U-Skyline: A New Skyline Query for Uncertain Databases

Xingjie Liu, De-Nian Yang, *Senior Member*, *IEEE*, Mao Ye, and Wang-Chien Lee, *Member*, *IEEE*

Abstract—The skyline query, aiming at identifying a set of skyline tuples that are not dominated by any other tuple, is particularly useful for multicriteria data analysis and decision making. For uncertain databases, a probabilistic skyline query, called P-Skyline, has been developed to return skyline tuples by specifying a probability threshold. However, the answer obtained via a P-Skyline query usually includes skyline tuples undesirably dominating each other when a small threshold is specified; or it may contain much fewer skyline tuples if a larger threshold is employed. To address this concern, we propose a new uncertain skyline query, called *U-Skyline query*, in this paper. Instead of setting a probabilistic threshold to qualify each skyline tuple independently, the U-Skyline query searches for a set of tuples that has the highest probability (aggregated from all possible scenarios) as the skyline answer. In order to answer U-Skyline queries efficiently, we propose a number of optimization techniques for query processing, including 1) computational simplification of U-Skyline probability, 2) pruning of unqualified candidate skylines and early termination of query processing, 3) reduction of the input data set, and 4) partition and conquest of the reduced data set. We perform a comprehensive performance evaluation on our algorithm and an alternative approach that formulates the U-Skyline processing problem by integer programming. Experimental results demonstrate that our algorithm is 10-100 times faster than using CPLEX, a parallel integer programming solver, to answer the U-Skyline query.

Index Terms—Skyline query, uncertain databases, query processing

1 Introduction

A skyline query is a powerful tool for multicriteria data analysis, data mining, and decision making. Given a set of data tuples with multiple attributes, a skyline query retrieves a set of data tuples, called *skyline tuples*, to form a *skyline*. These skyline tuples are not dominated by any other tuples. Here a tuple p is said to dominate another tuple q if p is not worse than q on all attributes and p is strictly better than q on at least one attribute. The skyline tuples are considered to be important because they exhibit the following properties:

- **Nondominance.** Skyline tuples are not dominated by any tuple outside the skyline set.
- **Incomparability.** Skyline tuples do not dominate each other, i.e., they hold on to their own ground/importance in skyline against each other.
- **Coverage.** All together, the skyline tuples dominate all the nonskyline tuples, i.e., each nonskyline tuple is dominated by *at least one* skyline tuple.

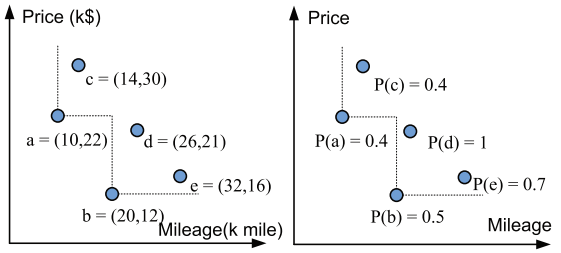
- *X. Liu is with the Pennsylvania State University, 224 Flynn Ave, Mountain View, CA 94043. E-mail: xzl106@cse.psu.edu.*
- *D.-N. Yang is with the Institute of Information Science and Research Center for Information Technology Innovation, Academia Sinica, No 128, Academia Road, Section 2, Nankang, Taipei 11529, Taiwan. E-mail: dnyang@iis.sinica.edu.tw.*
- *M. Ye is with Klout, 77 Stillman St. San Francisco, CA 94107. E-mail: mxy177@cse.psu.edu.*
- *W.-C. Lee is with the Department of Computer Science and Engineering, Pennsylvania State University, 360D Information Sciences and Technology Building, University Park, PA 16802. E-mail: wclee@cse.psu.edu.*

Manuscript received 17 May 2011; revised 24 Oct. 2011; accepted 20 Dec. 2011; published online 13 Feb. 2012.
Recommended for acceptance by K.-L. Tan.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2011-05-0270.
Digital Object Identifier no. 10.1109/TKDE.2012.33.

The following example illustrates the skyline query. Suppose a user John wants to buy a used car and graphs the vehicle data set in Fig. 1a in terms of price and driven mileage. From the figure, we can see that vehicle a and b are the best choices because other vehicles are all inferior to either a or b in at least one attribute and not better than them in all the other attributes (e.g., d has a higher price and a higher mileage than b and, thus, is inferior to b). Therefore, the answer for the skyline query upon this data set is $\{a, b\}$, which forms a *skyline* that dominates the rest of vehicles.

