{F}) = \mathcal{S}^* = \arg\max_{\mathcal{S} \subseteq \mathcal{D}} \Pr_{\text{rsky}}(\mathcal{S})$$

[2] This definition coincides with the one used in [30].

2.3 Hardness

We first prove that no algorithm can solve the ARSP problem in strongly subquadratic time without preprocessing, unless the orthogonal vectors conjecture fails. All proofs of the claims can be found in the Appendix.

▶ **Orthogonal Vectors Conjecture [11].** Given two sets A, B, each of n vectors in $\{0,1\}^d$, for every $\delta > 0$, there is a $c \geq 1$ such that no $O(n^{2-\delta})$-time algorithm can determine if there is a pair $(a, b) \in A \times B$ such that $a \times b = 0$ with $d = c \log n$.

Theorem 1 *Given an uncertain dataset $\mathcal{D}$ and a set of monotone scoring functions $\mathcal{F}$, no algorithm can solve the ARSP problem with $d = \Theta(\log n)$ in $O(n^{2-\delta})$ time for any constant $\delta > 0$, unless the Orthogonal Vectors conjecture fails.*

Given an uncertain dataset $\mathcal{D} = \{T_1, \cdots, T_m\}$, the most-likely skyline (MLS) problem aims to find a set $\mathcal{S} \subseteq \mathcal{D}$ with the highest skyline probability

$$\begin{aligned}\Pr_{\text{sky}}(\mathcal{S}) &= \sum_{S \sqsubseteq \mathcal{S} \wedge |S| = |\mathcal{S}|} \Pr_{\text{sky}}(S) \\ &= \sum_{S \sqsubseteq \mathcal{S} \wedge |S| = |\mathcal{S}|} \sum_{D \sqsubseteq \mathcal{D}} \Pr(D) \times \mathbf{1}[S = \text{SKY}(D)],\end{aligned}$$

where $\mathbf{1}[\cdot]$ is the indicator function and $\text{SKY}(D)$ is the skyline of D [4,30]. It was proved that when $d \geq 3$, the MLS problem is NP-hard even if $\forall 1 \leq i \leq m : |T_i| = 1$, i.e., each uncertain tuple has only one instance [4,30]. In what follows, we prove that for any $d \geq 2$, the MLRS problem is also NP-hard under this simplified model.

Theorem 2 *The MLRS problem is* NP*-hard for $d \geq 2$.*

Let $\mathcal{A}$ denote a polynomial-time algorithm for the MLRS problem. If for any input $\mathcal{D}$ and $\mathcal{F}$, $\mathcal{A}$ can always return a set $\mathcal{S} \subseteq \mathcal{D}$ such that $\Pr_{\text{rsky}}(\mathcal{S}) \geq c \times \Pr_{\text{rsky}}(\mathcal{S}^*)$, where $S^* = \text{MLRS}(\mathcal{D}, \mathcal{F})$, then we say the MLRS problem can be approximated within c in polynomial time. Since the reduction established in the proof of Theorem 2 guarantees that $\Pr_{\text{sky}}(\mathcal{S}) = \Pr_{\text{rsky}}(\mathcal{S}')$ holds for any $\mathcal{S} \subseteq \mathcal{D}$ and its corresponding set $\mathcal{S}'$ (see Appendix for details), this indicates that if the MLRS problem can be approximated within c in polynomial time, so can the MLS problem. Thus, according to the inapproximability result proved for the MLS problem in [4], the following corollary is derived.

Corollary 1 *For any $\delta > 0$, the MLRS problem can not be approximated within $2^{-O(m^{1-\delta})}$ in polynomial time, unless* P = NP*.*

3 Algorithms for the ARSP problem

The linear scoring function is one of the most commonly used scoring functions in practice [16]. Given a weight $\omega = (\omega[1], \cdots, \omega[d])$ and a tuple t, where $w[i]$ measures the importance of $t[i]$, the score of t is defined as $S_\omega(t) = \sum_{i=1}^{d} \omega[i]t[i]$. Since the ordering of any two tuples under $S_\omega(\cdot)$ is independent of the magnitude of ω, we assume that ω belongs to the unit $(d-1)$-simplex $\mathbb{S}^{d-1}$, i.e., $\forall 1 \leq i \leq d$, $\omega[i] \geq 0$, and $\sum_{i=1}^{d} \omega[i] = 1$. To capture user preferences, a notable approach is to add linear constraints $A \times \omega \leq b$ on weights in $\mathbb{S}^{d-1}$ [17,41], where A is a $c \times d$ matrix and b is a $c \times 1$ vector. The resulting region is called the preference region hereafter and is denoted by $\Omega = \{\omega \in \mathbb{S}^{d-1} \mid A \times \omega \leq b\}$.

In this section, we focus on solving the ARSP problem when $\mathcal{F}$ is of the form $\{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge A \times \omega \leq b\}$. We first propose two baseline algorithms in Section 3.1. Then, in Section 3.2 and 3.3, we propose a space mapping based algorithm that achieves a near-optimal time complexity, and a branch-and-bound algorithm that offers a better expected time complexity and is more practically efficient, respectively. We further explore preprocessing techniques to improve the query time complexity in Section 3.4. We develop an algorithm with polynomial preprocessing time and sublinear query time for a special linear constraint.

3.1 Baseline Algorithms

According to formula (1), a baseline algorithm for the ARSP problem is to enumerate each possible world $D \sqsubseteq \mathcal{D}$, compute $\mathrm{RSKY}(D, \mathcal{F})$, and add $\Pr(D)$ to $\Pr_{\mathrm{rsky}}(t)$ for each $t \in \mathrm{RSKY}(D, \mathcal{F})$. However, this brute-force algorithm is infeasible due to its exponential time complexity. Note that for each $D \sqsubseteq \mathcal{D}$, an instance $t \in T_i$ belongs to $\mathrm{RSKY}(D, \mathcal{F})$ if and only if T_i occurs as t in D and none of $T_{j \neq i}$ $(1 \leq j \leq m)$ appears as an instance that $\mathcal{F}$-dominates t in D. Thus, $\Pr_{\mathrm{rsky}}(t)$ can be equivalently represented as

$$\Pr_{\mathrm{rsky}}(t) = p(t) \times \prod_{j=1, j \neq i}^{m} (1 - \sum_{s \in T_j, s \prec_{\mathcal{F}} t} p(s)). \tag{3}$$

Formula (3) implies that the major challenge in computing $\Pr_{\mathrm{rsky}}(t)$ is to find all instances that $\mathcal{F}$-dominate t for every $T_{j \neq i}$ $(1 \leq j \leq m)$. A straightforward approach is to perform $\mathcal{F}$-dominance tests between t and all instances in $I \setminus T_i$. With the fact that the preference region $\Omega = \{\omega \in \mathbb{S}^{d-1} \mid A \times \omega \leq b\}$ is a closed convex polytope, the $\mathcal{F}$-dominance relation between two instances can be determined by comparing their scores under each vertex of Ω. Here, a weight ω is called a vertex of Ω if and only if it is the unique solution to a d-subset inequalities of $A \times \omega \leq b$.

Theorem 3 ($\mathcal{F}$-dominance test [13]) *Given $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge A \times \omega \leq b\}$, let V be the set of vertices of the preference region $\Omega = \{\omega \in \mathbb{S}^{d-1} \mid A \times \omega \leq b\}$, an instance t $\mathcal{F}$-dominates another instance s if and only if $S_\omega(t) \leq S_\omega(s)$ holds for all weights $\omega \in V$.*

Based on Theorem 3, an alternative baseline algorithm is constructed as follows. Since the preference region Ω is closed, the set of linear constraints can be transformed into a set of points using *polar duality* [40], such that the intersection of the linear constraints is the dual of the convex hull of the points. After the transformation, the baseline algorithm invokes Quickhull algorithm proposed in [7] to compute the set of vertices V of Ω. Next, the algorithm sorts the set of instances using a scoring function $S_\omega(\cdot)$ with some $\omega \in V$. This ensures that if an instance t precedes another instance s in the sorted list, then $s \not\prec_{\mathcal{F}} t$. After sorting, for each instance t, the baseline tests t against every preceding instance s in the sorted list to compute $\Pr_{\mathrm{rsky}}(t)$ according to formula (3). Since V can be computed in $O(c^2)$ time [23], where c is the number of linear constraints, and each $\mathcal{F}$-dominance test can be performed in $O(dd')$ time, where $d' = |V|$, after computing V, the time complexity of the baseline algorithm is $O(c^2 + dd'n^2) = O(dd'n^2)$.

3.2 Space Mapping Algorithm

When $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge A \times \omega \leq b\}$, Theorem 3 essentially implies that the ARSP problem can be reduced to the ASP problem by mapping all instances into the score space defined by the set of vertices of the preference region. With this reduction, the ARSP problem can be solved using efficient algorithms designed for the ASP problem [2,5,6,26]. We introduce an algorithm based on this space mapping and prove that this algorithm achieves a near-optimal time complexity.

We start with the reduction. A tuple t is said to dominate another tuple $s \neq t$, denoted by $t \preceq s$, if $\forall 1 \leq i \leq d : t[i] \leq s[i]$. The skyline probability of an instance $t \in T_i$ is defined as

$$\Pr_{\mathrm{sky}}(t) = p(t) \times \prod_{j=1, j \neq i}^{m} (1 - \sum_{s \in T_j, s \preceq t} p(s)). \tag{4}$$

The ASP problem aims to compute skyline probabilities for all instances [2, 5, 6, 26]. Let $V = \{\omega_1, \cdots, \omega_{d'}\}$ be the set of vertices of the preference region $\Omega = \{\omega \in \mathbb{S}^{d-1} \mid A \times \omega \leq b\}$ and $d' = |V|$. For each $t \in I$, $S_V(t) =$

Algorithm 1: Space Mapping Algorithm

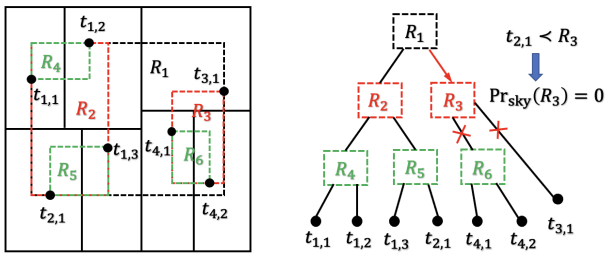
Input: an uncertain dataset $\mathcal{D}$, a set of linear scoring functions $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge A \times \omega \leq b\}$
Output: ARSP

```
1  Compute vertices V of Ω = {ω ∈ S^{d-1} | A × ω ≤ b};
2  Construct the uncertain dataset D';
3  ARSP ← ∅; χ ← 0; β ← 1;
4  foreach i ← 1 to m do σ[i] ← 0;
5  kd-ASP*(I', I');
6  return ARSP;
7  Procedure kd-ASP*(P, C)
8      C_par ← C; C ← ∅; D ← ∅;
9      foreach S_V(t) ∈ C_par do
10         if S_V(t) ⪯ P_min (say t ∈ T_i) then
11             Insert S_V(t) into D;
12             σ[i] ← σ[i] + p(t);
13             if σ[i] = 1 then
14                 χ ← χ + 1; β ← β/p(t);
15             else
16                 β ← β × (1 − σ[i])/(1 − σ[i] + p(t));
17         else if S_V(t) ⪯ P_max then
18             Insert S_V(t) into C;
19     if χ = 0 and |P| = 1 then
20         // suppose P = {S_V(t)} and t ∈ T_i
21         Insert (t, β × p(t)/(1 − σ[i])) into ARSP;
22     else if χ = 0 and |P| > 1 then
23         Partition P into P_l and P_r with selected
           axis;
24         kd-ASP*(P_l, C);
25         kd-ASP*(P_r, C);
26     foreach t ∈ D do
27         Undo the changes to restore σ, χ, β;
28     C ← C_par;
```

$(S_{\omega_1}(t), \cdots, S_{\omega_{d'}}(t))$ is a d'-dimensional point whose i-th coordinate is the score of instance t under $\omega_i \in V$. Theorem 3 states that for any two instances $t, s \in I$, $t \prec_{\mathcal{F}} s$ if and only if $S_V(t) \preceq S_V(s)$. Then, according to formula (3) and (4), it is known that for each $t \in I$, $\Pr_{\text{rsky}}(t) = \Pr_{\text{sky}}(S_V(t))$.

The space mapping algorithm first constructs a d'-dimensional uncertain dataset $\mathcal{D}'$ based on the above reduction. Specifically, for each uncertain tuple $T_i \in \mathcal{D}$, it creates an uncertain tuple T_i' in $\mathcal{D}'$. Then, for each instance $t \in T_i$, it computes $S_V(t)$ as an instance of T_i' and sets $p(S_V(t)) = p(t)$. Finally, the algorithm employs the procedure *kd*-ASP* on $\mathcal{D}'$ to compute skyline probabilities for all instances in $I' = \cup_{i=1}^{m} T_i'$. Algorithm 1 presents the pseudocode of this algorithm.

kd-ASP* is an optimized implementation of the state-of-the-art algorithm for the ASP problem, which is proposed in [2]. The original algorithm first constructs a *kd*-tree T on I', and then progressively computes skyline probability of each instance by performing a pre-order traversal of T. We optimized this algorithm by integrating the preorder traversal into the construction of T, and pruning the construction of a subtree if all instances included in the subtree have zero skyline probabilities. Although these optimizations do not improve the algorithm's asymptotic time complexity, they enhance its experimental performance.



(a) *kd*-tree for I'. (b) Pruning at node R_3.

Fig. 1 Running example for *kd*-ASP*.

kd-ASP* always maintains a path from the root of T to the current node reached, in the main memory. For each node N in the path, let P be the set of instances contained in N, and let $P_{\min}$ and $P_{\max}$ denote the minimum and maximum corners of the minimum bounding rectangle of P, respectively. *kd*-ASP* maintains the following information for each node, (1) a set C including instances that dominate $P_{\max}$, (2) an array $\sigma = \langle \sigma[1], \sigma[2], \cdots \sigma[m] \rangle$, where $\sigma[i] = \sum_{t \in T_i, S_V(t) \preceq P_{\min}} p(t)$, i.e., the sum of probabilities over instances of T_i' that dominate $P_{\min}$, (3) a value $\beta = \prod_{1 \leq i \leq m, \sigma[i] \neq 1} (1 - \sigma[i])$, and (4) a counter $\chi = |\{i \mid \sigma[i] = 1\}|$.

At the beginning, *kd*-ASP* initializes $C = I'$, $\sigma[i] = 0$ for $1 \leq i \leq m$, $\beta = 1$, and $\chi = 0$ at the root node of T. Assuming the information of all nodes in the maintained path is available, *kd*-ASP* constructs the next arriving node N as follows. Let P denote the set of instances in N. For each instance $S_V(t) \in C_{par}$, where C_{par} is the set C of the parent node of N, *kd*-ASP* tests $S_V(t)$ against $P_{\min}$. If $S_V(t) \preceq P_{\min}$, say $t \in T_i$, it updates $\sigma[i]$, β, and χ as stated in lines 12-16 of Algorithm 1. Otherwise, it further tests $S_V(t)$ against $P_{\max}$ and inserts $S_V(t)$ into the set C of N if $S_V(t) \preceq P_{\max}$. When χ becomes one, we know that $\Pr_{\text{sky}}(P_{\min}) = 0$, and thus, all instances in N also have zero probability due to the transitivity of dominance. In this case, *kd*-ASP* prunes the construction of the subtree rooted at N and returns to its parent node. Otherwise, *kd*-ASP* continues growing the path by partitioning set P like a *kd*-tree until it reaches a node containing only one instance $S_V(t)$, at which point it computes $\Pr_{\text{sky}}(S_V(t))$ based on β and σ.

Example 1 As shown in Fig. 1, suppose all instances of an uncertain tuple occur with the same probability. The original algorithm keeps the entire *kd*-tree in the main

memory, while kd-ASP* only maintains a path from the root node, e.g., $R_1 \rightarrow R_2 \rightarrow R_5$. Moreover, when kd-ASP* traverses from R_1 to R_3, it updates $\sigma[2]$ to 1 and χ to 1 since $t_{2,1} \preceq R_3$. This indicates that the skyline probabilities of all instances in the subtree rooted at R_3 are zero. Thus, kd-ASP* prunes the construction of the subtree rooted at R_3 as shown in Fig. 1(b).

The running time of Algorithm 1 comprises of two parts, namely the construction time of $\mathcal{D}'$ and the running time of kd-ASP*. As stated previously, the computation of V takes $O(c^2)$ time [23], where c is the number of linear constraints. After that, for each instance $t \in I$, $S_{_V}(t)$ can be computed in $O(dd')$ time, where $d' = |V|$. According to [2], the running time of kd-ASP* on a set of n instances in d'-dimensional data space is $O(n^{2-1/d'})$. The overall running time of Algorithm 1 is $O(c^2 + d'dn + n^{2-1/d'})$. Although the theoretical upper bound of d' is $\Theta(c^{\lfloor d/2 \rfloor})$ [24], the actual size of V is experimentally observed to be much smaller than n.

Theorem 4 *Given $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge A \times \omega \leq b\}$, Algorithm 1 takes $O(d'dn + n^{2-1/d'})$ time to solve the ARSP problem, where n is the number of instances and d' is the number of vertices of the preference region $\Omega = \{\omega \in \mathbb{S}^{d-1} \mid A \times \omega \leq b\}$.*

The following Theorem proves that Algorithm 1 achieves a near-optimal time complexity.

Theorem 5 *Given an uncertain dataset $\mathcal{D}$ and a set of monotone scoring functions $\mathcal{F}$, even if $\mathcal{F}$ is restricted to the form $\{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge A \times \omega \leq b\}$, no algorithm can solve the ARSP problem with $d = \Theta(\log n)$ in $O(n^{2-\delta})$ time for any constant $\delta > 0$, unless the Orthogonal Vectors conjecture fails.*

Discussion: Algorithm 1 also works when kd-ASP* adopts any other space-partitioning tree. The only modification required is in the method used to partition the data space (lines 23-25 of Algorithm 1). In our experimental study, we implement a variant of Algorithm 1 based on the quadtree, which partitions the data space across all dimensions at each step. We observe that selecting an appropriate space-partitioning tree can significantly improve the performance of Algorithm 1. For instance, the quadtree-based implementation performs well in low-dimensional data spaces, whereas the kd-tree-based implementation exhibits better scalability as the data dimensionality increases.

3.3 Online Mapping Algorithm

A drawback of Algorithm 1 is that it requires a complete mapping of $\mathcal{D}$ to $\mathcal{D}'$. However, assuming instances in I are processed in ascending order based on their scores under some function $f \in \mathcal{F}$, $S_{_V}(t)$ only contributes to computations for instances after t. If ignoring $S_{_V}(t)$ does not impact these computations, the mapping of t to $S_{_V}(t)$ can be avoided, and $S_{_V}(t)$ will not be involved in subsequent computations. To enhance time efficiency, we show how to do the mapping on the fly so that unnecessary computations can be avoided.