After its introduction in [10], the skyline query has received significant attention from the database community. Many variants of the skyline query and challenging research issues have been studied [9], [15], [23], [24], [29]. A recent development of this research direction is to support skyline queries over uncertain databases [30], [40], [8], [41], [25]. This is a vital research topic with many potential real-life applications of coming future. For example, vehicles in bidding services (e.g., www.motors.ebay.com) may not always be available to the users for sure during the decision making process of a potential buyer. Similarly, the availability and pricing of air tickets show a lot of volatility. As a result, travel web services such as www.bing.com/travel have started to provide the probabilities of ticket prices going up or down to assist decision making of their users. Generally, data uncertainty arises inherently from various causes, such as incomplete survey results, data measure and collection methods, statistical and data mining techniques, and so on. Therefore, in many real-life applications, databases contain incomplete, outdated, noisy, and uncertain data. In this paper, we aim to study the problem of supporting the skyline query over uncertain databases.

Our study is based on the possible worlds semantics [4], a probabilistic model of data uncertainty widely adopted in



(a) Data Tuples (b) Tuple Probabilities

Fig. 1. Skyline example.

the literature of uncertain databases [6], [12], [20], [28], [32]. A possible world represents one of the possible scenarios derivable from the entire uncertain database. Consider our used car example in Fig. 1. Since each vehicle may or may not be available for the user, there are $2^5 = 32$ possible worlds, each of which consists of a unique combination of vehicles available in that scenario. Additionally, each world is associated with a possible world probability, indicating the probability for this scenario to occur. For example, the possible world $\{a, b, d\}$ has its probability $P(\{a, b, d\}) = P(a) \cdot P(b) \cdot \overline{P(c)} \cdot P(d) \cdot \overline{P(e)} = 0.036$.[1] With the possible worlds semantic, we can evaluate the skyline query under each possible world and aggregate the results from different possible worlds to obtain probabilistic query answers.

The first study on supporting the skyline query over uncertain databases, called P-Skyline, has been reported in [30]. This pioneering work inspires a number of follow-up studies [40], [8], [41], [25], as well as our research work presented in this paper. In P-Skyline, the authors define a nondominance probability for each tuple by aggregating over all the possible worlds within which the tuple is not dominated. For example, consider the uncertain used car data set in Fig. 1b, where each vehicle is associated with its availability probability. Based on [30], $P_{\text{non-dom}}(d) = \overline{P(b)} \cdot P(d) = 0.5$, because d is not dominated only in those possible worlds in which b does not appear. Similarly, $P_{\text{non-dom}}(b) = P(b) = 0.5$ because no other tuple can dominate it. Accordingly, P-Skyline returns individual data tuples with nondominance probabilities greater than or equal to a specified threshold. In this example, given a threshold 0.5, the P-Skyline answer set is $\{b, d\}$.[2] Notice that after each tuple's nondominance probability is obtained, the interplay/relationship among tuples are no longer considered in the answer filtering step. Thus, it is quite common that the P-Skyline answer set does not form a skyline in any possible world after all (e.g., $\{b, d\}$ is not a valid skyline in any possible world).

In this paper, we propose a new uncertain skyline operator, namely, *U-Skyline*, that aims to return an uncertain skyline answer set from a different but complementary perspective to P-Skyline. Instead of considering individual tuples, U-Skyline focuses on returning an answer set that forms a valid skyline with the maximum probability. The probability associated with a skyline answer set to be valid among all the possible worlds is, thus, termed as U-Skyline probability. Recall the same example in Fig. 1b. The U-Skyline probability for $\{a, b\}$ is $P_{\text{u-sky}}(\{a, b\}) = P(a) \cdot P(b) = 0.2$, because both tuples are not dominated by others, i.e., they are the skyline in all the possible worlds that have these two tuples. Similarly, $P_{\text{u-sky}}(\{b\}) = P(b) \cdot \overline{P(c)} \cdot \overline{P(a)} = 0.12$ because $\{b\}$ is a valid skyline only when both a and c are not in the possible world. Notice that, $P_{\text{u-sky}}(\{b\})$ is different from $P_{\text{non-dom}}(b)$ in P-Skyline, because the former considers the global dominance among all data tuples, while the later focuses on only the local tuple b. Also, by comparing U-Skyline probabilities of all candidate skyline answer sets, the U-Skyline with the maximum probability in our example is $\{a, b\}$.