Unlike conducting probabilistic analysis under top-k or threshold semantics, it is helpless to maintain upper and lower bounds on each instance's rskyline probability as pruning criteria since our goal is to compute exact rskyline probabilities for all instances. Thus, the only pruning strategy that can be utilized is that if an instance t is $\mathcal{F}$-dominated by another instance s and $\Pr_{\text{rsky}}(s)$ is zero, then $\Pr_{\text{rsky}}(t)$ is also zero due to the transitivity of $\mathcal{F}$-dominance. A straightforward method for efficiently performing this pruning strategy is to maintain an rskyline of all instances processed so far whose rskyline probabilities are zero and compare the next instance to be processed against all instances in the rskyline. However, this maintained rskyline may suffer from huge scale on anti-correlated datasets. In

Algorithm 2: Online Mapping Algorithm

Input: an uncertain dataset $\mathcal{D}$, a set of linear scoring functions $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge A \times \omega \leq b\}$
Output: ARSP

```
1  Compute vertices V of Ω = {ω ∈ S^{d-1} | A × ω ≤ b};
2  Initialize a min-heap H with respect to S_ω(·) and m
     d'-dimensional aggregated R-trees R_1, ··· , R_m;
3  P ← ∅; ARSP ← ∅;
4  Insert the root of R-tree on I into H;
5  while H is not empty do
6      Let N be the top node in H;
7      if N is not pruned by P then
8          if N = {t} is a leaf node (say t ∈ T_i) then
9              Compute S_V(t);
10             Pr_rsky(t) ← p(t);
11             foreach aggregated R-tree R_{j≠i} do
12                 σ[j] ← perform window query with
                     the orign and S_V(t) on R_j;
13                 Pr_rsky(t) ← Pr_rsky(t) × (1 − σ[j]);
14             Insert S_V(t) into R_i;
15             Insert (t, Pr_rsky(t)) into ARSP;
16             p(T_i) ← p(T_i) + p(t);
17             foreach j ← 1 to |V| do
18                 p_i[j] ← max(p_i[j], S_V(t)[j]);
19             if p(T_i) = 1 then Insert p_i into P ;
20         else
21             foreach child node N' of N do
22                 if N' is not pruned by P then
23                     Insert N' into H;
24 return ARSP;
```

what follows, we prove that all instances with zero rskyline probability can be safely ignored, and a set P of size at most m is sufficient for pruning tests.

Theorem 6 *All instances with zero rskyline probability can be safely discarded.*

Theorem 7 *Let $V = \{\omega_1, \cdots, \omega_{d'}\}$ be the set of vertices of the preference region $\Omega = \{\omega \in \mathbb{S}^{d-1} \mid A \times \omega \leq b\}$, there is a set P such that for any instance t, $\Pr_{\text{rsky}}(t) = 0$ if and only if $S_V(t)$ is dominated by some instance $p \in P$ and $|P| \leq m$.*

The pseudocode of the proposed algorithm is shown in Algorithm 2. The algorithm begins by computing the set of vertices V of the preference region Ω, and initializes m aggregated R-trees $R_1, \cdots, R_m$, where R_i is used to incrementally index $S_V(t)$ for all instances $t \in T_i$ with $\Pr_{\text{rsky}}(t) > 0$ that have been processed so far. After that, the algorithm traverses the index R on I in a best-first manner. Specifically, it first inserts the root of R-tree into a minimum heap H, sorted according to its score under some $S_{\omega \in V}(\cdot)$, where the score of a node N is defined as $S_\omega(N_{\min})$. At each step, the algorithm handles the top node N popped from H. If $S_V(N_{\min})$ is dominated by some instance in P, then the algorithm ignores all instances in N since their rskyline probabilities are zero due to the transitivity of $\mathcal{F}$-dominance. Otherwise, if N is a leaf node, say $t \in T_i$ is contained in N, the algorithm computes $S_V(t)$ and issues the window query with the origin and $S_V(t)$ on each aggregated R-tree $R_{j \neq i}$ to compute $\sigma[j] = \sum_{s \in T_j, s \prec_{\mathcal{F}} t} p(s)$. Then, it inserts $S_V(t)$ into R_i. The algorithm updates p_i, which records the maximum corner of the minimum bounding rectangle of $S_V(t)$ for all $t \in T_i$ with $\Pr_{\text{rsky}}(t) > 0$ that have been processed so far, and inserts p_i into P if all instances in T_i have non-zero rskyline probability. Or if N is an internal node, it inserts all non-pruned child nodes of N into H for further computation.

Since $m-1$ orthogonal range queries are performed on aggregated R-trees for each instance in I, the expected time complexity of Algorithm 2 is $O(nm \log n)$. Furthermore, Algorithm 2 only visits nodes that contain instances t with $\Pr_{\text{rsky}}(t) > 0$ and never accesses the same node twice. The number of nodes accessed by Algorithm 2 is optimal for solving the ARSP problem.

Theorem 8 *Given $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge A \times \omega \leq b\}$, Algorithm 2 takes $O(nm \log n)$ time in expectation, where n and m are the number of instances and uncertain tuples, respectively, and the number of nodes accessed by Algorithm 2 is optimal.*

3.4 Sublinear-time Algorithm in a Special Case

We apply preprocessing techniques to further accelerate ARSP computation in a special case. Let $R = \prod_{i=1}^{d-1}[l_i, h_i]$ be a set of user-specified ranges. The ratio constraints R on $\mathbb{S}^{d-1}$ require that $\omega[d] > 0$ and $l_i \leq \omega[i]/\omega[d] \leq h_i$ holds for every $1 \leq i < d$. We use $\omega \models R$ to indicate that ω satisfies the ratio constraints R. Liu et al. studied this special $\mathcal{F}$-dominance, called *eclipse-dominance*, on traditional datasets, and introduced the *eclipse* query to retrieve the set of all non-*eclipse-dominated* tuples [28]. We refer the readers to their paper for the wide applications of the eclipse query.

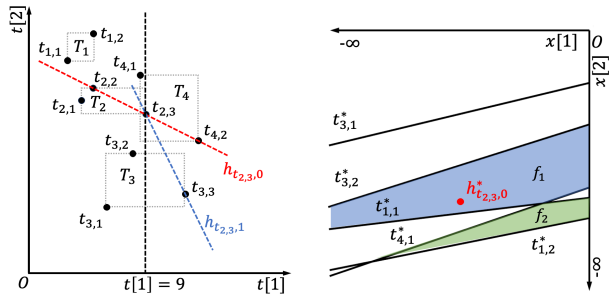
To compute $\Pr_{\text{rsky}}(t)$ for each instance t, it is essential to identify all instances in I that $\mathcal{F}$-dominate t. Under ratio constraints, we observe that the problem of finding all such instances can be Turing reduced to the half-space reporting problem [3]. Thus, we first propose a primitive algorithm with preprocessing based on this reduction in Section 3.4.1. Then, in Section 3.4.2, we introduce the multi-level and shift strategies to achieve sublinear query time.

3.4.1 Reduction to Half-space Reporting Problem

Before introducing the reduction, we start with a new method for determining the $\mathcal{F}$-dominance relation between two instances t and s under the ratio constraints $R = \prod_{i=1}^{d-1}[l_i, h_i]$. Let ω^* be the optimal solution of the following linear programming (LP) problem,

$$\begin{aligned} \text{minimize} \quad & h(\omega) = \sum_{i=1}^{d}(s[i] - t[i]) \times \omega[i] \\ \text{subject to} \quad & l_i \leq \omega[i]/\omega[d] \leq h_i \qquad 1 \leq i < d \\ & \omega[d] > 0, \quad \sum_{i=1}^{d} \omega[i] = 1. \end{aligned} \tag{5}$$

The $\mathcal{F}$-dominance test condition stated in Theorem 3 is equivalent to determining whether $h(\omega^*) \geq 0$. A crucial observation is that the sign of $h(\omega^*)$ can be determined more efficiently without solving problem (5). To be specific, since $\omega[d] > 0$, we can transform the objective function $h(\omega) = \sum_{i=1}^{d}(t[i] - s[i]) \times \omega[i]$ into a new form $h'(r) = \sum_{i=1}^{d-1}(t[i] - st[i]) \times r[i] + (s[d] - t[d])$, where $r[i] = \omega[i]/\omega[d]$. This guarantees that $h(\omega^*) \geq 0$ if and only if $h'(r*) \geq 0$. Here r^* is the optimal solution of problem (5) with $h'(r)$ as the objective function. Since each $r[i] = \omega[i]/\omega[d]$ can be choose independently from the interval $[l_i, h_i]$, r^* can be determined in $O(d)$ time based on the relationships between the first $d-1$ attributes of t and s. Theorem 9 formally states the new $\mathcal{F}$-dominance test condition under the ratio constraints.



(a) Partition data space $\mathbb{R}^2$ with $t_{2,3}$, and two hyper-planes $h_{t_{2,3},0}, h_{t_{2,3},1}$ of $t_{2,3}$.

(b) Dual point $h^*_{t_{2,3},0}$ lies in face f_1 of dual hyper-planes in $I^*_{t_{2,3},0}$.

Fig. 2 An illustration of the reduction and performing point location queries in dual space.

Theorem 9 ($\mathcal{F}$-dominance test under ratio constraints) *Given $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge \omega \models R = \prod_{i=1}^{d-1}[l_i, h_i]\}$, an instance t $\mathcal{F}$-dominates another instance s if and only if $t[d] - s[d] \leq \sum_{i=1}^{d-1}(\mathbf{1}[s[i] > t[i]]l_i + (1 - \mathbf{1}[s[i] > t[i]])h_i) \times (s[i] - t[i])$, where $\mathbf{1}[\cdot]$ is the indicator function.*

Geometrically, Theorem 9 suggests that all instances $\mathcal{F}$-dominating an instance t lie in the intersection of 2^{d-1} half-spaces, which are determined by the outputs of $d-1$ indicator functions. Based on this, we establish a reduction from the problem of finding all such instances in I to a series of 2^{d-1} half-space reporting problem[3] [3]. First, the data space $\mathbb{R}^d$ is partitioned into 2^{d-1} regions using $d-1$ hyper-planes $x[i] = t[i]$ $(1 \leq i < d)$. Each region is uniquely identified by a $(d-1)$-bit binary code. The i-th bit is 0 if the i-th attributes of instances in the region are less than $t[i]$, and 1 otherwise. The region with the binary code k is referred to as region k. Then, assuming $t \in T_i$, define $I_{t,k}$ as the set of instances from other uncertain tuples lying in region k, e.g., $I_{t,0} = \{s \in I \setminus T_i \mid \forall 1 \leq i < d : s[i] < t[i]\}$. For all instances in $I_{t,k}$, the outputs of $d-1$ indicator functions in Theorem 9 are identical. Thus, every instance in $I_{t,k}$ that $\mathcal{F}$-dominates t must lie below or on the following hyper-plane,

$$h_{t,k} : x[d] = \sum_{i=1}^{d-1}((1 - |k|_2[i])l_i + |k|_2[i]h_i) \times (t[i] - x[i]) + t[d], \tag{6}$$

where $|k|_2[i]$ is i-th bit of binary representation of k. That is for each $0 \leq k < 2^{d-1}$, by solving half-space reporting problem $h_{t,k}$ in region k, we can identify all instances in I that $\mathcal{F}$-dominate t.

[3] The half-space reporting problem aims to preprocess a set of points in $\mathbb{R}^d$ into a data structure so that all points lying below or on a query hyper-plane can be reported quickly.

Example 2 See Fig. 2(a) for an illustration of the reduction. Given the ratio constraints $R = [0.5, 2]$, to find all instance in I that $\mathcal{F}$-dominate $t_{2,3}$, the data space $\mathbb{R}^2$ is partitioned by the line $t[1] = t_{2,3}[1] = 9$. Accordingly, $I_{t_{2,3},0} = \{t_{1,1}, t_{1,2}, t_{3,1}, t_{3,2}, t_{4,1}\}$ and $I_{t_{2,3},1} = \{t_{3,3}, t_{4,2}\}$. According to formula (6), the hyper-plane $h_{t_{2,3},0}$ in region 0 is $t[2] = -0.5t[1]+16.5$ and the hyper-plane $h_{t_{2,3},1}$ in region 1 is $t[2] = -2t[1] + 30$. Thus, all instances in $I_{t_{2,3},0}$ (resp., $I_{t_{2,3},1}$) that $\mathcal{F}$-dominate $t_{2,3}$ lie below or on $h_{t_{2,3},0}$ (resp., $h_{t_{2,3},1}$), which includes $t_{3,1}$ and $t_{3,2}$ (resp., $t_{3,3}$).

The half-space reporting problem can be efficiently solved using the well-known *point hyper-plane duality* [31]. The duality maps a point $p = (p[1], \cdots, p[d]) \in \mathbb{R}^d$ to the hyper-plane $p^* : x[d] = p[1]x[1] + \cdots + p[d-1]x[d-1] - p[d]$, and a hyper-plane $h : x[d] = \alpha[1]x[1] + \cdots + \alpha[d-1]x[d-1] - \alpha[d]$ to the point $h^* = (\alpha[1] \cdots, \alpha[d])$. It is proved that if a point p lies above (resp., below, on) a hyper-plane h, then the dual point h^* lies above (resp., below, on) the dual hyper-plane p^*. Thus, the dual version of the half-space reporting problem becomes that given a set of n hyper-planes in $\mathbb{R}^d$ and a query point q, report all hyper-planes lying above or through q. Let H be the set of n hyper-planes in $\mathbb{R}^d$, the *arrangement* of H, denoted by $\mathcal{A}(H)$, is a subdivision of $\mathbb{R}^d$ into *faces* of dimension k for $0 \leq k \leq d$. Each face in $\mathcal{A}(H)$ is a maximal connected region of $\mathbb{R}^d$ where all points lie in the same subset of hyper-planes of H. Fig 2(b) for an illustration. For a query point q, let $\lambda(q, H)$ denote the set of hyper-planes in H lying above or through q. It is easy to verify that all points p lying on the same face f of $\mathcal{A}(H)$ have the same $\lambda(p, H)$, denoted by $\lambda(f, H)$. Therefore, by precomputing $\lambda(f, H)$ for each face f of $\mathcal{A}(H)$ and using an efficient structure for point location in $\mathcal{A}(H)$, e.g., the structure stated in Theorem 10, $\lambda(q, H)$ can be found in logarithmic time.

Theorem 10 (Structure for Point Location [32]) *Given a set H of n hyper-planes in $\mathbb{R}^d$ and a query point q, there is a data structure of size $O(n^{d+\varepsilon})$ which can be constructed in $O(n^{d+\varepsilon})$ expected time for any $\varepsilon > 0$, so that the face of $\mathcal{A}(H)$ containing q can be located in $O(\log n)$ time.*

The primitive algorithm works as follows. In the preprocessing stage, for each instance $t \in I$, say $t \in T_i$, the algorithm partitions $I \setminus T_i$ into 2^{d-1} sets $I_{t,k} = \{s \in I \setminus T_i \mid s$ in region k derived by partitioning $[0, 1]^d$ with $t\}$ $(0 \leq k < 2^{d-1})$. Then, for each set $I_{t,k}$, the algorithm computes the set of dual hyper-planes $I^*_{t,k} = \{s^* \mid s \in I_{t,k}\}$ and constructs a point location structure for $I^*_{t,k}$. Finally, the algorithm constructs $\mathcal{A}(I^*_{t,k})$ and records an array $\sigma_f = \langle \sigma_f[1], \cdots, \sigma_f[m] \rangle$ for each face f of $\mathcal{A}(I^*_{t,k})$, where $\sigma_f[j] = \sum_{s^* \in \lambda(f, I^*_{t,k}) \wedge s \in T_j} p(s)$.

In the query processing stage, given the ratio constraints $R = \prod_{i=1}^{d-1}[l_i, h_i]$, the algorithm processes each instance t as follows. It first initializes $\Pr_{\text{rsky}}(t) = p(t)$ and $\sigma[i] = 0$ for $1 \le i \le m$, where $\sigma[i]$ records the sum of existence probabilities of instances from T_i that $\mathcal{F}$-dominate t found so far. Then, for each $0 \le k < 2^{d-1}$, it compute the dual point $h^*_{t,k}$ of the hyper-plane $h_{t,k}$ defined in formula (6), and performs a point location query $h^*_{t,k}$ on the structure built for $I^*_{t,k}$. Let f be the face returned by the point location query $h^*_{t,k}$, for $1 \le j \le m$, it updates $\Pr_{\text{rsky}}(t)$ to $\Pr_{\text{rsky}}(t) \times (1-\sigma[j]-\sigma_f[j])/(1-\sigma[j])$ and adds $\sigma_f[j]$ to $\sigma[j]$. After processing all queries, it returns $\Pr_{\text{rsky}}(t)$ as the final rskyline probability of t. Since each point location query can be performed in $O(\log n)$ time, and updating $\Pr_{\text{rsky}}(t)$ requires $O(m)$ time for each σ_f, the overall query time complexity is $O(2^d mn \log n)$.

Theorem 11 *When $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge \omega \models R = \prod_{i=1}^{d-1}[l_i, h_i]\}$, the ARSP problem can be solved in $O(2^d mn \log n)$ time after a polynomial-time preprocessing, where n and m are the number of instances and uncertain tuples, respectively.*

Example 3 Continue with Example 2. For instance $t_{2,3}$, in the preprocessing stage, the algorithm will compute the dual hyper-planes of $I_{t_{2,3},0}$ and $I_{t_{2,3},1}$, build point location structures for $I^*_{t_{2,3},0}$ and $I^*_{t_{2,3},1}$, and record σ_f for each face f of $\mathcal{A}(I^*_{t_{2,3},0})$ and $\mathcal{A}(I^*_{t_{2,3},1})$. As an example, the hyper-planes in $I^*_{t_{2,3},0}$ are plotted in Fig. 2(b). For face f_1, it records $\sigma_{f_1}[1] = \sigma_{f_1}[4] = 0$, $\sigma_{f_1}[3] = p(t_{3,1}) + p(t_{3,2}) = 2/3$ since $t^*_{3,1}$ and $t^*_{3,2}$ lie above or through every point in f_1. For face f_2 it records $\sigma_{f_2}[1] = p(t_{1,1}) = 1/2$, $\sigma_{f_2}[3] = p(t_{3,1}) + p(t_{3,2}) = 2/3$, $\sigma_{f_2}[4] = p(t_{4,1}) = 1/2$ since $t^*_{1,1}, t^*_{3,1}, t^*_{3,2}$, and $t^*_{4,1}$ lie above or through every point in f_1.

Let the ratio constraints $R = [0.5, 2]$. To compute $\Pr_{\text{rsky}}(t_{2,3})$, the algorithm first initializes $\Pr_{\text{rsky}}(t_{2,3}) = p(t_{2,3}) = 1/3$ and $\sigma[i] = 0$ $(1 \le i \le 4)$. Then, it performs point location queries $h^*_{t_{2,3},0}$ and $h^*_{t_{2,3},1}$ on $I^*_{t_{2,3},0}$ and $I^*_{t_{2,3},1}$, respectively, to update $\Pr_{\text{rsky}}(t_{2,3})$ and $\sigma[i]$ $(1 \le i \le 4)$. By locating $h^*_{t_{2,3},0} = (-0.5, -16.5)$, which is the dual point of $h_{t_{2,3},0} : t[2] = -0.5t[1] + 16.5$, face f_1 is returned. Since only $\sigma_{f_1}[3] \neq 0$, the algorithm updates $\Pr_{\text{rsky}}(t_{2,3})$ to $\Pr_{\text{rsky}}(t_{2,3}) * (1-\sigma[3]-\sigma_{f_1}[3])/(1-\sigma[3]) = 1/9$ and updates $\sigma[3]$ to $\sigma[3] + \sigma_{f_1}[3] = 2/3$.

3.4.2 Algorithm with Sublinear Query Time

To achieve sublinear query time, two bottlenecks in the primitive algorithm must be addressed. First, for each instance, 2^{d-1} arrays are accessed, and it seems unrealistic to efficiently merge them based on formula (3). Second, since instances are processed sequentially, the query time cannot be less than n. We introduce two strategies to overcome these inefficiencies, respectively.

Multi-level strategy. To compute $\Pr_{\text{rsky}}(t)$, the primitive algorithm performs point location queries in 2^{d-1} regions separately. This results in 2^{d-1} arrays to be accessed. We resort to the *multi-level strategy* [9] to address this issue. In general, a 2^{d-1}-level point location structure is recursively constructed for the set of dual hyper-planes $I^* = \{t^* \mid t \in I\}$. The k-th level is used to identify instances in I that lie below or on the k-th query hyper-plane. Specifically, an one-level structure is the point location tree constructed for I^* with each face f of $\mathcal{A}(I^*)$ recording: (1) an array $\sigma_f = \langle \sigma_f[j] \mid 1 \le j \le m \rangle$, where $\sigma_f[j] = \sum_{s^* \in \lambda(f, I^*) \wedge s \in T_j} p(s)$, (2) a product $\beta_f = \prod_{j=1, \sigma_f[j] \neq 1}^{m} (1 - \sigma_f[j])$, and (3) a count $\chi_f = |\{j \mid \sigma_f[j] = 1\}|$. A k-level structure is an one-level structure constructed for I^* with each face f of $\mathcal{A}(I^*)$ containing an additional $(k-1)$-level structure constructed for $\lambda(f, I^*)$.

After constructing the 2^{d-1}-level structure for I^*, the ratio constraints $R = \prod_{i=1}^{d-1}[l_i, h_i]$ are processed as follows. For each instance t, a set of hyper-planes H and an integer k are initialized as I^* and zero, respectively. While $k < 2^{d-1}$, a dual point $h^*_{t,k}$ is generated according to formula (6), and a point location query $h^*_{t,k}$ is performed on the structure constructed for H. Let f be the face returned by the query. Then, H and k are updated as $\lambda(f, H)$ and $k+1$, respectively. Note that the query $h^*_{t,k}$ is used to find all instances that lie below or on $h_{t,k}$ among the results of the first $k-1$ queries. Let f be the final face returned. Using the information recorded in f, $\Pr_{\text{rsky}}(t)$ is calculated as follows. If $\chi_f = 0$, then $\Pr_{\text{rsky}}(t) = \beta_f \cdot p(t)/(1 - \sigma_f[i])$, or if $\chi_f = 1 \wedge \sigma_f[i] = 1$, $\Pr_{\text{rsky}}(t) = \beta_f \times p(t)$, otherwise, $\Pr_{\text{rsky}}(t) = 0$. Since for each instance $t \in I$, $\Pr_{\text{rsky}}(t)$ can be computed in constant time after performing 2^{d-1} point location queries, the total query time complexity is reduced to $O(2^{d-1} n \log n)$.

Theorem 12 *When $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge \omega \models R = \prod_{i=1}^{d-1}[l_i, h_i]\}$, the ARSP problem can be solved in $O(2^{d-1} n \log n)$ time after a polynomial-time preprocessing using the multi-level strategy.*

Shift strategy. The major challenge in addressing the second bottleneck is that the 2^{d-1} point location queries $h^*_{t,k}$ $(0 \le k < 2^{d-1})$ are different for each instance $t \in I$. The ratio constraints $R = \prod_{i=1}^{d-1}[l_i, h_i]$ can be regarded as a $(d-1)$-dimensional hyper-rectangle. Let $V = \{v = (v[1], \cdots, v[d-1]) \mid \forall 1 \le i < d : v[i] \in \{l_i, h_i\}\}$ be the set of vertices of R. For each $0 \le k < 2^{d-1}$, we call a vertex $v \in V$ the k-vertex of R if there are k vertices preceding v in the lexicographical order of vertices in V, e.g., $(l_1, \cdots, l_{d-1})$ is the 0-vertex of R.

According to formula (6), $h^*_{t,k} = (-v_k[1], \cdots, -v_k[d-1], -(\sum_{i=1}^{d-1} v_k[i]t[i]+t[d]))$. It is worth noting that points $h^*_{t,k}$ and $h^*_{s,k}$ differ only in the last dimension for any two instances t and s. We introduce the *shift strategy* to unify the procedures of performing point location queries across all instances by making the last dimension of their $h^*_{t,k}[d]$ the same for every $0 \leq k < 2^{d-1}$.

Specifically, for each instance $t \in I$, say $t \in T_i$, a shifted dataset I_t is first created by treating t as the origin, i.e., $I_t = \{s - t \mid s \in I \setminus T_i\}$. Then, all sets I_t are merged into a key-value pair set $\mathcal{I} = \{(s, \langle t \mid s \in I_t\rangle) \mid s \in \bigcup_{t\in I} I_t\}$. Finally, a 2^{d-1}-level structure is constructed for the set of dual hyper-planes $\mathcal{I}^* = \{s^* \mid (s,-) \in \mathcal{I}\}$ as described earlier. The information recorded in the one-level structure for each face f of $\mathcal{A}(\mathcal{I}^*)$ is redefined as $\Pr_f = \langle \Pr_f[t] \mid t \in I\rangle$, where $\Pr_f[t] = \prod_{j=1,j\neq i}^{m}(1 - \sum_{s^*\in\lambda(f,\mathcal{I}^*)\wedge s+t\in T_j} p(s))$.

Given the ratio constraints $R = \prod_{i=1}^{d-1}[l_i, h_i]$, 2^{d-1} point location queries $h^*_k = (v_k[1], \cdots, v_k[d-1], 0)$ $(0 \leq k < 2^{d-1})$ are first generated and then executed on the 2^{d-1}-level structure constructed for $\mathcal{I}^*$. Let f be the final face returned, $\Pr_{\text{rsky}}(t)$ of each instance $t \in I$ is computed as $p(t) \times \Pr_f(t)$. Because a total of 2^{d-1} point location queries are performed, the query time complexity is reduced to $O(2^{d-1} \log n)$.

Theorem 13 *When $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge \omega \models R = \prod_{i=1}^{d-1}[l_i, h_i]\}$, the ARSP problem can be solved in $O(2^{d-1} \log n + k)$ time, where k is the size of the final result, after a polynomial-time preprocessing using the multi-level and shift strategies.*

4 Algorithms for the MLRS problem

Similar to [4, 30], in this section, we focus on designing efficient algorithms for the MLRS problem under the simplified model, where each uncertain tuple only has one instance. All proposed algorithms can be extended to handle the multi-instance model with the technique introduced in [30]. Note that when $\mathcal{D} = (T_1 = \{t_1\}, \cdots, T_m = \{t_m\})$, for any $\mathcal{S} \subseteq \mathcal{D}$, $\Pr_{\text{rsky}}(\mathcal{S}) = \Pr_{\text{rsky}}(S)$, where $S = \{t_i \mid T_i = \{t_i\} \in \mathcal{S}\}$. Therefore, the MLRS problem can be solved by identifying a set of instances $S \subseteq I = \{t_1, \cdots, t_m\}$ with the largest $\Pr_{\text{rsky}}(S)$ defined in formula (2). For any possible world D, $S = \text{RSKY}(D, \mathcal{F})$ if and only if all instances in S appear in D and no instance that is not $\mathcal{F}$-dominated by any instance in S appears in D. Therefore, $\Pr_{\text{rsky}}(S)$ can be equivalently represented as

$$\Pr_{\text{rsky}}(S) = \prod_{t\in S} p(t) \times \prod_{t\in I\setminus S:\nexists s\in S: s\prec_{\mathcal{F}} t} (1 - p(t)). \quad (7)$$

We refer to a set of instances $S \subseteq I$ as the *candidate rskyline* if instances in S are not mutually $\mathcal{F}$-dominated.

We first introduce a series of data reduction rules to reduce the input data size in Section 4.1. Then, in Section 4.2, we present a baseline algorithm by modifying the state-of-the-art algorithm for the MLS problem. In Section 4.3, we newly propose a more efficient algorithm with effective pruning strategies following the branch-and-bound paradigm. Finally, in Section 4.4, we introduce an approximation algorithm to further improve time efficiency at the cost of some solution quality.

4.1 Data Reduction

Data reduction rules allow us to discard instances that cannot be part of any optimal solution. The major challenge in designing reduction rules is to ensure that the result derived from the reduced dataset is identical to the result derived from the entire dataset. The core idea is to find an instance t such that all instances that $\mathcal{F}$-dominated by t can be discarded without affecting the final result. As such, any candidate skyline containing a $\mathcal{F}$-dominated tuple s has a lower rskyline probability than an alternative candidate constructed by substituting s with t. Two data reduction rules are designed for the MLS problem in [30]. We first adapt them to the MLRS problem.

Reduction Rule 1 (Point Reduction [30]) *Given an uncertain dataset $\mathcal{D}$ and a set of monotone scoring functions $\mathcal{F}$, if there is an instance t with $p(t) > 1/2$, then all instances $\mathcal{F}$-dominated by t can be discarded.*

Reduction Rule 2 (Region Reduction [30]) *Given an uncertain dataset $\mathcal{D}$ and a set of monotone scoring functions $\mathcal{F}$, if there exist two instances t and s such that $t \prec_{\mathcal{F}} s$ and*

$$\frac{p(t)}{(1-p(t))(1-p(s))} \times \prod_{x\in I: t\prec_{\mathcal{F}} x\prec_{\mathcal{F}} s} (1-p(x))^{-1} > 1, \quad (8)$$

then all instances $\mathcal{F}$-dominated by s can be discarded.

While region reduction is effective, it can be time-consuming Specifically, checking all instances t that $\mathcal{F}$-dominate an instance s for each instance $s \in I$ takes $O(m^3)$ time. When the data size is small, the data reduction algorithm takes up most of execution time. To improve efficiency, we introduce path reduction as a fast approximation of region reduction.

A $\mathcal{F}$-dominant graph $G_{\mathcal{F}}$ of the instance set I is a directed acyclic graph, where nodes represent instances in I, and edges represent their $\mathcal{F}$-dominance relationships. In what follows, words "instance" and "node"

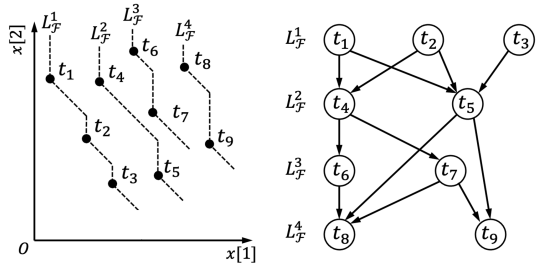


Fig. 3 Maximal rskyline layers and $\mathcal{F}$-dominant graph.

will be used interchangeably. For each node t in $G_{\mathcal{F}}$, the in-neighbor $N_{in}(t)$ of t includes all instances that $\mathcal{F}$-dominate t, and the out-neighbor $N_{out}(t)$ of t consists of all instances that are $\mathcal{F}$-dominated by t. The in-degree of t is $d_{in}(t) = |N_{in}(t)|$, and the out-degree of t is $d_{out}(t) = |N_{out}(t)|$. Nodes in $G_{\mathcal{F}}$ are partitioned into *maximal layers*. The first layer $L^1_{\mathcal{F}}$ is the set of nodes with zero in-degree, i.e., $L^1_{\mathcal{F}} = \text{RSKY}(I, \mathcal{F})$. For $i > 1$, the i-th layer $L^i_{\mathcal{F}}$ is the set of nodes with zero in-degree after removing nodes in $\bigcup_{j=1}^{i-1} L^j_{\mathcal{F}}$ from $G_{\mathcal{F}}$, i.e., $L^i_{\mathcal{F}} = \text{RSKY}(I - \bigcup_{j=1}^{i-1} L^j_{\mathcal{F}}, \mathcal{F})$. We refer to $L^i_{\mathcal{F}}$ as the i-th layer of $G_{\mathcal{F}}$. It is easy to verify that edges can only exist from nodes in $L^j_{\mathcal{F}}$ to nodes in $L^i_{\mathcal{F}}$, where $j < i$. Fig. 3 illustrates an example graph of nine instances with four layers. For clarity, all indirect $\mathcal{F}$-dominance edges such as (t_2, t_6) are omitted.

Given two distinct nodes s and t in $G_{\mathcal{F}}$, a *path* $p(s,t)$ from s to t is an ordered sequence of edges. The nodes s and t are called the *source* and *target* nodes, respectively, while the remaining nodes constitute the set of *internal nodes* $Int(p(s,t))$. For example, as shown in Fig. 3, a path from t_1 to t_8 is $p(t_1, t_8) = ((t_1, t_4), (t_4, t_6), (t_6, t_8))$, and the internal nodes of this path are $Int(p(t_1, t_8)) = \{t_4, t_6\}$.

Reduction Rule 3 (Path Reduction) *Given an uncertain dataset $\mathcal{D}$ and a set of monotone scoring functions $\mathcal{F}$, if there are two instances t, s and a path $p(t,s)$ from t to s in the $\mathcal{F}$-dominant graph $G_{\mathcal{F}}$ such that*

$$\frac{p(t)}{(1-p(t))(1-p(s))} \times \prod_{x \in Int(p(t,s))} (1-p(x))^{-1} > 1, \quad (9)$$

then all instances $\mathcal{F}$-dominated by s can be discarded.

Note that given two instances t and s such that $t \prec_{\mathcal{F}} s$, region reduction considers all instances in $N_{out}(t) \cap N_{in}(s)$, while path reduction only considers instances on a path from t to s. Thus, path reduction is a sufficient condition for region reduction. This proves the correctness of path reduction.

To perform path reduction, for each instance $s \in I$, let $\beta(s)$ record the maximum value of the right hand of formula (9) among all paths ending at s. It follows that

$$\begin{aligned}\beta(s) &= \max_{\substack{p(t,s),\\ t \in N_{in}(s)}} \frac{p(t)}{(1-p(t))(1-p(s))} \times \prod_{x \in Int(p(t,s))} \frac{1}{1-p(x)} \\ &= \max_{(x,s) \in G} \max_{\substack{p(t,x),\\ t \in N_{in}(x)}} \frac{1}{1-p(s)} \times \frac{p(t)}{(1-p(t))(1-p(x))} \\ &\quad \times \prod_{y \in Int(p(t,x))} (1-p(y))^{-1} = \max_{(x,s) \in G} \frac{\beta(x)}{1-p(s)}.\end{aligned}$$

Accordingly, during the construction of the $\mathcal{F}$-dominant graph $G_{\mathcal{F}}$, $\beta(s)$ can be efficiently computed by accessing $\beta(x)$ for all instances $x \in N_{in}(s)$.

Algorithm 3 shows the pseudocode of the newly proposed data reduction algorithm. It performs data reduction during the construction of the $\mathcal{F}$-dominant graph $G_{\mathcal{F}}$. A set P is maintained such that all instances $\mathcal{F}$-dominated by instances in P will be discarded. The algorithm first sorts instances in I in ascending order according to their scores under some function $f \in \mathcal{F}$. It then scans I once to assign each unpruned instance to the appropriate layer in $G_{\mathcal{F}}$. Suppose there are k layers in $G_{\mathcal{F}}$ when instance t_i is processed. The algorithm begins by comparing t_i with instances in layer $L^{\lceil k/2 \rceil}_{\mathcal{F}}$. If t_i is $\mathcal{F}$-dominated by an instance in that layer, then t_i must be assigned to a higher layer, above $\lceil k/2 \rceil$. Otherwise, t_i belongs to that layer or a lower layer. This binary search continues recursively until t_i is assigned

Algorithm 3: Data Reduction Algorithm

Input: an uncertain dataset $\mathcal{D}$, a set of monotone scoring functions $\mathcal{F}$
Output: the reduced dataset $\mathcal{D}$, the $\mathcal{F}$-dominant graph $G_{\mathcal{F}}$

```
1  P ← ∅;
2  Sort I ← {t_1, t_2, ··· , t_m} according to some f ∈ F;
3  foreach i ← 1 to m do
4      if ∄s ∈ P : s ≺_F t_i then
5          j* ← arg min_j ∄s ∈ L^j_F : s ≺_F t_i;
6          L^{j*}_F ← L^{j*}_F ∪ {t_i};
7          foreach s ∈ ⋃_{l=1}^{j*-1} L^l_F do
8              if s ≺_F t_i then
9                  Insert edge (s, t_i) into G;
10                 β(t_i) ← max(β(t_i), β(s)/1 − p(t_i));
11         if p(t_i) > 1/2 then P ← P ∪ {t_i};
12         else if β(t_i) > 1 then P ← P ∪ {t_i};
13         else
14             if ∃s ∈ N_in(t_i) : p(s)/((1−p(s))(1−p(t_i))) ×
               ∏_{x ∈ N_out(s) ∩ N_in(t_i)} (1 − p(x))^{-1} > 1
               then
15                 P ← P ∪ {t_i};
16     else Remove t_i from D ;
17 return D, G_F;
```

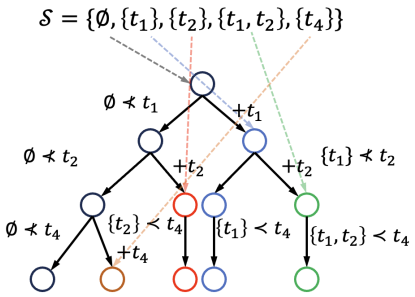


Fig. 4 An illustration of DSA.

to a layer $L_{\mathcal{F}}^{j^*}$. Next, the algorithm compares t_i with each instance $s \in \bigcup_{l=1}^{j^*-1} L_{\mathcal{F}}^{l}$. If $s \prec_{\mathcal{F}} t_i$, it inserts an edge from s to t_i in $G_{\mathcal{F}}$ and update $\beta(t_i)$ based on $\beta(s)$. After that, if $p(t_i) > 1/2$ or $\beta(t_i) > 1$, t_i is inserted into P, as all instances $\mathcal{F}$-dominated by t_i can be safely discarded due to point reduction and path reduction. Otherwise, all instances in $N_{in}(t_i)$ are tested to determine if all instances $\mathcal{F}$-dominated by t_i can be discarded via region reduction.

Due to the inefficiency of region reduction, the worst-case time complexity of Algorithm 3 is still $O(m^3)$. But if region reduction (line 14-15) is omitted, the time complexity of Algorithm 3 improves to $O(m^2)$. We compare these two algorithms in subsequent experimental studies. The results show that path reduction serves as a good approximation of region reduction, which prunes a considerable amount of tuples (1.2% fewer tuples on average) in less time (4 times faster on average).

4.2 Baseline: Dynamic Search based Exact Algorithm

To solve the MLRS problem, our first attempt is to modify the state-of-the-art algorithm for the MLS problem, called DSA [30]. Similarly, a set of instances $S \subseteq I$ is called a *candidate skyline* if instances in S are not mutually dominated. We start with an overview of DSA. The main idea of DSA is to construct all candidate skylines in a *breadth-first* manner and compute their probabilities based on dynamic programming.

Given an uncertain dataset $\mathcal{D}$, DSA first sorts instances in I according to their scores under the function $\min_{1\le i\le d} t[i]$. Let t_i denote the i-th instances in I and $I_i = \{t_1, \cdots, t_i\}$. DSA defines the *partial skyline probability* of a candidate skyline S under set I_i as

$$\Pr_{\text{sky}}^{i}(S) = \prod_{t\in S} p(t) \times \prod_{t\in I_i\setminus S:\nexists s\in S: s\prec t} (1-p(t)).$$

It is easy to verify that $\Pr_{\text{sky}}(S) = \Pr_{\text{sky}}^{n}(S)$. DSA then processes instances in I sequentially to construct candidate skylines and update their partial skyline probabilities as follows. Let $\mathcal{S}$ be the set of candidate skylines constructed by DSA. Initially, $\mathcal{S} = \{\emptyset\}$ and $\Pr_{\text{sky}}^{0}(\emptyset) = 1$. Assume that when processing t_{i+1}, $\Pr_{\text{sky}}^{i}(S)$ is available for each $S \in \mathcal{S}$. DSA computes $\Pr_{\text{sky}}^{i+1}(S)$ and generates new candidate skylines as follows. For each $S \in \mathcal{S}$, DSA checks whether there is an instance $s \in S$ such that $s \prec t_{i+1}$. If so, DSA updates $\Pr_{\text{sky}}^{i+1}(S)$ to $\Pr_{\text{sky}}^{i}(S)$. Otherwise, DSA adds candidate skyline $S\cup\{t_{i+1}\}$ to $\mathcal{S}$ and computes $\Pr_{\text{sky}}^{i+1}(S\cup\{t_{i+1}\})$ as $\Pr_{\text{sky}}^{i}(S) \times p(t_{i+1})$. DSA also updates $\Pr_{\text{sky}}^{i+1}(S)$ to $\Pr_{\text{sky}}^{i}(S) \times (1-p(t_{i+1}))$. Finally, DSA returns the candidate skyline $S \in \mathcal{S}$ with the largest $\Pr_{\text{sky}}^{n}(S)$.