Compared with the P-Skyline probability for a tuple, the U-Skyline probability of a tuple set is relatively small. However, the probability of any P-Skyline set that considers multiple skyline tuples and follows incomparability may also be small because the probability in this case corresponds to a set, instead of a tuple. In other words, because the sample space of U-Skyline probability is exponentially larger than the sample space of P-Skyline probability, the U-Skyline probability of an answer set is naturally smaller than the P-Skyline probability defined based on a tuple. We would like to emphasize that 1) the probabilities in P-Skyline and 2) U-Skyline represent different conceptual notions, and the two skyline variants are complement to each other. In fact, the U-Skyline probability is similar to the likelihood probability [5] in the maximum likelihood estimation problem, which is widely applied in areas of data mining and statistics [18], [19]. Although the U-Skyline probability is small due to the large sample space, the U-Skyline provides insights for a set of tuples that most likely to be the skyline for the uncertain data set.

Here, we further compare U-Skyline and P-Skyline in terms of the properties of *nondominance, incompatibility, and coverage*. Strictly speaking, both P-Skyline and U-Skyline do not possess the nondominance property exhibited in conventional database since they aim to return skyline tuples with threshold-qualified and highest probabilities, respectively. In other words, there is a chance for some nonreturned skyline tuples to dominate the returned tuples within some possible worlds. This is inherented from the nature of probabilistic queries for uncertain databases. Therefore, here we reconsider the nondominance property for uncertain database as "it's not likely for returned skyline tuples to be dominated by nonanswer tuples." Both P-Skyline and U-Skyline have exhibited this property by ensuring the probability for the undesired scenario to be insignificant. For P-Skyline, by returning those tuples that are dominated by none or a few low-probability tuples as the answer set, the probability that a P-Skyline answer set gets dominated by other nonanswer tuples is small. For U-Skyline, on the other hand, if a candidate skyline is dominated by other tuples, its U-Skyline probability is small and, thus, not likely to be the answer.

P-Skyline does not guarantee incomparability because tuples are selected individually after their nondominance probabilities are obtained. For example, incomparability does not hold in the P-Skyline answer, i.e., $\{b, d\}$, for data tuples in Fig. 1. Additional steps may be taken to determine which tuple in the answer set is superior. Nevertheless, even if the P-Skyline answer is postprocessed, the final answer may not be ideal since the postprocessing operation

1. $\overline{P(x)} = 1 - P(x)$ here and in the rest of this paper.
2. $P_{\text{non-dom}}(a) = 0.4$, $P_{\text{non-dom}}(c) = 0.24$ and $P_{\text{non-dom}}(e) = 0.35$.

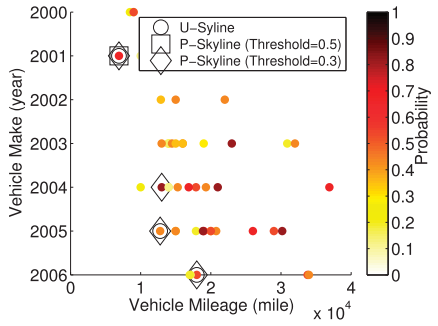


Fig. 2. Comparison of U-Skyline and P-Skyline.

does not take into account the data probabilities at all. In addition, trying various probability thresholds in P-Skyline may be tedious for users. On the other hand, every U-Skyline answer always satisfies the incomparability property because the skyline returned is required to hold in some possible worlds.