Two pruning strategies are used in DSA to improve efficiency. First, let t_i be the instance currently being processed. A candidate skyline S is removed from $\mathcal{S}$ if it satisfies the *early termination condition*

$$\min_{s\in S} \max_{1\le j\le d} s[i] < \min_{1\le j\le d} t_i[j]. \tag{10}$$

This condition ensures that all unprocessed instances are dominated by some instance in S, implying that $\Pr_{\text{sky}}^{i}(S) = \Pr_{\text{sky}}^{n}(S) = \Pr_{\text{sky}}(S)$. Second, let S^* be the best solution found so far. Since for each candidate skyline S, $\Pr_{\text{sky}}^{i}(S) \ge \Pr_{\text{sky}}(S)$ holds for any i, a candidate skyline S is removed from $\mathcal{S}$ if $\Pr_{\text{sky}}^{i}(S) < \Pr_{\text{sky}}(S^*)$.

Example 4 Fig. 4 illustrates DSA on the uncertain data shown in Fig. 3. The procedure of DSA is represented as a tree. The dashed line points to the node where the candidate skyline is generated for the first time. DSA initializes $\mathcal{S}$ as $\{\emptyset\}$ and $\Pr_{\text{sky}}^{0}(\emptyset) = 1$. Then, DSA processes t_1. Since there is no instance in $\emptyset$ that dominates t_1, $\Pr_{\text{sky}}^{1}(\emptyset)$ is updated to $\Pr_{\text{sky}}^{0}(\emptyset) \times (1-p(t_1))$ (represented by a directed edge from the root to its right child), and $\emptyset \cup \{t_1\}$ is inserted to $\mathcal{S}$ and $\Pr_{\text{sky}}^{1}(\{t_1\})$ is computed as $\Pr_{\text{sky}}^{0}(\emptyset) \times p(t_1)$ (represented by a directed edge from the root to its right child). Note that when processing t_4, since $t_1 \prec t_4$, DSA only computes $\Pr_{\text{sky}}^{3}(\{t_1\})$ as $\Pr_{\text{sky}}^{2}(\{t_1\})$. And no new candidate skyline is inserted to $\mathcal{S}$.

The major obstacle in applying DSA to solve the MLRS problem is the inability to function $\min_{1\le i\le d} t[i]$. This is because it no longer guarantees that after sorting, instances $\mathcal{F}$-dominating t_{i+1} all belong to I_i. For instance, let $t = (3,3)$, $s = (5,2)$, and $\mathcal{F} = \{\omega[1]t[1] + \omega[2]t[2] \mid \omega[1] \ge \omega[2]\}$. It is easy to verify that $t \prec_{\mathcal{F}} s$, but $\min(s[1] = 5, s[2] = 2) = 2 < 3 = \min(t[1] = 3, t[2] = 3)$. The early termination condition stated in formula (10) also fails for the same reason.

We resort to the $\mathcal{F}$-dominant graph $G_{\mathcal{F}}$ to overcome this obstacle. The pseudocode of the modified DSA is presented in Algorithm 4. It follows the basic idea of DSA. Instances are accessed in the topological order of $G_{\mathcal{F}}$. This ensures that for each instance t, all instances

Algorithm 4: Dynamic Search Algorithm

Input: an uncertain dataset $\mathcal{D}$, a set of monotone scoring functions $\mathcal{F}$
Output: MLRS($\mathcal{D}, \mathcal{F}$)

$\mathcal{D}, G_{\mathcal{F}} \leftarrow$ perform data reduction on $\mathcal{D}$;
$\mathcal{S} \leftarrow \{\emptyset\}$;
$S^* \leftarrow \emptyset$; $\Pr_{\text{rsky}}(S^*) \leftarrow 0$;
Sort I following the topological order of $G_{\mathcal{F}}$;
foreach $t \in I$ **do**
  foreach $S \in \mathcal{S}$ **do**
    if $|L(S)| = |L^{i^*}_{\mathcal{F}}|$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S\}$;
      if $\Pr_{\text{rsky}}(S) > \Pr_{\text{rsky}}(S^*)$ **then**
        $S^* \leftarrow S$; $\Pr_{\text{rsky}}(S^*) \leftarrow \Pr_{\text{rsky}}(S)$;
    else if $\nexists s \in S : s \prec_{\mathcal{F}} t$ **then**
      $S' \leftarrow S \cup \{t\}$;
      $\Pr_{\text{rsky}}(S') \leftarrow \Pr_{\text{rsky}}(S) \times p(t)$;
      Collect $L(S')$;
      $\Pr_{\text{rsky}}(S) \leftarrow \Pr_{\text{rsky}} \times (1 - p(t))$;
      if $\Pr_{\text{rsky}}(S') > \Pr_{\text{rsky}}(S^*)$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{S'\}$;
      if $\Pr_{\text{rsky}}(S) < \Pr_{\text{rsky}}(S^*)$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S\}$;
if $\max_{S \in \mathcal{S}} \Pr_{\text{rsky}}(S) > \Pr_{\text{rsky}}(S^*)$ **then**
  $S^* \leftarrow \arg\max_{S \in \mathcal{S}} \Pr_{\text{rsky}}(S)$;
  $\Pr_{\text{rsky}}(S^*) \leftarrow \max_{S \in \mathcal{S}} \Pr_{\text{rsky}}(S)$;
return S^*;

$\mathcal{F}$-dominating t are processed before t. Thus, the modified DSA can construct candidate rskylines and update their partial rskyline probability similar to DSA.

For the early termination condition, its essence is to ensure that all unprocessed instances are $\mathcal{F}$-dominated by some instance in S. To this end, the modified DSA checks if there exists a layer $L^i_{\mathcal{F}}$ in $G_{\mathcal{F}}$ such that $\forall t \in L^i_{\mathcal{F}} : \exists s \in S : s \prec_{\mathcal{F}} t$. Let $L^{i^*}_{\mathcal{F}}$ be the highest layer containing instances in S, i.e. $i^* = \arg\max_i L^i_{\mathcal{F}} \cap S \neq \emptyset$. If instances in S $\mathcal{F}$-dominate all instances in $L^{i^*}_{\mathcal{F}}$, then they $\mathcal{F}$-dominates all instances in each $L^j_{\mathcal{F}}$ ($j \geq i^*$) according to the maximality of each layer in $G_{\mathcal{F}}$. Thus, for each $S \in \mathcal{S}$, the modified DSA maintains a set $L(S) = \{t \in L^{i^*}_{\mathcal{F}} \mid \exists s \in S : s \prec_{\mathcal{F}} t\}$, and removes S from $\mathcal{S}$ if $|L(S)| = |L^{i^*}_{\mathcal{F}}|$. The algorithm updates $L(S)$ as follows. Let t be the currently processed instance. Suppose $t \in L^i_{\mathcal{F}}$ and $i^* = \arg\max_i L^i_{\mathcal{F}} \cap S \neq \emptyset$. When a new candidate rskyline $S' = S \cup \{t\}$ is constructed, the modified DSA sets $L(S')$ as $L(S) \cup \{t\}$ if $i = i^*$. Otherwise ($i > i^*$), it collects $L(S')$ by performing a graph traversal from $L(S) \cup \{t\}$ to $L^i_{\mathcal{F}}$.

Let m' represent the data size after data reduction. The running time of Algorithm 4 depends on the number of candidate rskylines in $\mathcal{S}$. Since a candidate rskyline S is an incomparable instance subset of I, $|\mathcal{S}|$ can be approximated as $O(\gamma^{m'})$ with $1 < \gamma \leq 2$. For each S, it takes $O(m'^2)$ time to collect $L(S')$ by performing the graph traversal and constant time to compute its partial rskyline probability. Therefore, the time complexity of Algorithm 4 is $O(m'^2\gamma^{m'})$.

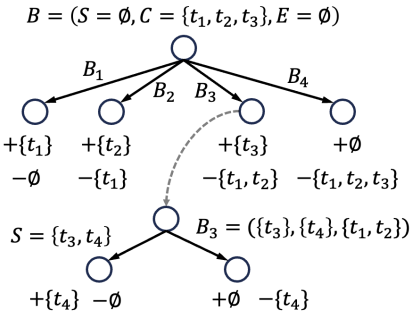


Fig. 5 An illustration of BBA.

Theorem 14 *DSA takes $O(m'^2\gamma^{m'})$ time to exactly solve the MLRS problem, where m' is the data size after data reduction and $1 < \gamma \leq 2$.*

4.3 An Improved Exact Algorithm

Several drawbacks exist in DSA. First, the breadth-first manner makes a large number of candidate rskylines being stored in $\mathcal{S}$. Second, the partial rskyline probability is a loose upper bound, so that only a small number of candidate rskylines are pruned. Third, the properties of unreduced instances are not utilized to speed up the search process. To address these issues, we propose an improved exact algorithm based on branch-and-bound paradigm, called BBA.

Given an uncertain dataset $\mathcal{D}$ and a set of monotone scoring functions $\mathcal{F}$, BBA generates candidate rskylines in a *best-first* manner. It maintains three disjoint sets S, C, and E, where

- S is the set of instances that must be included in every generated candidate rskyline,
- C is the set of instances that can further be added to S to form a valid candidate rskyline,
- E is the set of instances that must not be included in any generated candidate rskyline.

Given (S, C, E), BBA defines the *partial rskyline probability* of a candidate rskyline S as

$$\Pr_{\text{rsky}}(S, E) = \prod_{t \in S} p(t) \times \prod_{t \in E} (1 - p(t)).$$

It is easy to see that if C is empty, then $\Pr_{\text{rsky}}(S, E) = \Pr_{\text{rsky}}(S)$. And for any candidate rskyline $S' \supseteq S$ generated from (S, C, E), $\Pr_{\text{rsky}}(S') \leq \Pr_{\text{rsky}}(S, E)$.

The pseudocode of BBA is shown in Algorithm 5. Unlike traditional branch-and-bound algorithms, which

typically initialize C as I and then refine it based on S and E, BBA keeps the invariant that C always contains all nodes with in-degree zero in $G_{\mathcal{F}} \setminus E$. Here, $G_{\mathcal{F}} \setminus E$ refers to the graph obtained by removing all instances in E from $G_{\mathcal{F}}$. This invariant ensures that for each instance t, t is added to C if and only if all instances that $\mathcal{F}$-dominate t are already included in E. Accordingly, BBA starts with $(S = \emptyset, C = L^1_{\mathcal{F}}, E = \emptyset)$. Suppose BBA reaches the state (S, C, E). According to point reduction, for each instance $t \in C$, if $p(t) > 1/2$, then the out-degree of t in $G_{\mathcal{F}}$ is zero. Hence, BBA first removes all such instances from C and inserts them to S. After that, if C is empty, BBA compares S with the best solution S^* found so far, and then backtracks. Otherwise, BBA sorts instances in C in decreasing order of their probabilities. Let $\{t_1, \cdots, t_k\}$ denote the sorted sequence of instances after sorted. BBA generated $k+1$ branches as follows. For each $1 \leq i \leq k+1$, the i-th branch includes all candidate rskylines that must include $S_i = S \cup \{t_i\}$ and exclude $E_i = E \cup \{t_1, \cdots, t_{i-1}\}$. Here, t_0 and t_{k+1} both correspond to null. The partial rskyline probability of S_i is computed as

$$\Pr_{\text{rsky}}(S_i, E_i) = \Pr_{\text{rsky}}(S, E) \times p(t_i) \times \prod_{1 \leq j \leq i-1} (1 - p(t_j)),$$

where $p(t_0) = 0$ and $p(t_{k+1}) = 1$. BBA dynamically construct C_i for each branch. Initially, C_1 is set to $C \setminus \{t_1\}$. For each subsequent branch $(2 \leq i \leq k+1)$, C_i is constructed as $C_{i-1} \setminus \{t_i\}$, augmented by out-neighbors of t_{i-1} whose in-degree becomes zero after removing t_{i-1} from $G_{\mathcal{F}}$.

Example 5 Fig. 5 illustrates BBA on the uncertain data shown in Fig. 3. According to the $\mathcal{F}$-dominant graph, BBA starts with $(S = \emptyset, C = \{t_1, t_2, t_3\}, E = \emptyset)$. Then, four branches are created as shown in Fig. 5. Based on the invariant, t_4 will be added to C_3 when generating B_3 since the in-degree of t_4 becomes zero when t_1 and t_2 are removed from $G_{\mathcal{F}}$.

Two pruning techniques are used in BBA to improve its efficiency. First, since the partial rskyline probability reaches its final value when C is empty, a tighter upper bound on $\Pr_{\text{rsky}}(S')$ for all candidate rskylines S' generated from (S, C, E) can be derived as follows,

$$\Pr_{\text{rsky}}(S') \leq \Pr_{\text{rsky}}(S, E) \times \prod_{t \in C} \max(p(t), 1 - p(t)) = \Pr^+_{\text{rsky}}(S, C, E),$$

where $\max(p(t), 1-p(t))$ accounts for adding t either to S or to E. Thus, a branch is pruned if its upper bound is less than the rskyline probability of the best solution found so far. Second, when branching, each tuple $t \in C$ satisfies $p(t) \leq 1/2$. Therefore, for $1 \leq i \leq k-1$, it follows that

$$\Pr^+_{\text{rsky}}(S_i, C_i, E_i) = \Pr_{\text{rsky}}(S, E) \times p(t_i) \times \prod_{1 \leq j \neq i \leq k} (1 - p(t_j)) \times \prod_{t \in C_i \setminus C} \max(p(t), 1 - p(t)).$$

Moreover, since $p(t_i) \geq p(t_{i+1})$ and $C_i \setminus C \subseteq C_{i+1} \setminus C$, it is derived that

$$\Pr^+_{\text{rsky}}(S_i, C_i, E_i) \geq \Pr^+_{\text{rsky}}(S_{i+1}, C_{i+1}, E_{i+1}).$$

Therefore, if the i-th branch is pruned, all subsequent branches, except the last one, can also be pruned.

Example 6 We give an example of the second pruning strategy with Fig. 5. At branch $B_2 = (S_2 = \{t_2\}, C_2 = \{t_3\}, E_2 = \{t_1\})$, $\Pr^+_{\text{rsky}}(S_2, C_2, E_2) = p(t_2) \times (1 - p(t_1)) \times \max(p(t_3), 1 - p(t_3))$. At branch $B_3 = (S_3 = \{t_3\}, C_3 = \{t_4\}, E_3 = \{t_1, t_2\})$, $\Pr^+_{\text{rsky}}(S_3, C_3, E_3) = p(t_3) \times (1 - p(t_1)) \times (1 - p(t_2)) \times \max(p(t_4), 1 - p(t_4))$. After point reduction, the in-degree of any instance t is zero if $p(t) \geq 1/2$. Based on the sorting strategy used in BBA, we have $1/2 \geq p(t_2) \geq p(t_3)$. This implies that $\max(p(t_3), 1 - p(t_3)) = 1 - p(t_3) \geq 1 - p(t_2)$. Hence, $\Pr^+_{\text{rsky}}(S_2, C_2, E_2) \geq \Pr^+_{\text{rsky}}(S_3, C_3, E_3)$. Thus, if branch B_2 is pruned, branch B_3 can also be pruned.

The worst-cast running time of BBA depends on the number of branches created. It is upper-bounded by the number of candidate rskylines $O(\gamma^{m'})$, where m' is the number of data size after data reduction and $1 < \gamma \leq 2$. For each branch, BBA takes $O(m')$ to dynamically collect C_i, and constant time to generate S_i and E_i and compute $\Pr_{\text{rsky}}(S_i, E_i)$ (see lines 17-24). Note that if instances in I are pre-sorted in descending order of their probabilities, their relative order will not change when C is sorted in line 15. Hence, at each branch, BBA can use counting sort to sort C based on instances' positions in pre-sorted I within $O(m')$ time. To sum up, the running time of Algorithm 5 is $O(m'\gamma^{m'})$.

Theorem 15 *BBA takes $O(m'\gamma^{m'})$ time to exactly solve the MLRS problem, where m' is the data size after data reduction and $1 < \gamma \leq 2$.*

4.4 A Local Search Approximation Algorithm

Although our experiments show that BBA is quite efficient on small-to-medium sized datasets, its exponential time complexity still prevents it from handling large datasets. Since approximately solving the MLRS problem with a good theoretical guarantee is also NP-hard (see Corollary 1), we propose a local search approximation algorithm, called LSA, in this subsection.

Algorithm 5: Branch-and-Bound Algorithm

Input: an uncertain dataset $\mathcal{D}$, a set of monotone scoring functions $\mathcal{F}$
Output: MLRS($\mathcal{D}, \mathcal{F}$)

$\mathcal{D}, G_{\mathcal{F}} \leftarrow$ perform data reduction on $\mathcal{D}$;
$S^* \leftarrow \emptyset$; $\Pr_{\text{rsky}}(S^*) \leftarrow 0$;
$S \leftarrow \emptyset$; $C \leftarrow L^1_{\mathcal{F}}$; $E \leftarrow \emptyset$; $\Pr_{\text{rsky}}(S, E) \leftarrow 1$;
`BBA-rec`(S, C, E, $\Pr_{\text{rsky}}(S, E)$);
return S^*;
Procedure `BBA-rec`(S, C, E, $\Pr_{\text{rsky}}(S, E)$)
  if $C = \emptyset$ **then**
    if $\Pr_{\text{rsky}}(S, E) > \Pr_{\text{rsky}}(S^*)$ **then**
      $S^* \leftarrow S$; $\Pr_{\text{rsky}}(S^*) \leftarrow \Pr_{\text{rsky}}(S, E)$;
  if $\Pr^+_{\text{rsky}}(S, C, E) \geq \Pr_{\text{rsky}}(S^*)$ **then**
    $C' \leftarrow \{t \in C \mid p(t) > 1/2\}$;
    $C \leftarrow \{t \in C \mid p(t) \leq 1/2\}$;
    $S \leftarrow S \cup C'$;
    $\Pr_{\text{rsky}}(S, E) \leftarrow \Pr_{\text{rsky}}(S, E) \cdot \prod_{t \in C'} p(t)$;
    Sort $C \leftarrow \{t_1, \cdots, t_k\}$ in descending order of $p(t_i)$;
    foreach $i \leftarrow 1$ **to** $k + 1$ **do**
      $S_i \leftarrow S \cup \{t_i\}$;
      $E_i \leftarrow E \cup \{t_1, \cdots t_{i-1}\}$;
      $\Pr_{\text{rsky}}(S_i, E_i) \leftarrow \Pr_{\text{rsky}}(S, E) \cdot p(t_i) \cdot \prod_{j=1}^{i-1}(1 - p(t_j))$;
      if $\Pr^+_{\text{rsky}}(S_i, C_i, E_i) < \Pr_{\text{rsky}}(S^*)$ **then**
        $i \leftarrow k + 1$; **continue**;
      $C_i \leftarrow C \setminus \{t_1, \cdots, t_{i-1}\}$;
      $E_i \leftarrow E \cup \{t_1, \cdots, t_{i-1}\}$;
      Add nodes in $G_{\mathcal{F}} \setminus E_i$ with zero in-degree to C_i;
      `BBA-rec`(S_i, C_i, E_i $\Pr_{\text{rsky}}(S_i, E_i)$);

LSA is designed to find a near-optimal solution in polynomial time. The core idea of LSA is to first find a candidate rskyline S with a loose theoretical guarantee, and then improve its rskyline probability through a series of local search operations. After performing data reduction, LSA generates S by applying the approximation algorithm for the MLS problem proposed in [4]. Specifically, let $R = \{t \in I \mid p(t) > 1/2\}$, LSA initializes S as RSKY($R, \mathcal{F}$). Theorem 16 proves a lower bound of $\Pr_{\text{rsky}}(S)$ in relation to $\Pr_{\text{rsky}}(S^*)$, where S^* denotes the optimal solution to the MLRS problem. This result helps in upper-bounding the number of local search operations performed by LSA.

Theorem 16 *Let $S^* = \text{MLRS}(\mathcal{D}, \mathcal{F})$, then $\Pr_{\text{rsky}}(S) \geq 2^{-m'} \times \Pr_{\text{rsky}}(S^*)$, where m' is the data size after data reduction* [4].

Next, LSA employs two local search operations to iteratively construct a better candidate rskyline based on the initial candidate rskyline S. To perform them efficiently, LSA maintains a set $S(t) = (N_{in}(t) \cup N_{out}(t)) \cap S$ for each instance $t \notin S$. The first local search operation is the *local swap*. A valid local *k-swap* consists of inserting k instances into S and removing all instances that are $\mathcal{F}$-dominated by the new instances, or that $\mathcal{F}$-dominate the new instances, ensuring that S remains valid. LSA attempts to find 1-swaps and decides whether to perform them based on the *gain ratio*. For each instance $t \notin S$, the gain ratio of t, denoted by $r(t)$, is defined as follows. Consider two possible relationships between t and S.

Case 1: $\exists s \in S : s \prec_{\mathcal{F}} t$. In this case, a new candidate rskyline can be constructed by inserting t into S and removing all instances that $\mathcal{F}$-dominate t from S. The gain ratio of this operation is given by

$$r(t) = p(t) \times \prod_{s \in N_{in}(t) \cap S} \frac{1 - p(s)}{p(s)} \times \prod_{\substack{s \in N_{out}(S) - \\ N_{out}(S \cup \{t\} \setminus N_{in}(t))}} (1 - p(s)),$$

where given a set of instances S, $N_{out}(S) = \bigcup_{t \in S} N_{out}(t)$. The first part accounts for the insertion of t, the second part accounts for the removal of instances $\mathcal{F}$-dominating t, and the third part accounts for the instances $\mathcal{F}$-dominated by S but not by $S \cup \{t\} \setminus N_{in}(t)$. The first two parts are easy to compute, while the computation of the third part is time consuming. LSA identifies all relevant instances of the third part by accessing each $s \in I$ and checking whether s is $\mathcal{F}$-dominated by S but not $\mathcal{F}$-dominated by t, and $S(s) \subseteq N_{in}(t) \cap S$. Since updating $S(s)$ for all neighbors of $(N_{in}(t) \cap S) \cup \{t\}$ requires at most $O(|S|m')$ time, and checking the condition for each $s \in I$ takes at most $O(|S|)$ time, the overall time for computing $r(t)$ is $O(|S|m')$.

Case 2: $\nexists s \in S : s \prec_{\mathcal{F}} t$. In this case, a new candidate rskyline can be constructed by inserting t into S and removing any tuple $\mathcal{F}$-dominated by t from S. The gain ratio of this operation is given by,

$$r(t) = \frac{p(t)}{1 - p(t)} \times \prod_{s \in N_{out}(t) \cap S} \frac{1 - p(s)}{p(s)} \times \prod_{\substack{s \in N_{out}(t) - \\ N_{out}(S)}} \frac{1}{1 - p(s)}.$$

Note that if the set $N_{out}(t) \cap S$ is empty, the second part of $r(t)$ will be omitted. To compute $r(t)$, LSA simply takes $O(m'|S|)$ time to check all instances in $N_{out}(t)$.

The second local search operation used in LSA is the post-deletion of redundant instances. An instance $t \in S$ is considered *redundant* if $N_{out}(S) = N_{out}(S \setminus \{t\})$. Removing a redundant instance t from S will multiply $\Pr_{\text{rsky}}(S)$ by $(1 - p(t))/p(t)$. Therefore, LSA removes all redundant instances t with $p(t) < 1/2$ from S. To identify redundant instances, for each instance $t \in S$ with $p(t) < 1/2$, LSA checks whether every $s \in N_{out}(t)$ satisfies $|S(s)| > 1$. If so, LSA removes t from S. Note that whenever an instance t is added to or removed from S,

Algorithm 6: Local Search Algorithm

Input: an uncertain dataset $\mathcal{D}$, a set of monotone scoring functions $\mathcal{F}$, a hyper-parameter τ
Output: a near-optimal candidate rskyline S

$\mathcal{D}, G_{\mathcal{F}} \leftarrow$ perform data reduction on $\mathcal{D}$;
$R \leftarrow \{t \in I \mid p(t) > 1/2\}$;
$S \leftarrow \text{RSKY}(R, \mathcal{F})$; Compute $\Pr_{\text{rsky}}(S)$;
while $\exists t \in I \setminus S : r(t) > \tau$ **do**
  Insert t into S;
  Update S as a valid candidate rskyline;
  $\Pr_{\text{rsky}}(S) \leftarrow \Pr_{\text{rsky}}(S) \times r(t)$;
  foreach $t \in I \setminus S$ **do** Update $r(t)$;
foreach $t \in S$ **do**
  if $N_{out}(S) = N_{out}(S \setminus \{t\})$ **and** $p(t) < 1/2$ **then**
    $S \leftarrow S \setminus \{t\}$;
    $\Pr_{\text{rsky}}(S) \leftarrow \Pr_{\text{rsky}}(S) \times \frac{1-p(t)}{p(t)}$;
return S;

LSA can update $S(s)$ for every $s \in N_{in}(t) \cup N_{out}(t)$ within $O(m')$ time. Therefore, removing all such instance takes at most $O(|S|m')$ time.

The pseudocode of LSA is shown in Algorithm 6. LSA first performs data reduction and initializes S in at most $O(m'^3)$ time. Then, LSA computes $r(t)$ for each $t \notin S$. Whenever there is an instance $t \notin S$ such that $r(t) \geq \tau$, where τ is a user-specified hyper-parameter, LSA inserts t into S and updates $\Pr_{\text{rsky}}(S)$ accordingly. Since the initial S is a $2^{-m'}$-approximate solution and each 1-swap multiplies $\Pr_{\text{rsky}}(S)$ by at least τ, LSA performs at most $O(m' \log_\tau 2)$ 1-swaps. After that, LSA removes all redundant tuples form S in $O(|S|m')$ time. Since the gain ratio of each 1-swap operation can be computed in $O(|S|m')$ time, the overall time complexity of Algorithm 6 is $O(|S|m'^3) = O(m'^4)$.

Theorem 17 *Let l be the number of 1-swaps performed by LSA. LSA can approximately solve the MLRS problem within $\tau^l \times 2^{-m'}$ in $O(m'^4)$ time, where m' is the data size after data reduction.*

5 Experiments

5.1 Experimental Setup

Datasets. Both real and synthetic datasets are used in our experiments. The real data includes three datasets. IIP[4] contains 19,668 sighting records of icebergs with 2 attributes: melting percentage and drifting days. Each record in IIP is associated with a confidence level depending on the sighting method, including R/V (radar and visual), VIS (visual only), RAD (radar only). We treat each record as an uncertain tuple with one instance and convert these three confidence levels to probabilities 0.8, 0.7, and 0.6, respectively. CAR[5] contains 184,810 records with 4 attributes: price, power, mileage, and registration year. To convert CAR into an uncertain dataset, we group cars of the same model into an uncertain tuple T, and for each $t \in T$, we set $p(t) = 1/|T|$, i.e., when a customer wants to rent a specific model of car, any car of that model in the dataset will be offered with equal probability. NBA[6] includes 354,698 game records of 1,878 players with 8 metrics: points, assists, steals, blocks, turnovers, rebounds, minutes, and field goals made. We treat each player as an uncertain tuple T and each record of this player as an instance $t \in T$ with $p(t) = 1/|T|$.

The synthetic datasets are generated with the same procedure described in [5,26,29,38]. Let m be the number of uncertain tuples. For $1 \leq i \leq m$, we first generate the center c_i of each T_i in $[0,1]^d$ following independent (IND), anti-correlated (ANTI), or correlated (CORR) distribution [10]. Then, we generate a hyper-rectangle R_i centered at c_i. The edge length of R_i follows a normal distribution in range $[0, l]$ with a expectation of $l/2$ and a standard deviation of $l/8$. The number of T_i's instances follows a uniform distribution over interval $[1, cnt]$. We generate n_i instances uniformly within R_i and assign each instance the existence probability $1/n_i$. Finally, we remove one instance from the first $\phi \times m$ uncertain tuples so that for any $1 \leq j \leq \phi \times m$, $\sum_{t \in T_j} p(t) < 1$. The expected number of instances in a synthetic dataset is $(cnt/2 - \phi) \times m$.

Monotone Scoring Function Sets. In our experiments, $\mathcal{F}$ consists of all linear scoring functions whose weights satisfy user-specified linear constraints, i.e., $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge A \times \omega \leq b\}$.. Two methods are used to generate the linear constraints on the weights in our experiments. WR specifies weak rankings on weight attributes [17]. Given the number of constraints c, it requires $\omega[i] \geq \omega[i+1]$ for every $1 \leq i \leq c$. IM generates constraints in an interactive manner [41]. Specifically, it first randomly selects a weight ω^* in $\mathbb{S}^{d-1}$. Then, for each $1 \leq i \leq c$, it generates two tuples t_i, s_i uniformly in $[0,1]^d$, and divides $\mathbb{S}^{d-1}$ into two subspaces with $\sum_{j=1}^{d}(t_i[j] - s_i[j]) \times \omega[j] = 0$. IM selects the subspace containing ω^* as the i-th constraint. The main difference between WR and IM is that the preference region generated by WR always has d vertices, while the number of vertices of the preference region generated by IM typically increases with c.

Algorithms. For the ARSP problem, we implement the following algorithms in C++.

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

[5] https://data.world/data-society/used-cars-data

[6] https://www.nba.com/stats/

- ENUM: the first baseline algorithm in Section 3.1.
- LOOP: the second baseline algorithm in Section 3.1.
- KDTT: the kd-tree based space mapping algorithm in Section 3.2.
- KDTT+: the optimized kd-tree based space mapping algorithm in Section 3.2.
- QDTT+: the optimized quadtree based space mapping algorithm in Section 3.2.
- B&B: the online mapping algorithm in Section 3.3.
- DUAL (-M/S): the dual-based algorithm in Section 3.4, where -M is for multi-level strategy, -S is for shift strategy.

And for the MLRS problem, we implement the following algorithms in C++.

- DSA: the baseline algorithm in Section 4.2.
- BBA: the improved exact algorithm in Section 4.3.
- APA: the 2^n-approximation algorithm proposed in [4].
- LSA: the local search approximation algorithm in Section 4.4.

The source code is available at [19, 20]. All algorithms are complied by GNU G++ 7.5.0 with -O2 optimization. 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.

5.2 Effectiveness Results

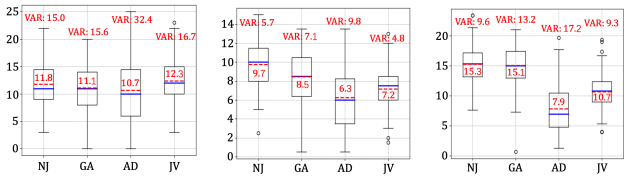
5.2.1 ARSP versus Aggregated RSkyline

We extract records in 2021 from NBA and consider 3 attributes for each player: rebound, assist, and points. We treat each player as an uncertain tuple T and each record of the player as an instance $t \in T$ with $p(t) = 1/|T|$. Let $\mathcal{F} = \{\omega[1]\text{Rebound} + \omega[2]\text{Assist} + \omega[3]\text{Point} \mid \omega[1] \geq \omega[2] \geq \omega[3]\}$. Fig. 10(a) shows the preference region of $\mathcal{F}$. Table 1 reports the top-14 players in rskyline probability ranking along with their rskyline probabilities. We also conduct the rskyline query on the aggregated dataset, which is derived by computing the average statistics for each player. We refer to this result as the aggregated rskyline and mark players in the aggregated rskyline with a "*" sign in Table 1.

We first observe that rskyline probabilities can effectively capture the differences between two incomparable players in the aggregated dataset under $\mathcal{F}$. Theorem 3 claims that $t \prec_{\mathcal{F}} s$ if and only if $\forall \omega \in V, S_\omega(t) \leq S_\omega(s)$, where $V = \{\omega_1 = (1,0,0), \omega_2 = (1/2,1/2,0), \omega_3 = (1/3,1/3,1/3)\}$. See Fig. 6 for scores of Nikola Jokic (NJ) and Jonas Valanciunas (JV) under weights in V. NJ not only performs well on average, placing him in the aggregated rskyline, but also performs best in some games, giving him a high rskyline probability. As for JV, his average performance under ω_1 is great, placing him in the aggregated rskyline as well. However, his large performance variance under ω_1 and relatively poor performance under ω_2 and ω_3 suggest that many of his records are $\mathcal{F}$-dominated by other players' records. This results in his rskyline probability being pretty low. Therefore, compared to aggregated rskyline players with high rskyline probabilities, who consistently perform well, those with low rskyline probabilities are more likely to have many records being $\mathcal{F}$-dominated by other players' records, which may be less attractive.

Table 1 Top-14 players in rskyline probability ranking.

Player	$\Pr_{\text{rsky}}$	Player	$\Pr_{\text{rsky}}$
*Russell Westbrook	0.349	*Rudy Gobert	0.142
*Nikola Jokic	0.331	*Clint Capela	0.134
Giannis Antetokounmpo	0.292	Nikola Vucevic	0.126
James Harden	0.213	Andre Drummond	0.109
Joel Embiid	0.186	Julius Randles	0.109
Luka Doncic	0.168	Kevin Durant	0.101
*Domantas Sabonis	0.162	*Jonas Valanciunas	0.095



(a) $S_{\omega_1}(\cdot)$ (b) $S_{\omega_2}(\cdot)$ (c) $S_{\omega_3}(\cdot)$

Fig. 6 Boxplots of players' scores under $\omega_1 = (1,0,0)$, $\omega_2 = (1/2,1/2,0)$, and $\omega_3 = (1/3,1/3,1/3)$, where average is marked with red dotted lines.

Second, we find that players not in the aggregated rskyline but have high rskyline probabilities is also appealing. For example, Giannis Antetokounmpo is $\mathcal{F}$-dominated by Nikola Jokic in the aggregated dataset, but his rskyline probability is higher than another four aggregated rskyline players. Compared to NJ, his scores (GA) have slightly lower averages and higher variances. In other words, he has some records, like NJ's, which $\mathcal{F}$-dominates most of other players' records and he also has some records that are $\mathcal{F}$-dominated by many of other players' records. Besides, the large performance variance also explains why Andre Drummond (AD) is $\mathcal{F}$-dominated by JV but has a higher rskyline probability. This suggests that looking for players with high rskyline probabilities can find excellent players with slightly lower averages but higher variances in performance.

Finally, a set of players with specified size can be retrieved by performing top-k queries on ARSP, while the size of the aggregated rskyline is uncontrollable. From these observations, we conclude that ARSP provides a

Table 2 Top-14 players in skyline probability ranking.

Player	$\Pr_{\text{sky}}$	**Player**	$\Pr_{\text{sky}}$
Nikola Jokic	0.557	LeBron James	0.308
Russell Westbrook	0.537	Domantas Sabonis	0.283
Giannis Antetokounmpo	0.479	Stephen Curry	0.266
James Harden	0.447	Kevin Durant	0.257
Luka Doncic	0.398	Nikola Vucevic	0.236
Joel Embiid	0.339	Julius Randle	0.224
Trae Young	0.309	Damian Lillard	0.208

more comprehensive view on uncertain datasets than the aggregated rskyline.

5.2.2 ARSP versus ASP

We also compare the distinction between the ASP problem and the ARSP problem. Similar to Table 1, Table 2 also reports the top-14 players in skyline probability ranking along with their skyline probabilities. By analyzing these results, we obtain several interesting observations. First, the rskyline probability of an uncertain tuple is typically smaller than its skyline probability since $\mathcal{F}$ improves each instance's dominance capability. But excellent players like Nikola Jokic, Russell Westbrook consistently maintain high rankings in both skyline probability and rskyline probability. Second, uncertain rskyline queries better serve specific preferences of individual users. Given different inputs $\mathcal{F}$ from different users, the rskyline probabilities of uncertain tuples are variant. In contrast, the skyline probabilities of uncertain tuples always remain the same. Moreover, since a skyline tuple may be $\mathcal{F}$-dominated by other tuples, an uncertain tuple with a high skyline probability may have a low rskyline probability, making it less attractive under $\mathcal{F}$. For instance, Trae Young's skyline probability is 0.309 (ranked 7th) but his rskyline probability under $\mathcal{F} = \{\omega[1]\text{Rebound} + \omega[2]\text{Assist} + \omega[3]\text{Point} \mid \omega[1] \geq \omega[2] \geq \omega[3]\}$ is only 0.029 (ranked 31st). These observations demonstrate that ARSP is more flexible than ASP in capturing individual preferences.

5.2.3 MLRS versus MLS

We then make a comparison between the MLRS problem and the MLS problem on CAR. We consider each car as an uncertain tuple with one instance having three attributes: price, mileage, and power. Following [30], we use the hidden Markov model (HMM) shown in Fig. 7 to infer each car's available probability. Fig. 8 plots 50 samples from the dataset. Suppose a shopper is selecting a used car form the dataset. To help with decision making, the shopper's preferences are refined based on the feedback on pairs of cars. For example, the preference for $t_1 = (180, 600, 160)$ over $s_1 = (116, 1500, 178)$ induces the constraint $180\omega[1] + 600\omega[2] + 160\omega[3] \leq 116\omega[1] + 1500\omega[2] + 178\omega[3]$, i.e., $82\omega[0] - 882\omega[1] \leq 18$. The resulting preference region is depicted in Fig. 10(b). Cars in MLRS under $\mathcal{F} = \{\omega[1]\text{Price} + \omega[2]\text{Mileage} + \omega[3]\text{Power} \mid 82\omega[0] - 882\omega[1] \leq 18\}$ are surrounded by squares in Fig. 8. For comparison, MLS is also computed, and cars in MLS are surrounded by circles. The first observation is that MLS contains more cars than MLRS and has a lower probability than MLRS. In our case, MLS contains 13 cars with a probability of 1.98×10^{-5}, while MLRS contains 9 cars with a probability of 1.41×10^{-3}. This is because $\mathcal{F}$ improves the dominance ability of each car, making it possible for different candidate skylines to produce the same candidate rskyline.. Thus, candidate skylines are grouped together to form candidate rskylines with higher probabilities. Second, MLRS is not always a subset of MLS, which contrasts with the relationship between rskyline and skyline. For instance, as shown in Fig. 8, the car marked by the green arrow belongs to MLRS but not to MLS. This occurs because the rskyline probability of each candidate rskyline is the cumulative sum of skyline probabilities of candidate skylines in its forming group. Hence, there is no guarantee that the group containing MLS still has the highest probability. Assume the shopper further expresses a preference for $t_2 = (182, 900, 137)$ over $s_2 = (109.8, 200, 160)$. This adds the constraint $95.2\omega[0] + 723\omega[1] \geq 23$. MLRS is refined by excluding the car marked by the red arrow, and its joint probability increases to 3.77×10^{-3}. These observations show that MLRS offers greater flexibility than MLS in capturing individual preferences and that users can increase its probability by refining their preferences.

5.2.4 MLRS versus ARSP

Finally, we compare the two proposed uncertain rskyline queries on CAR. Fig. 9 plots their results under $\mathcal{F} = \{\omega[1]\text{Price} + \omega[2]\text{Mileage} + \omega[3]\text{Power} \mid \omega[1] \geq \omega[2] \geq \omega[3]\}$. MLRS consists of five cars (surrounded by squares). For comparison, we retrieve cars with top-5 rskyline probabilities (surround by circles). However, it is noticed that instances in the top-5 result are mutually $\mathcal{F}$-dominated. In other word, the top-5 result is not a valid rskyline in any possible world. As shown in Fig. 9, t is always a better choice than s. This implies that it is meaningless to provide s to the shopper for further decision-making. A subsequent rskyline computation can be used to filter such instances. However, this affects the diversity of the top-5 result. Meanwhile, the top-5 result does not provide any cars with low mileage,

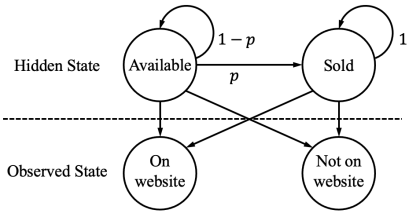


Fig. 7 HMM for generating probabilities of instances in CAR.