Next, we use our running example to compare the coverage property. As shown in Fig. 1, P-Skyline considers tuple a and tuple c to be unimportant since their nondominance probabilities are lower than the threshold 0.5. It seems to imply that no car with a high price but low mileage is likely to be available. This is not true since the probability for both a and c to be not available is merely 0.36. Setting a smaller threshold for P-Skyline can improve the coverage. However, the incomparability will deteriorate in this case. Additionally, users usually are not aware of the most proper threshold value. On the other hand, U-Skyline considers the global relationship among tuples and thus factors in the coverage property by design. Since a candidate skyline (e.g., $\{b\}$) not dominating a and c is not a skyline for those possible worlds having a or c, there is a probability penalty factor of $(\overline{P(a)} \cdot \overline{P(c)} = 0.36)$ for these candidate skylines. Therefore, compared to a's probability $P(a) = 0.4$, a skyline containing a has a higher U-Skyline probability than a candidate excluding a and c. Finally, we observe that there is a tradeoff between incomparability and coverage in P-Skyline. To satisfy incomparability, we need to increase the threshold. However, to secure coverage, threshold must be lowered. U-Skyline does not have this tradeoff, and our experiment later (see Fig. 9 in Section 8) demonstrates that U-Skyline strikes a good balance in this tradeoff. When both P-Skyline and U-Skyline have the same answer set size (via adjusting P-Skyline's threshold), U-Skyline's answer provides a better coverage (Having a lower probability that a nondominated tuple is not included in the solution.).

Fig. 2 shows an example that uses real data to compare P-Skyline with U-Skyline. A vehicle data set [1] is employed to extract the uncertain data set (see Section 8.2 for details). Here, mileage and make year are used as the skyline criteria. As shown, the lower left vehicles dominate the upper right vehicles. Also, the marker color denotes the available probability. When the P-Skyline threshold is chosen as 0.5, only one upper left vehicle is selected, ending up with a biased result. Lowering the P-Skyline threshold to 0.3 addresses this coverage problem but the incomparability issue arises because the answer set contains one vehicle made in 2005 that dominates another vehicle made in 2004. Notice that the U-Skyline addresses both the coverage and incomparability issues well. In addition, trying various probability thresholds in P-Skyline may be tedious for users. On the contrary, U-Skyline does not require a threshold and always return the skyline result with the maximum probability.

Although U-Skyline has many good properties, U-Skyline is required to evaluate all data tuples as a whole instead of each individual tuple. Therefore, efficiently processing U-Skyline is very critical. In addition, sophisticated analysis of the probability formula and development of pruning techniques of the search space are mandates. To meet these challenging needs, we made the following contributions in this paper:

- We propose a new skyline query (U-Skyline) for uncertain data. It focuses on meeting the nondominance, incomparability and coverage properties simultaneously for uncertain skyline query.
- We prove that U-Skyline query processing is NP-hard.
- We investigate the interplay among different data tuples during the computation, and transform U-Skyline query processing into an integer programming problem.
- We design a search algorithm based on dynamic programming (DP) to find U-Skyline. We also improve the algorithm with pruning and early termination (P&ET) techniques.
- We propose input data set reduction (SR) and partition (SP) techniques to reduce the input data set size in order to further expedite the U-Skyline processing time.
- We conduct a comprehensive evaluation on the proposed U-Skyline query. Our experimental results demonstrate that the proposed algorithm is much faster than using a parallel integer programming solver to obtain U-Skyline.

The rest of this paper is organized as follows, (see Fig. 3 for an overview of the proposed techniques for U-Skyline query processing. Section 2 offers preliminary analysis, notation definition, literature review, and most importantly, the U-Skyline definition. Section 3, analyzes the property of the U-Skyline probability, which allows us to formulate the U-Skyline query as an integer programming problem with the optimization objective function and constraints provided. In Section 4, we discover several important U-Skyline properties. A dynamic programming search algorithm is proposed and enhanced with candidate pruning and early termination. Section 5 presents the input data set reduction and partition techniques, which significantly reduce the problem scale by cutting down input data set. Section 6 extends the U-Skyline processing algorithm with an approximation heuristic and presents our extension for handling uncertain tuples with multiple instances. Section 7 reports the experimental results and findings for performance evaluation. Finally, Section 8 concludes the paper.

2 Preliminary

In this section, we first review the definition of dominance relationship and conventional skyline queries, as well as the