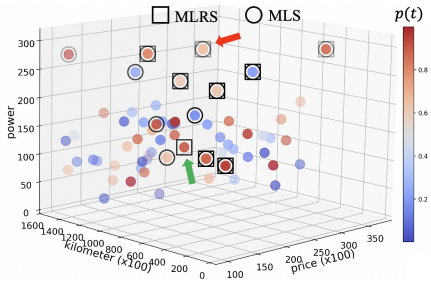


Fig. 8 Comparison between MLRS and MLS on CAR.

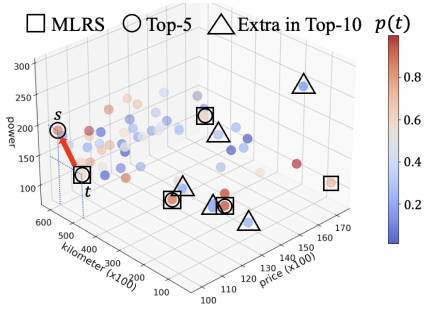
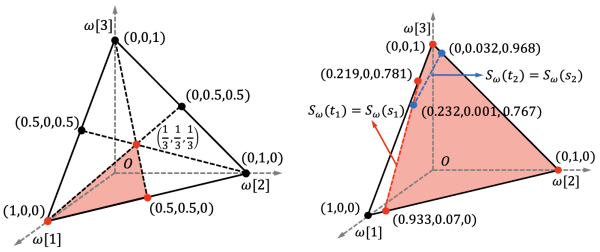


Fig. 9 Comparison between MLRS and ARSP on CAR.



(a) WR (b) IM

Fig. 10 Preference region.

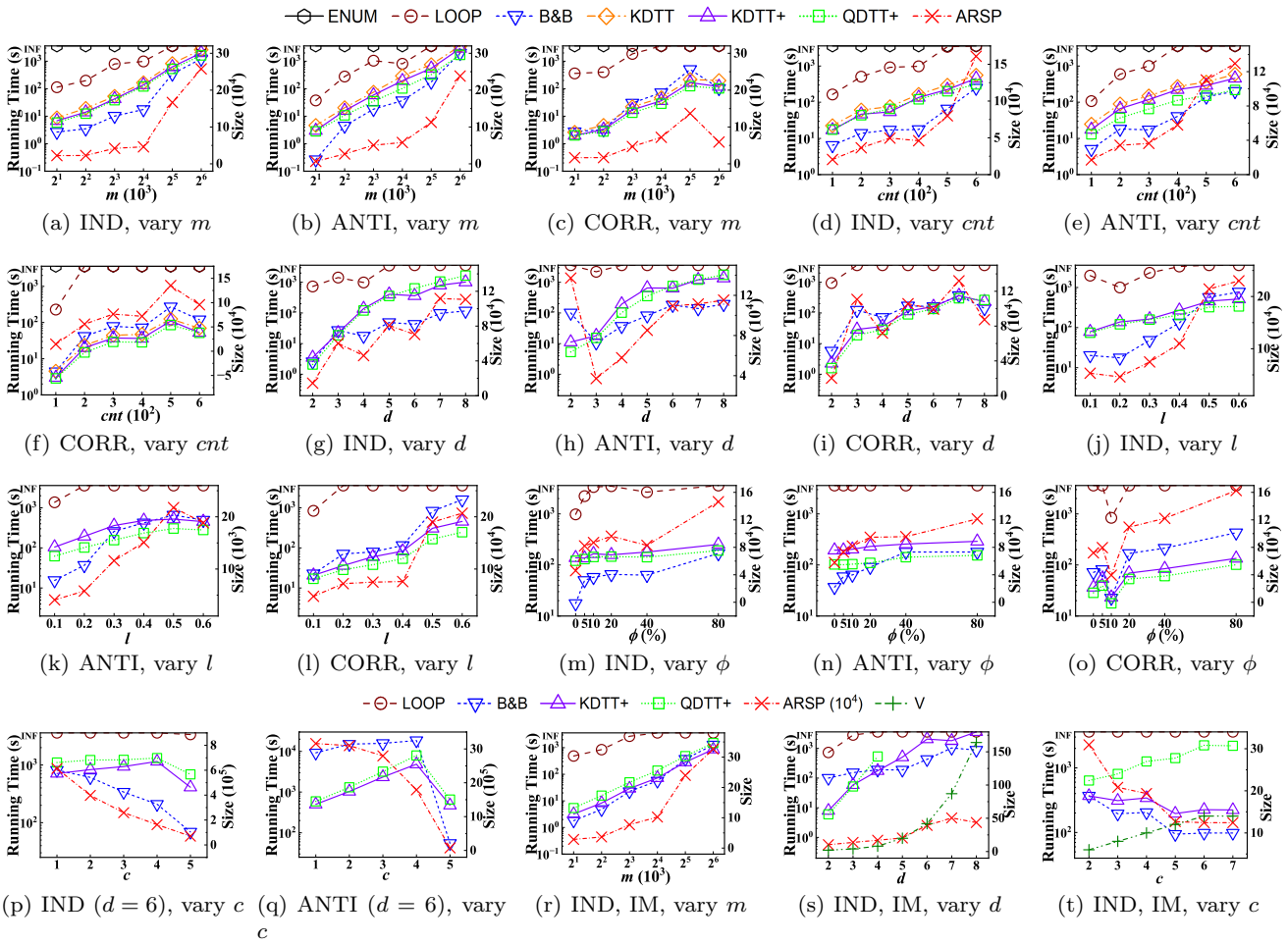
even though one such car is part of the MLRS. To enlarge the coverage of MLRS, we reset $k = 10$ (lowest value). But at the same time, this also includes more instances that are $\mathcal{F}$-dominated by others. It is challenging to specifying a proper k to balance diversity and incomparability. MLRS does not face this trade-off. It always provides insights into a set of non-$\mathcal{F}$-dominated instances with the highest probability of being the rskyline across all possible worlds. However, due to the enormous number of candidate rskylines, the joint probability may be naturally small. As a conclusion, these two uncertain rskyline queries provide complementary perspectives on rskylines of uncertain datasets and may together serve as a better guidance for multi-criteria decision making on uncertain datasets. For example, one possible appropriate method is to first find MLRS and then return a subset of MLRS that also has a high individual probability of being in the rskyline.

5.3 Efficiency Results

5.3.1 Algorithms for the ARSP problem

Fig. 11 and 12 show the running time of different algorithms and the size of ARSP on real and synthetic datasets. The size of ARSP refers to the number of instances with none-zero rskyline probabilities. Following [26, 38], the default values for uncertain tuple cardinality m, instance count cnt, data dimensionality d, region length l, and percentage ϕ of uncertain tuples with $\sum_{t\in T} p(t) < 1$ of synthetic datasets are set as $m = 16$K, $cnt = 400$, $d = 4$, $l = 0.2$, and $\phi = 0$. Unless otherwise stated, WR is used to generate $c = d - 1$ input linear constraints for $\mathcal{F}$. All datasets are indexed by R-trees in main memory. Since the construction time of the R-tree is a one-time cost for all subsequent queries, it is not included in the query time. The query time limit (INF) is set as 3,600 seconds.

Fig. 11 (a)-(c) present the results on synthetic datasets with m varying from 2K to 64K. Based on the generation process, the number of instances n increases as m grows. Thus, the running time of all algorithms and the size of ARSP increase. Due to its exponential time complexity, ENUM never finishes within the limited time. All proposed algorithms outperform LOOP by around an order of magnitude. This is because LOOP always performs a large number of $\mathcal{F}$-dominance tests and does not include any pruning strategy. B&B runs fastest on IND and ANTI with the help of the incremental mapping and pruning strategies. As m grows, the performance gap between B&B and other algorithms narrows because the more uncertain tuples, the more aggregated R-trees are queried per instance. KDTT+ and QDTT+ are more effective on CORR because pruning is triggered earlier by uncertain tuples near the origin during space partitioning. For example, when $m = 2$K, QDTT+ prunes 13 child nodes of the root node on CORR, compared to 9 on IND and 5 on ANTI. Although with similar strategies, QDTT+ performs better than KDTT+. The reason is that space is recursively divided into 2^d regions in QDTT+, which results in a smaller MBR and thus a greater possibility of being pruned. Results also demonstrate our optimization techniques significantly improve the experimental performance of KDTT. As shown in Fig. 11 (d)-(f), the relative performance of all algorithms remains basically unchanged with respect to cnt. And the size of ARSP also increases as cnt grows since the larger cnt, the more



(a) IND, vary m (b) ANTI, vary m (c) CORR, vary m (d) IND, vary cnt (e) ANTI, vary cnt

(f) CORR, vary cnt (g) IND, vary d (h) ANTI, vary d (i) CORR, vary d (j) IND, vary l

(k) ANTI, vary l (l) CORR, vary l (m) IND, vary ϕ (n) ANTI, vary ϕ (o) CORR, vary ϕ

(p) IND ($d = 6$), vary c (q) ANTI ($d = 6$), vary c (r) IND, IM, vary m (s) IND, IM, vary d (t) IND, IM, vary c

Fig. 11 Running time of different algorithms and the size of ARSP on synthetic datasets.

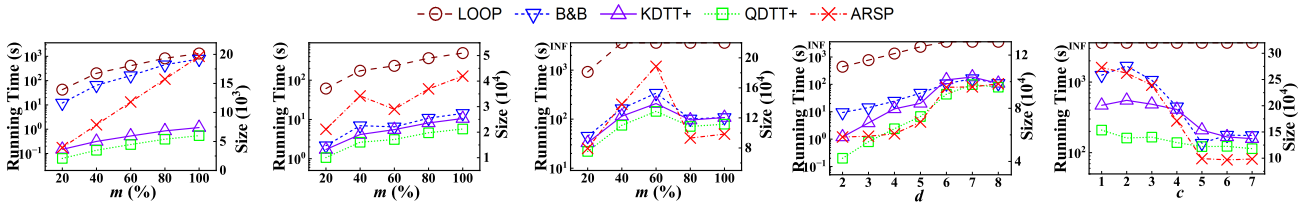
instances in I and the less likelihood of an instance being $\mathcal{F}$-dominated by all instance of an uncertain tuple.

Having established ENUM is inefficient for computing ARSP, it is excluded from the following experiments. The curve of KDTT is also omitted as it is consistently outperformed by KDTT+. Fig. 11 (g)-(i) plot the results on synthetic datasets with varying dimensionality d. With the increase of d, the cost of $\mathcal{F}$-dominance test increases. Thus, the running time of all algorithms increases. QDTT+ and KDTT+ are more efficient than B&B on low-dimensional datasets, but their scalability is relatively poor. This is because, as d grows, the dataset becomes sparser, causing the subtrees pruned during the preorder traversal in KDTT+ and QDTT+ to get closer to leaf nodes. Moreover, the exponential growth in the number of child nodes of QDTT+ also contributes to its inefficiency on high-dimensional datasets. When the dataset becomes sparser, it is more likely that an instance is not $\mathcal{F}$-dominated by others. Therefore, the size of ARSP increases with higher dimensionality.

Fig. 11 (j)-(l) show the effect of l by varying l from 0.1 to 0.6. As l increases, the number of instances $\mathcal{F}$-dominated by all instances of an uncertain tuple decreases. Thus, the size of ARSP and the running time of all algorithms increase. Compared to others, B&B is more sensitive to l since it determines not only the number of instances to be processed but also the time consumed in querying aggregated R-trees.

Fig. 11 (m)-(o) show the runtime of different algorithms and the size of ARSP on synthetic datasets with different ϕ. According to formula 3, the more uncertain tuples T with $\sum_{t \in T} p(t) < 1$, the less instances with zero rskyline probabilities. Hence, the running time and the size of ARSP both increase as ϕ increases. Similar to l, ϕ also significantly impacts B&B since the larger ϕ, the fewer instances are added to the pruning set P.

Fig. 11 (p)-(q) plot the effect of c on IND and ANTI ($d = 6$). Results on CORR are omitted, in which the running time of all algorithms and the size of ARSP decrease as c grows. The reason is that the preference region narrows with the growth of c, which enhances the $\mathcal{F}$-dominance ability of each instance. Therefore, more



(a) IIP, vary m (b) CAR, vary m (c) NBA, vary m (d) NBA, vary d (e) NBA, vary c

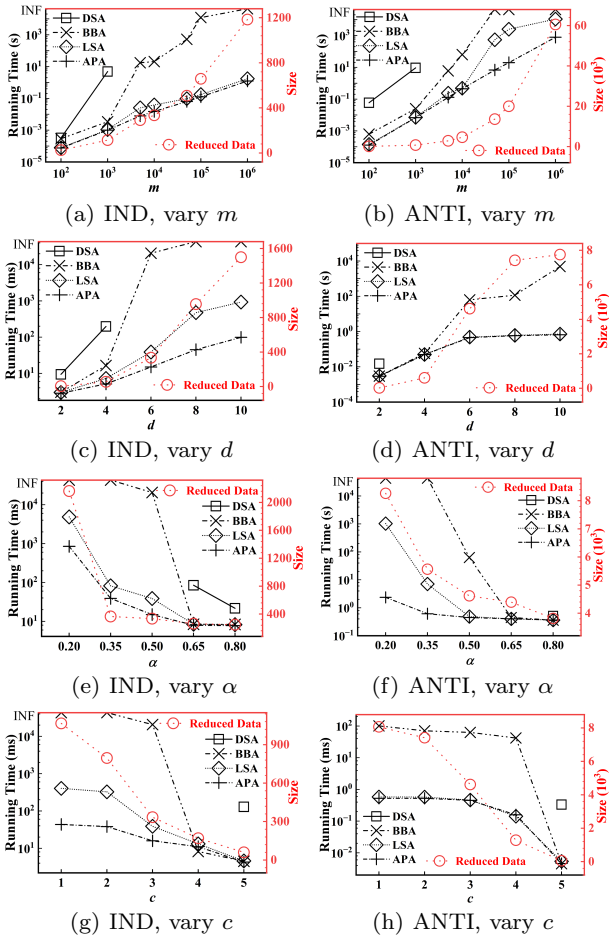
Fig. 12 Running time of different algorithms and the size of ARSP on real datasets.

instances are pruned during the computation. However, this also results in the need to perform more $\mathcal{F}$-dominance tests to compute rskyline probabilities of unpruned instances. The trends of the running time on IND and ANTI reflect the compromise of these two factors. B&B perform inconsistently as its pruning strategy is more effective on IND. Fig. 11 (r)-(t) show the results under linear constraints generated by IM. Running time of all algorithms and the size of ARSP show similar trends to WR under all parameters except c. As stated above, the $\mathcal{F}$-dominance ability of each instance improves with the growth of c. Thus, as shown in Fig. 11(t), the running time of all algorithms decreases, except QDTT+. This is because the number of vertices of the preference region generated by IM increases as c grows (see the curve of V), thus leading to the dimensional disaster of the quadtree. This also accounts for the failure of QDTT+ when $d \geq 5$ in Fig. 11(s).

Experimental results on real datasets confirm the above observations. Fig. 12 (a) shows the results on IIP with varying m. As introduced in the datasets' description, each records in IIP is treated as an uncertain tuple with one instance. This implies that $\phi = 1$, i.e., every uncertain tuple T in IIP satisfies $\sum_{t \in T} p(t) < 1$. Thus, the size of ARSP is the number of input instances. And B&B nearly degenerates into LOOP, as no instances are pruned and no computations are reused. Fig. 12 (b)-(c) show the results on CAR and NBA with different m. It is noticed that attribute variance is pretty large in these two datasets. For example, in NBA, about half of the players scored zero points in some games but more than 20 points in other games. Therefore, relative performance of algorithms are similar to synthetic datasets with large l. This also holds for results on NBA with different d and c as shown in Fig. 12 (d)-(e).

5.3.2 Algorithms for the MLRS problem

Following [30], our experiments focus on the simplified uncertain model, i.e., each uncertain tuple only has one instance. Hence, parameters cnt, l, and ϕ are always 1, 0, and 0, respectively. We uniformly generate the probability of each instance in the interval $[\alpha - \delta, \alpha + \delta]$, where α is the probability center and δ represents one

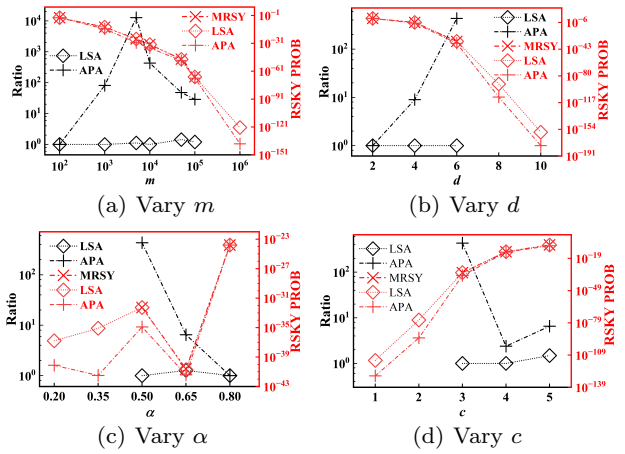


(a) IND, vary m (b) ANTI, vary m
(c) IND, vary d (d) ANTI, vary d
(e) IND, vary α (f) ANTI, vary α
(g) IND, vary c (h) ANTI, vary c

Fig. 13 Running time on synthetic datasets

side width, i.e., $\delta = \min(\alpha, 1 - \alpha)$. The default values for uncertain tuple cardinality m, data dimensionality d, probability center α of synthetic datasets are set as $m = 10^4$, $d = 6$, and $\alpha = 0.5$. Unless otherwise stated, WR is used to generate $c = d/2$ input linear constraints for $\mathcal{F}$. The running time limit (INF) is set as 12 hours.

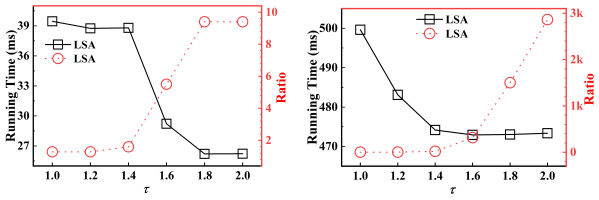
Fig. 13 (a)-(f) show the running time of different algorithms on synthetic datasets with varying m from 10^2 to 10^6, d from 2 to 10, and α from 0.2 to 0.8, respectively. Overall, the running time of all algorithms is proportional to the size of the reduced dataset (as indicated by the second y-axis). It is affected by the following factors. First, apparently, as m increases, the size



(a) Vary m (b) Vary d

(c) Vary α (d) Vary c

Fig. 14 Solution quality on synthetic datasets (IND)



(a) IND (b) ANTI

Fig. 15 Effect of τ on LSA.

of the reduced data increases. Second, the $\mathcal{F}$-dominant power of an instance drops when d grows, which can be explained by the fact that the size of rskyline increases with d. Therefore, the number of pruned instances also drops when d grows. As for α, since the probability of each instance is uniformly generated in the interval $[\alpha-\delta, \alpha+\delta]$, where $\delta = \min(\alpha, 1-\alpha)$, the smaller α is, the fewer instances meet reduction rules, and thus the fewer instances are reduced.

Next, we provide a fine-grained comparison of different algorithms. For exact algorithms, some results of DSA are omitted since it failed to obtain any solution due to excessive memory consumption. This is because the breadth-first search strategy and the coarse upper bound estimation force it to store an enormous number of candidates in the main memory. Thanks to its efficient search strategy, precise terminal condition, and effective pruning strategies, BBA outperform DSA by around two orders of magnitude and shows better scalability. However, since the time complexity of BBA is still exponential with respect to the input data size, it cannot complete within the time limit (12 hours) under certain extreme parameter settings.

For approximation algorithms, APA and LSA run faster than exact algorithms because they only find suboptimal solutions. APA runs fastest due to its simplicity. But this also leads to a poor quality of its solution as shown in Fig. 14 (a)-(c). The ratio here denotes

Table 3 Comparison between path reduction and region reduction on IND datasets with varying m ($d = 6, \alpha = 0.2$).

m	Running Time (s)		Reducing Ratio (%)	
	PATH	REGION	PATH	REGION
10^3	0.025	0.08	32	37.4
10^4	0.164	0.957	76.7	78
10^5	1.665	13.23	94.1	94.5
10^6	23.49	214.03	98.5	98.6

$\Pr_{\text{rsky}}(S^*)/\Pr_{\text{rsky}}(S)$, where S^* is the optimal solution and S is the result returned by LSA. Typically, the extra time taken by LSA for performing local search is proportional to the probability gap between the solution returned by APA and the optimal solution. Although LSA has a loose theoretical guarantee, the local search significantly improves the solution quality. In most cases, LSA returns a solution with approximation ratios less than 1.5. Fig. 15 shows the effect of the hyper-parameter τ on the running time and approximation ratio of LSA. As τ increases, its running time decreases while its approximation ratio deteriorates dramatically.

We then evaluate the effect of the number of constraints c. Fig. 13 (g)-(h) and Fig. 14(d) plot the running time and solution quality as c varies from 1 to 5. As c grows, the preference region shrinks, which improves the $\mathcal{F}$-dominant power of each instance. Thus, the running time of each algorithm decreases as c grows because the size of the reduced dataset decreases. Despite these changes, the solution quality of LSA remains close to optimal.

In summary, BBA outperforms DSA and is efficient enough to handle small-to-medium sized datasets. As for large-scale datasets, LSA further enhances time efficiency at the cost of a slight loss in solution quality. Although with a loose theoretical guarantee, LSA returns a near-optimal solution in most cases.

5.3.3 Path Reduction versus Region Reduction

We compare the efficiency and effectiveness of the path reduction and region reduction on synthetic datasets. To avoid the impact of point reduction, the probability center α is set to 0.2. This guarantees that the probability of each instance is in $[0, 0.4]$, and therefore no point reduction will be performed.

Table 3 and 4 report the running time and reducing ratio (the percentage of reduced instances) of these two reductions on IND datasets with varying m and d, respectively[7]. Overall, the reducing ratios of path

[7] Similar results can be observed on ANTI datasets, so they are omitted due to space constraints

Table 4 Comparison between path reduction and region reduction on IND datasets with varying d ($m = 10^4, \alpha = 0.2$).

d	Running Time (ms)		Reducing Ratio (%)	
	PATH	REGION	PATH	REGION
2	3.6	3.8	98.7	99.5
4	17.1	73.2	94.3	94.7
6	124.1	808.8	76.7	78
8	357.5	4192.9	56.6	56.8
10	1110.1	15939.2	20.2	20.9

reduction and region reduction keep roughly the same throughout all datasets. Region reduction prunes around 1.2% more instances than path reduction. As m and d increase, the running time increases for both path reduction and region reduction. Corresponding to the quadratic time complexity of path reduction and the cubic time complexity of region reduction, the running time of region reduction increases faster than that of path reduction with respect to m. As d grows, the advantage in time efficiency of path reduction over region reduction becomes more apparent. This is due to the following two reasons. First, region reduction performs more $\mathcal{F}$-dominance tests than path reduction, and the time required for each $\mathcal{F}$-dominance test increases linearly with d. Second, the $\mathcal{F}$-dominant power of an instance decreases with the increase of d which makes more instances are tested for reducing the data size. The decrease in the $\mathcal{F}$-dominant power also explains the decrease in the reducing ratios of these two reduction rules as d grows.

In summary, path reduction serves as an good approximation of region reduction, which improves time efficiency at the expense of a bit of loss in reducing ratio. Thus, path reduction is more suitable for small sized uncertain datasets, where nearly 90% of the running time is spent on data reduction. Additionally, path reduction can be used as a filter, i.e., region reduction is performed only if path reduction fails on an instance. This approach may reduce overall time consumption while preserving an optimal reducing ratio.

6 Related Work

In this section, we elaborate on two pieces of previous work that are most related to ours.

Uncertain skyline queries. Pei et al. first studied skyline queries on uncertain datasets [38]. They proposed two algorithms to identify uncertain tuples whose skyline probabilities exceed the input threshold p. Recognizing the inherent limitations of threshold queries, Atallah and Qi subsequently addressed the problem of computing skyline probabilities for all instances [5]. They proposed a $\tilde{O}(n^{2-1/(d+1)})$-time algorithm by combining two basic methods for all skyline probabilities computation, the weighted dominance counting method and the grid method. Building on this, Atallah et al. improved the time complexity to $\tilde{O}(n^{2-1/d})$ by substituting the grid method with a more efficient sweeping method [6]. However, these two algorithms are limited to 2D datasets due to a hidden factor that is exponential in the dimensionality of the dataset, which came from the high dimensional weighted dominance counting algorithm. To get rid of this, Afshani et al. proposed a new algorithm based on a pre-order traversal of a modified KD-tree [2]. With the well-know 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 [26]. Note that all existing work [2, 5, 6, 26] never studied the hardness of the ASP problem. Our result also gives the conditional lower bound of the ASP problem, and proves the near optimality of algorithms proposed in [2, 5, 6]. Besides, adapting these algorithms for solving the ARSP problem is non-trivial. This is because all of them rely on the dominance region of an instance is a hyper-rectangle in the data space. However, this no longer holds under $\mathcal{F}$-dominance as shown in [15, 38].

The work most relevant to the MLRS problem is the studies on the MLS problem [4, 30]. Liu et al. proved that when $d \geq 3$, the MLS problem is NP-hard [30]. They proposed a search algorithm based on dynamic programming, incorporating pruning and early termination strategies. To further improve efficiency, they also introduced 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 MLS problem when $d = 2$ based on dynamic programming [4]. They also proved that when $d \geq 3$, the MLS problem cannot be approximated within $2^{-O(m^{1-\delta})}$ in polynomial time for any $\delta > 0$, unless P = NP. Additionally, they gave a 2^m-approximation algorithm. As a further extension, Liu et al. investigated the problem of finding a set of k uncertain tuples such that the probability of being contained in the skyline is maximized [29]. We argue that the MLRS problem studied in this paper is significantly more challenging than the MLS problem due to the introduction of a user-specified function set $\mathcal{F}$. Specifically, we prove that for any $d \geq 2$, the MLRS problem is NP-hard and also hard to approximate. Moreover, our proposed algorithms differ from previous work in several key aspects. First, our exact algorithm employs a novel search strategy and advanced

pruning strategies, outperforming the adapted state-of-the-art method for the MLS problem by at least two orders of magnitude, Second, our approximation algorithm uses local search strategies to further enhance the solution quality. Note that our algorithms also provide new state-of-the-art solutions for the MLS problem.

Queries with relaxed preferences. Characterizing exact user preference is a hard problem [14, 15]. To address this issue, many queries with relaxed preferences have been introduced in recent years. Given a family $\mathcal{F}$ of monotone scoring functions, Ciaccia et al. introduced the concept of $\mathcal{F}$-dominance and two restricted skyline queries, ND and PO [13]. The ND query retrieves the set of non-$\mathcal{F}$-dominated tuples, while the PO query identifies tuples that are optimal according to at least one function in $\mathcal{F}$. They designed several linear programming based algorithms to solve these two queries when scoring functions are in the form of weighted L_p norms with a convex preference region. In their subsequent work [15], they considered a more general class of scoring functions and introduced new algorithms for solving the PO query. In [14], the same authors studied the problem of processing multi-source top-k queries, when only constraints on weights, rather than a specific scoring function, are available. Specifically, given a family $\mathcal{F}$ of linear scoring functions with weights satisfying linear constraints, this problem aims to identify tuples that are not $\mathcal{F}$-dominated by more than k tuples. They proposed an instance optimal algorithm to solve this problem, along with several optimizations to improve efficiency. Mouratidis et al. extended PO query under the top-k semantic. In the case where $\mathcal{F}$ is a family of linear scoring functions with a convex preference region Ω, they studied the problem of identifying all tuples that appear in the top-k result for at least one $\omega \in \Omega$ [34]. They first disqualified records $\mathcal{F}$-dominated by k or more others, and then determined the top-k-th in each partition of Ω among the remaining candidates. Liu et al. investigated a scenario of $\mathcal{F}$-dominance where $\mathcal{F}$ consists of $d-1$ constraints on the weight ratio of other dimensions to the user-specified reference dimension [28]. They defined the eclipse query, which retrieves the set of all non-eclipse-dominated tuples and, proposed a series of algorithms to efficiently solve this query. In addition, a unified index was proposed in [50] to solve a series of queries with relaxed preferences. The above works focused on datasets without uncertainty. Our work extends the $\mathcal{F}$-dominance based ND query to uncertain datasets. The algorithms proposed in [13–15, 28, 34] cannot be applied to our problems since the introduction of uncertainty makes the problems challenging as the number of possible worlds exponentially increases by data size.

In [33], Mouratidis et al. suggested that it might be easier for user to specify the output size m rather than the set of scoring functions $\mathcal{F}$. Accordingly, they studied the problem of determining the minimum radius ρ such that the output size of ND and PO queries under $\mathcal{F} = \{S_\omega(\cdot) \mid |\omega - \omega^*| \leq \rho\}$ is exactly m, where ω^* represents an estimate of a user's preference. The regret-minimizing set (RMS) problem aims to select a subset of tuples, with the objective that the top tuple in that subset for any possible user preference will not score far worse than the top-scorer in the entire dataset [36, 46]. Most studies of the RMS problem are not concerned with personalization. To fill this gap, in [45], Xiao et al. studied the problem of finding a set of r tuples to minimize the maximum rank-regret for all weights in a convex preference region. In some cases, user preferences may also be associated with a probability distribution, reflecting the likelihood that a specific weight matches the user's true preference. Soliman et al. studied the problem of finding the most probable top-k result under such a distribution [43]. Peng et al. studied the r-Hit problem, which aims to find a fixed size subset of tuples to maximize the probability of containing a tuple that is attractive to the user [39]. A tuple is said to be attractive, if it is the highest scoring tuple for the user Xiao et al. extended the r-Hit problem under the top-k semantics. These tasks are very different from the two problems studied in this paper.

7 Conclusion

In this paper, we study the problem of conducting rskyline queries on uncertain data. We first formalize the ARSP problem and the MLRS problem. Then, we prove that the ARSP problem cannot be solved in strongly subquadratic time unless the orthogonal vectors conjecture fails, and the MLRS problem is NP-hard. Next, we develop two efficient algorithms for solving the ARSP problem when $\mathcal{F}$ is a set of linear scoring functions subject to a set of linear constraints on weights and an algorithm with sublinear query time under ratio constraints. To solve the MLRS problem, we first design data reduction rules to reduce the input data size, and then propose exact and approximation algorithms with different search strategies. Finally, we conduct extensive experiments on real and synthetic datasets to confirm the efficacy and efficiency of our proposed algorithms.

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

References

1. Abiteboul, S., Kanellakis, P.C., Grahne, G.: On the representation and querying of sets of possible worlds. In: U. Dayal, I.L. Traiger (eds.) 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, pp. 34–48. ACM Press (1987)
2. Afshani, P., Agarwal, P.K., Arge, L., Larsen, K.G., Phillips, J.M.: (approximate) uncertain skylines. Theory Comput. Syst. **52**(3), 342–366 (2013)
3. Agarwal, P.K.: Simplex range searching and its variants: A review. A Journey Through Discrete Mathematics pp. 1–30 (2017)
4. Agrawal, A., Li, Y., Xue, J., Janardan, R.: The most-likely skyline problem for stochastic points. Comput. Geom. **88**, 101,609 (2020)
5. Atallah, M.J., Qi, Y.: Computing all skyline probabilities for uncertain data. In: J. Paredaens, J. Su (eds.) 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, pp. 279–287. ACM (2009)
6. Atallah, M.J., Qi, Y., Yuan, H.: Asymptotically efficient algorithms for skyline probabilities of uncertain data. ACM Trans. Database Syst. **36**(2), 12:1–12:28 (2011)
7. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The quickhull algorithm for convex hulls. ACM Trans. Math. Softw. **22**(4), 469–483 (1996)
8. Bedo, M., Ciaccia, P., Martinenghi, D., de Oliveira, D.: A k–skyband approach for feature selection. In: International Conference on Similarity Search and Applications, pp. 160–168. Springer (2019)
9. Bentley, J.L.: Decomposable searching problems. Inf. Process. Lett. **8**(5), 244–251 (1979)
10. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: D. Georgakopoulos, A. Buchmann (eds.) Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany, pp. 421–430. IEEE Computer Society (2001)
11. Bringmann, K.: Fine-grained complexity theory (tutorial). In: R. Niedermeier, C. Paul (eds.) 36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany, *LIPIcs*, vol. 126, pp. 4:1–4:7. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
12. Chen, R., Li, W., Zhang, Z., Bao, R., Harimoto, K., Sun, X.: Stock trading volume prediction with dual-process meta-learning. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 137–153. Springer (2022)
13. Ciaccia, P., Martinenghi, D.: Reconciling skyline and ranking queries. Proc. VLDB Endow. **10**(11), 1454–1465 (2017)
14. Ciaccia, P., Martinenghi, D.: FA + TA <fsa: Flexible score aggregation. In: A. Cuzzocrea, J. Allan, N.W. Paton, D. Srivastava, R. Agrawal, A.Z. Broder, M.J. Zaki, K.S. Candan, A. Labrinidis, A. Schuster, H. Wang (eds.) Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018, pp. 57–66. ACM (2018)
15. Ciaccia, P., Martinenghi, D.: Flexible skylines: Dominance for arbitrary sets of monotone functions. ACM Trans. Database Syst. **45**(4), 18:1–18:45 (2020)
16. Dyer, J.S., Sarin, R.K.: Measurable multiattribute value functions. Oper. Res. **27**(4), 810–822 (1979)
17. Eum, Y.S., Park, K.S., Kim, S.H.: Establishing dominance and potential optimality in multi-criteria analysis with imprecise weight and value. Comput. Oper. Res. **28**(5), 397–409 (2001)
18. Fay, S.A., Xie, J.: Probabilistic goods: A creative way of selling products and services. Mark. Sci. **27**(4), 674–690 (2008)
19. Gao, X.: Source code (2022). URL `https://github.com/gaoxy914/ARSP`
20. Gao, X.: Source code (2024). URL `https://github.com/gaoxy914/Most-likely-rskyline`
21. Gao, X., Li, J., Miao, D.: Computing all restricted skyline probabilities on uncertain datasets. In: 40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024, pp. 4773–4786. IEEE (2024)
22. Ge, T., Zdonik, S.B., Madden, S.: Top-k queries on uncertain data: on score distribution and typical answers. In: U. Çetintemel, S.B. Zdonik, D. Kossmann, N. Tatbul (eds.) Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009, pp. 375–388. ACM (2009)
23. Greenfield, J.S.: A proof for a quickhull algorithm (1990)
24. Henk, M., Richter-Gebert, J., Ziegler, G.M.: Basic properties of convex polytopes. In: Handbook of discrete and computational geometry, pp. 383–413. Chapman and Hall/CRC (2017)
25. Jamieson, K.G., Nowak, R.: Active ranking using pairwise comparisons. Advances in neural information processing systems **24** (2011)
26. Kim, D., Im, H., Park, S.: Computing exact skyline probabilities for uncertain databases. IEEE Trans. Knowl. Data Eng. **24**(12), 2113–2126 (2012)
27. Li, L., Wang, H., Li, J., Gao, H.: A survey of uncertain data management. Frontiers Comput. Sci. **14**(1), 162–190 (2020)
28. Liu, J., Xiong, L., Zhang, Q., Pei, J., Luo, J.: Eclipse: Generalizing knn and skyline. In: 37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021, pp. 972–983. IEEE (2021)
29. Liu, J., Zhang, H., Xiong, L., Li, H., Luo, J.: Finding probabilistic k-skyline sets on uncertain data. In: J. Bailey, A. Moffat, C.C. Aggarwal, M. de Rijke, R. Kumar, V. Murdock, T.K. Sellis, J.X. Yu (eds.) Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015, pp. 1511–1520. ACM (2015)
30. Liu, X., Yang, D., Ye, M., Lee, W.: U-skyline: A new skyline query for uncertain databases. IEEE Trans. Knowl. Data Eng. **25**(4), 945–960 (2013)
31. Mark, d.B., Otfried, C., Marc, v.K., Mark, O.: Computational Geometry Algorithms and Applications. Spinger (2008)
32. Meiser, S.: Point location in arrangements of hyperplanes. Inf. Comput. **106**(2), 286–303 (1993)
33. Mouratidis, K., Li, K., Tang, B.: Marrying top-k with skyline queries: Relaxing the preference input while producing output of controllable size. In: G. Li, Z. Li, S. Idreos, D. Srivastava (eds.) SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, pp. 1317–1330. ACM (2021)
34. Mouratidis, K., Tang, B.: Exact processing of uncertain top-k queries in multi-criteria settings. Proc. VLDB Endow. **11**(8), 866–879 (2018)

35. Mujumdar, A., Vaidehi, V.: Diabetes prediction using machine learning algorithms. Procedia Computer Science **165**, 292–299 (2019)
36. Nanongkai, D., Sarma, A.D., Lall, A., Lipton, R.J., Xu, J.: Regret-minimizing representative databases. Proceedings of the VLDB Endowment **3**(1-2), 1114–1124 (2010)
37. Nguyen, H.T.H., Cao, J.: Preference-based top-k representative skyline queries on uncertain databases. In: T.H. Cao, E. Lim, Z. Zhou, T.B. Ho, D.W. Cheung, H. Motoda (eds.) Advances in Knowledge Discovery and Data Mining - 19th Pacific-Asia Conference, PAKDD 2015, Ho Chi Minh City, Vietnam, May 19-22, 2015, Proceedings, Part II, *Lecture Notes in Computer Science*, vol. 9078, pp. 280–292. Springer (2015)
38. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: C. Koch, J. Gehrke, M.N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C.Y. Chan, V. Ganti, C. Kanne, W. Klas, E.J. Neuhold (eds.) Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007, pp. 15–26. ACM (2007)
39. Peng, P., Wong, R.C.: k-hit query: Top-k query with probabilistic utility function. In: T.K. Sellis, S.B. Davidson, Z.G. Ives (eds.) Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015, pp. 577–592. ACM (2015)
40. Preparata, F.P., Shamos, M.I.: Computational geometry: an introduction. Springer Science & Business Media (2012)
41. Qian, L., Gao, J., Jagadish, H.V.: Learning user preferences by adaptive pairwise comparison. Proc. VLDB Endow. **8**(11), 1322–1333 (2015)
42. Soliman, M.A., Ilyas, I.F., Chang, K.C.: Top-k query processing in uncertain databases. In: R. Chirkova, A. Dogac, M.T. Özsu, T.K. Sellis (eds.) Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007, pp. 896–905. IEEE Computer Society (2007)
43. Soliman, M.A., Ilyas, I.F., Martinenghi, D., Tagliasacchi, M.: Ranking with uncertain scoring functions: semantics and sensitivity measures. In: T.K. Sellis, R.J. Miller, A. Kementsietsidis, Y. Velegrakis (eds.) Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011, pp. 805–816. ACM (2011)
44. Wang, X., Shen, D., Yu, G.: Uncertain top-k query processing in distributed environments. Distributed Parallel Databases **34**(4), 567–589 (2016)
45. Xiao, X., Li, J.: Rank-regret minimization. In: 38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022, pp. 1848–1860. IEEE (2022)
46. Xie, M., Wong, R.C.W., Lall, A.: An experimental survey of regret minimization query and variants: bridging the best worlds between top-k query and skyline query. The VLDB Journal **29**(1), 147–175 (2020)
47. Yang, Z., Li, K., Zhou, X., Mei, J., Gao, Y.: Top k probabilistic skyline queries on uncertain data. Neurocomputing **317**, 1–14 (2018)
48. Yong, H., Lee, J., Kim, J., Hwang, S.: Skyline ranking for uncertain databases. Inf. Sci. **273**, 247–262 (2014)
49. Zhang, J., Cui, S., Xu, Y., Li, Q., Li, T.: A novel data-driven stock price trend prediction system. Expert Syst. Appl. **97**, 60–69 (2018)
50. Zhang, J., Tang, B., Yiu, M.L., Yan, X., Li, K.: T-levelindex: Towards efficient query processing in continuous preference space. In: Proceedings of the 2022 International Conference on Management of Data, pp. 2149–2162 (2022)
51. Zhang, Y., Zhang, W., Lin, X., Jiang, B., Pei, J.: Ranking uncertain sky: The probabilistic top-k skyline operator. Inf. Syst. **36**(5), 898–915 (2011)
52. Zoabi, Y., Deri-Rozov, S., Shomron, N.: Machine learning-based prediction of covid-19 diagnosis based on symptoms. npj digital medicine **4**(1), 3 (2021)

Appendix A Proofs

This appendix reports all the proofs of the claims included in the main body of the paper.

Proof of Theorem 1. We establish a fine-grained reduction from the orthogonal vectors problem to the ARSP problem. Given two sets A, B, each of n vectors in $\{0,1\}^d$, we construct an uncertain dataset $\mathcal{D}$ and a set $\mathcal{F}$ of monotone scoring functions as follows. First, for each vector $b \in B$, we construct an uncertain tuple T_b with a single instance b and $p(b) = 1$. Then, we construct an uncertain tuple T_A with n instances $\xi(a)$ and $p(\xi(a)) = \frac{1}{n}$ for all vectors $a \in A$, where $\xi(a)[i] = \frac{3}{2}$ if $a[i] = 0$ and $\xi(a)[i] = \frac{1}{2}$ if $a[i] = 1$ for $1 \leq i \leq d$. Finally, let $\mathcal{F}$ consist of d linear scoring functions $f_i(t) = t[i]$ for $1 \leq i \leq d$, which means instance t $\mathcal{F}$-dominates another instance s if and only if $t[i] \leq s[i]$ for $1 \leq i \leq d$.

We claim that for each instance $\xi(a) \in T_A$, there exists an instance b from other uncertain tuple T_b $\mathcal{F}$-dominating $\xi(a)$ if and only if a is orthogonal to b. Suppose there is a pair $(a, b) \in A \times B$ such that $a \times b = 0$, then $a[i] = 0$ or $b[i] = 0$ for $1 \leq i \leq d$. If $a[i] = 0$, then $b[i]$ can be either 0 or 1 and $\xi(a)[i] = \frac{3}{2} > b[i]$. Or if $b[i] = 0$, then $a[i]$ can be either 0 or 1 and $\xi(a)[i] \geq \frac{1}{2} > b[i]$. That is $b \prec_{\mathcal{F}} \xi(a)$. On the other side, suppose there is a pair of instances b and $\xi(a)$ such that $b \prec_{\mathcal{F}} \xi(a)$. For each $1 \leq i \leq d$, $b[i]$ is either 0 or 1 and $\xi(a)[i]$ is either $\frac{3}{2}$ and $\frac{1}{2}$. If $b[i] = 0$, then $b[i] \cdot a[i] = 0$. Or if $b[i] = 1$, then $\xi(a)[i] = \frac{3}{2}$ since $b[i] \leq \xi(a)[i]$. So $a[i] = 0$ according to the mapping $\xi(\cdot)$. Hence $a[i] \cdot b[i] = 0$. Thus we conclude that there is a pair $(a, b) \in A \times B$ such that $a \times b = 0$ if and only if there exists an instance $\xi(a) \in T_A$ with $\Pr_{\text{rsky}}(\xi(a)) = 0$. Since $\mathcal{D}$ can be constructed in $O(nd)$ time and whether such instance exists can be determined in $O(n)$ time, any $O(n^{2-\delta})$-time algorithm for all rskyline probabilities computation for some constant $\delta > 0$ would yield an algorithm for orthogonal vectors problem in $O(nd + n^{2-\delta} + n) = O(n^{2-\delta'})$ time for some constant $\delta' > 0$ when $d = \Theta(\log n)$, which contradicts the orthogonal vectors conjecture. □

Proof of Theorem 2. Since the dominance relationship is a special case of the $\mathcal{F}$-dominance relationship [13], the MLS problem is a special case of the MLRS problem. It was proved that when $d \geq 3$, the MLS problem is NP hard even if $\forall 1 \leq i \leq m : |T_i| = 1$, i.e., each uncertain tuple only has one instance [4, 30]. Hence, it is straight to derive that the MLRS problem is also NP-hard when $d \geq 3$. In what follows, we further prove that the MLRS problem is still NP-hard when $d = 2$ by establishing a polynomial-time reduction from the MLS problem with $d = 3$.

Given a 3-dimensional uncertain dataset $\mathcal{D} = (T_1 = \{t_1\}, \cdots, T_m = \{t_m\})$, we construct a 2-dimensional uncertain dataset $\mathcal{D}' = (T'_1 = \{t'_1\}, \cdots, T'_m = \{t'_m\})$ and a set of three monotone scoring functions $\mathcal{F} = \{f_1(\cdot), f_2(\cdot), f_3(\cdot)\}$ as follows. For each $i \in \{1, \cdots, m\}$, let $t'_i = (\frac{i}{m+1}, \frac{m+1-i}{m+1})$ and $p(t'_i) = p(t_i)$. For each $i \in \{1, 2, 3\}$, let $f_i(x)$ be a piece-wise function, i.e.,

$$f_i(x) = \begin{cases} t_j[i] & \text{if } \exists 1 \leq j \leq m : x = t'_j, \\ \min_{1 \leq j \leq n} t_j[i] & \text{else if } x[2] \leq 1 - x[1], \\ \max_{1 \leq j \leq n} t_j[i] & \text{otherwise.} \end{cases}$$

For each $i \in \{1, 2, 3\}$, $f_i(x)$ is monotone. Thus, we obtain a valid input of the MLRS problem.

We claim that finding a set $\mathcal{S}$ with the largest skyline probability on $\mathcal{D}$ is equivalent to finding a set $\mathcal{S}'$ with the largest rskyline probability on $\mathcal{D}'$ with respect to $\mathcal{F}$. For any two uncertain tuples $T'_p = \{t'_p\}, T'_q = \{t'_q\} \in \mathcal{D}'$, t'_p $\mathcal{F}$-dominates t'_q if and only if $f_i(t'_p) = t_p[i] \leq f_i(t'_q) = t_q[i]$ holds for all $i \in \{1, 2, 3\}$, i.e., t_p dominates t_q. Thus, for each possible world $D \sqsubseteq \mathcal{D}$, a set S is the skyline of D if and only if its corresponding set $S' = \{t'_i \mid t_i \in S\}$ is the rskyline of $D' = \{t'_i \mid t_i \in D\}$ with respect to $\mathcal{F}$. Since $p(t_i) = p(t'_i)$ for $i \in \{1, \cdots, m\}$, we have $\Pr(D) = \Pr(D')$. This means $\Pr_{\text{sky}}(S) = \Pr_{\text{rsky}}(S')$. From Definition 2, it is known that when $\mathcal{D} = (T_1 = \{t_1\}, \cdots, T_m = \{t_m\})$, for any $\mathcal{S} \subseteq \mathcal{D}$, $\Pr_{\text{rsky}}(\mathcal{S}) = \Pr_{\text{rsky}}(S)$, where $S = \{t_i \mid T_i = \{t_i\} \in \mathcal{S}\}$. We finally conclude that $\Pr_{\text{sky}}(\mathcal{S}) = \Pr_{\text{rsky}}(\mathcal{S}')$. This completes the proof. □

Proof of Theorem 5. We refine the fine-grained reduction established in the proof of Theorem 1 by restricting the format of $\mathcal{F}$ to $\{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge A \times \omega \leq b\}$. Given two sets of vectors A, B, we consturct the uncertain dataset $\mathcal{D}$ following the procedure described in the proof of Theorem 5. Then, we let $\mathcal{F}$ be the set of all linear scoring functions, i.e., there is no constraint on $\mathbb{S}^{d-1}$. Next, we claim that for any two instances t and s, t $\mathcal{F}$-dominates s if and only if $t[i] \leq s[i]$ for $1 \leq i \leq d$. If $t \prec_{\mathcal{F}} s$, then $t[i] \leq s[i]$ for $1 \leq i \leq d$ since $\omega_i \in \Omega$ where $\omega_i[i] = 1$ and $\omega_i[j] = 0$ for all $1 \leq j \neq i \leq d$. If $t[i] \leq s[i]$ for $1 \leq i \leq d$, it is known that $t \prec_{\mathcal{F}} s$ since all linear scoring functions are monotone. Thus, using the same statement in the proof of Theorem 1, we can conclude that there is no strongly subquadratic-time algorithm for the ARSP problem even if $\mathcal{F} = \{S_\omega(\cdot) \mid \omega \in \mathbb{S}^{d-1} \wedge A \times \omega \leq b\}$. □

Proof of Theorem 6. Let $t \in T_i$ be an instance with $\Pr_{\text{rsky}}(t) = 0$. Recall from the formule for rskyline probability in equation 3 that all other instances of T_i will not be affected by t. This also holds for instances of other T_j that are not $\mathcal{F}$-dominated by t. Now, suppose s is an instance of $T_{j \neq i}$ and s is $\mathcal{F}$-dominated by t. Since $t \prec_{\mathcal{F}} s$ and $\Pr_{\text{rsky}}(t) = 0$, it is easy to see that there exists a set $\mathcal{T} = \{T_k \mid k \neq j \wedge k \neq i\}$ such that all instances

of each $T_k \in \mathcal{T}$ $\mathcal{F}$-dominate t. Moreover, because $\mathcal{F}$-dominance is asymmetric, it is known that there exists at least one uncertain tuple $T_k \in \mathcal{T}$, all instances of which have non-zero rskyline probability. Therefore, according to the transitivity of $\mathcal{F}$-dominance, s is also $\mathcal{F}$-dominated by all instances of T_k and thus $\Pr_{\text{rsky}}(s) = 0$. The theorem follows. □

Proof of Theorem 7. We start with the construction of the pruning set P. For each uncertain tuple T_i with $\sum_{t\in T_i} p(t) = 1$, we insert an instance $p_i = (\max_{t\in T_i} S_{\omega_1}(t), \cdots, \max_{t\in T_i} S_{\omega_{d'}}(t))$ into P. Note that in order to facilitate the understanding of the proof, the above construction also requires to map all instances into the score space in advance. However, in the proposed algorithm, we construct P incrementally during the computation. It is straight to verify that $|P| \leq m$ from the construction of P. Then, let t denote an instance of T_i, we prove that $\Pr_{\text{rsky}}(t) = 0$ if and only if $S_V(t)$ is dominated by some $p_{j\neq i} \in P$. From equation 3, it is easy to see that $\Pr_{\text{rsky}}(t) = 0$ if and only if there exists an uncertain tuple $T_{j\neq i}$ such that every instance $s \in T_j$ $\mathcal{F}$-dominates t and $\sum_{s\in T_j} p(s) = 1$. That is $S_V(s) \preceq S_V(t)$ holds for all instances $s \in T_j$ according to Theorem 3. Moreover, since a set of instances dominates another instance if and only if the maximum corner of their minimum bounding rectangle dominates that instance, it is derived that $\Pr_{\text{rsky}}(t) = 0$ if and only if $p_j = (\max_{s\in T_j} S_{\omega_1}(s), \cdots, \max_{s\in T_j} S_{\omega_{d'}}(s)) \preceq t$. Based on the construction of P, it is known that all of them are included in P, thus completing the proof. □

Proof of Point Reduction. Let s be a tuple that is $\mathcal{F}$-dominated by t. We prove that any candidate rskyline S containing s is not the most-likely rskyline and removing all such s will not affect the rskyline probability of any most-likely rskyline. Since $t \prec_{\mathcal{F}} s$, we construct a new candidate rskyline S' by inserting t into S and removing all tuples in S that are $\mathcal{F}$-dominated by t, i.e., $S' = S \cup \{t\} \setminus \{s \in S \mid t \prec_{\mathcal{F}} s\}$. The relationship between $\Pr_{\text{rsky}}(S)$ and $\Pr_{\text{rsky}}(S')$ is

$$\frac{\Pr_{\text{rsky}}(S')}{\Pr_{\text{rsky}}(S)} = \frac{p(t)}{1-p(t)} \times \prod_{\substack{s\in S,\\ t\prec_{\mathcal{F}} s}} \frac{1}{p(t)} \times \prod_{\substack{x\in T,\\ t\prec_{\mathcal{F}} x \wedge S\not\prec_{\mathcal{F}} x}} \frac{1}{1-p(x)} \geq \frac{p(t)}{1-p(t)} > 1.$$

Then, let S^* be the most-likely rskyline. S^* must $\mathcal{F}$-dominate t or include t, otherwise inserting t into S^* will result in a strictly better candidate rskyline for the same reason stated above. Since $S^* \prec_{\mathcal{F}} t$ or $t \in S^*$ whether tuples $\mathcal{F}$-dominated by t appear in the dataset will not affect $\Pr_{\text{rsky}}(S^*)$ according to formula (7). □

Proof of Region Reduction. Let x be a tuple that is $\mathcal{F}$-dominated by s and S be a candidate rskyline containing x. We construct a new candidate rskyline S' by inserting t into S. Since $t \prec_{\mathcal{F}} s \prec_{\mathcal{F}} x$, the relationship between $\Pr_{\text{rsky}}(S)$ and $\Pr_{\text{rsky}}(S')$ is

$$\frac{\Pr_{\text{rsky}}(S')}{\Pr_{\text{rsky}}(S)} = \frac{p(t)}{1-p(t)} \times \prod_{\substack{s\in S,\\ t\prec_{\mathcal{F}} s}} \frac{1}{p(t)} \times \prod_{\substack{x\in T,\\ t\prec_{\mathcal{F}} x \wedge S\not\prec_{\mathcal{F}} x}} \frac{1}{1-p(x)} \geq \frac{p(t)}{(1-p(t))(1-p(x))} \prod_{y\in T, t\prec_{\mathcal{F}} y\prec_{\mathcal{F}} x} (1-p(y))^{-1} \geq \frac{p(t)}{(1-p(t))(1-p(s))} \prod_{\substack{y\in T,\\ t\prec_{\mathcal{F}} y\prec_{\mathcal{F}} s}} (1-p(s))^{-1} > 1.$$

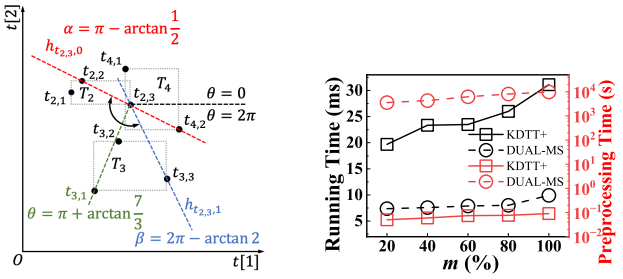
Then, let S^* be the most-likely rskyline. S^* must $\mathcal{F}$-dominate s or include s, otherwise inserting t into S^* will result in a strictly better candidate rskyline since

$$p(t) > (1-p(t)) \times (1-p(s)) \times \prod_{x\in T, t\prec_{\mathcal{F}} x\prec_{\mathcal{F}} s} (1-p(x)).$$

Since $S^* \prec_{\mathcal{F}} s$ or $s \in S^*$ whether tuples $\mathcal{F}$-dominated by s appear in the dataset will not affect $\Pr_{\text{rsky}}(S^*)$. □

Proof of Path Reduction. According to formula (8) and (9), it is easy to see that region reduction considers all instances in $N_{out}(t) \cap N_{in}(s)$ while path reduction only considers instances on a path from t to s. That is $Int(p(t,s)) \subseteq N_{out}(t) \cap N_{in}(s)$. Thus, path reduction is a sufficient condition for region reduction. This completes the proof. □

Appendix B Experiments under Ratio Constraints



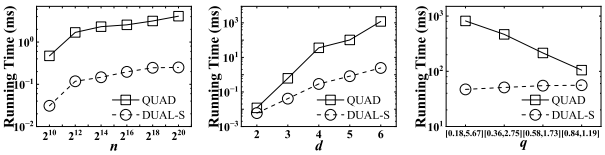
(a) Specialized version (b) Running time on IIP

Fig. 16 A specialized version of DUAL-MS for $d = 2$ and its running time on IIP dataset.

Since the data structure stated in Theorem 10 is theoretical, we introduce a specialized version of DUAL-MS for $d = 2$ to avoid this. Recall that for each instance t, we reduce the computation of $\Pr_{\text{rsky}}(t)$ to 2^{d-1} half-space reporting problems in Section 3.4. When $d = 2$, we observe that these two half-space queries can be reinterpreted as a continuous range query. Fig. 16(a)

provides an illustration. For example, when processing $t_{2,3}$, we can treat $t_{2,3}$ as the origin, and the ray $y = t_{2,3}[2], x \geq t_{2,3}[1]$ as the base. Each instance can be represented by an angle, such as $\theta = \pi + \arctan\frac{12-5}{9-6}$ for $t_{3,1}$. In this case, the two query halfspaces $h_{t_{2,3},0}$: $t[2] \leq -0.5t[1] + 16.5$ and $h_{t_{2,3},1} : t[2] \leq -2t[1] + 30$ can be transformed into a range query $[\pi - \arctan\frac{1}{2}, 2\pi - \arctan 2]$ with respect to the angle. After this transformation, we can simply use a binary search tree to organize the instances, instead of the point location tree. We implemented this specialized DUAL-MS and evaluate its performance on IIP. For comparison, we attach a simple preprocessing strategy to KDTT+, which removes all instances with zero skyline probability from I. Fig. 16(b) shows the running time of these two algorithms. Although the query efficiency is improved, the substantial preprocessing time and memory consumption prevents its application on big datasets.

The above drawbacks of DUAL-MS are alleviated when it comes to process eclipse queries. This is because eclipse is always a subset of skyline S, which has a logarithmic size in expectation. Meanwhile, the multi-level strategy is no longer needed since for each tuple $t \in S$, t belongs to the eclipse of D iff all point location queries on $S^*_{t,k}$ $(0 \leq k < 2^{d-1})$ return emptiness. We implement the DUAL-S for eclipse query processing, in which we use a kd-tree to index the dataset constructed by performing shifted strategy. For comparison, we also implement the state-of-the-art index-based algorithm QUAD [28] in C++ and compare their efficiency and scalability with respect to data cardinality n, data dimensionality d, and ratio range q. Similar to [28], the defaulted value is set as $n = 2^{14}$, $d = 3$, and $q = [0.36, 2.75]$.



(a) Effect of n (b) Effect of d (c) Effect of q

Fig. 17 Running time for eclipse query (IND).

As shown in Fig. 17 (a)-(b), the running time of these two algorithms increases as n and d grows. DUAL-S outperforms QUAD by at least an order of magnitude and even more on high-dimensional datasets. The reason is that QUAD needs to iterate over the set of hyperplanes returned by the window query performed on its Intersection Index, and then reports all objects with zero order vector as the final result. This takes $O(s^2)$ time, where s denotes the skyline size of the dataset. But DUAL-S excludes an object from the result if there is a query returns non-empty result, which only take $O(s)$ time. Moreover, the hyperplane quadtree adopted in QUAD scales poorly with respect to d for the following two reasons. On the one hand, the tree index has a large fan-out since it splits all dimensions at each internal node. On the other hand, the number of intersection hyperplanes of a node decreases slightly relative to its parent, especially on high-dimensional datasets, which results in an unacceptable tree height. Moreover, as shown in Fig. 17(c), QUAD is more sensitive to the ratio range than DUAL-S because the number of hyperplanes returned by the window query actually determines the running time.