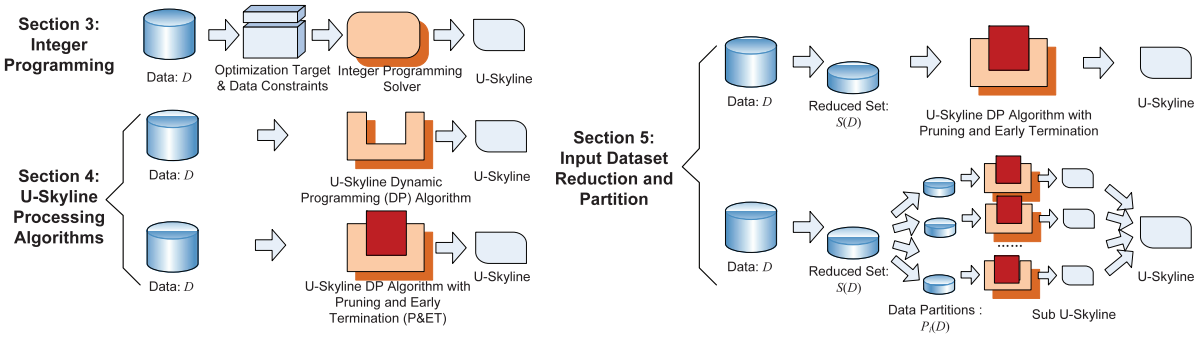


Fig. 3. Overview of the proposed techniques for U-Skyline query processing.

possible worlds semantics in uncertain database. Then, we formally define the U-Skyline query and finally review some relevant studies on skyline queries in both certain and uncertain databases.

2.1 Skyline Query

A skyline query retrieves tuples that are *not dominated* by any other tuples in a multidimensional data set. Therefore, we first define the dominance relationship.

Definition 1 (Dominance Relationship). *Given a data set $D = \{t_1, t_2, \ldots, t_N\}$ with each tuple t_i consisting of d-dimensional attributes, denoted as $\{t_i.a_1, t_i.a_2, \ldots, t_i.a_d\}$. A tuple t_i dominates t_j (denoted as $t_i \prec t_j$ or $t_j \succ t_i$) when*

1. $\forall x, (1 \leq x \leq d)$, $t_i.a_x \leq t_j.a_x$.
2. $\exists y, (1 \leq y \leq d)$, $t_i.a_y < t_j.a_y$.

In other words, t_i dominating t_j means t_i is no worse than t_j in all attributes, and strictly better than t_j in at least one attribute. The dominance relationship offers a way to compare any two tuples, and the comparison result between two tuples t_i and t_j are either $t_i \prec t_j$, $t_i \succ t_j$, or $t_i \not\prec t_j \wedge t_i \not\succ t_j$. Note that the third case implies t_i and t_j not dominating each other. Furthermore, we denote the dominance relationship between a tuple and a tuple set as follows:

- $S_i \prec t_j$ means $\exists t_x \in S_i, t_x \prec t_j$.
- $S_i \not\prec t_j$ means $\nexists t_x \in S_i, t_x \prec t_j$.

The skyline for a certain data set D is defined as:

Definition 2 (Skyline). *The skyline of a certain multidimensional data set D, denoted as $\mathrm{SKY}(D)$, is a subset of tuples that are not dominated by any other tuples in D:*

$$\mathrm{SKY}(D) = \{t_i | t_i \in D, \forall t_j \in D, t_j \not\prec t_i\}. \quad (1)$$

2.2 Uncertain Data Model

In relational databases, tuples refer to data records. In order to represent data uncertainty, previous research works incorporate a probability for each tuple. In this paper, we mainly focus on the independent probability model [6], [33], i.e., each tuple's probability is independent to any others. Nevertheless, the U-Skyline operator can be generalized to handle other probability models. The processing techniques presented in this paper can also be extended accordingly.

We use $D = \{t_1, t_2, \ldots, t_N\}$ to represent the entire data set of N tuples. Each tuple t_i has a corresponding *attendance probability* $P(t_i)$, which is probability that the object or event represented by this tuple does occur, e.g., a particular used car is available. Similar to other related work, we adopt the possible worlds semantics [4] to explain the uncertainty brought by the attendance probability.

Definition 3 (Possible World). *Given an uncertain data set D, a possible world W is a subset of D that represents a possible scenario being captured by D. The probability that W exists, denoted as $P(W)$, is called its possible world probability.*

In this paper, since we assume all the tuples' attendance probabilities are independent, the possible world probability of W is as follows:

$$P(W) = \prod_{\forall t_i \in W} P(t_i) \cdot \prod_{\forall t_j \notin W} \overline{P(t_j)}. \quad (2)$$

By collecting all possible worlds, the entire possible world set is represented as $\mathcal{PW} = \{W_1, W_2, \ldots, W_M\}$, where M is the number of possible worlds.

The possible worlds semantics provides a nice aspect to view uncertain data. By considering each possible world W_i as a certain database instance, conventional queries for certain database can be applied. Thus, the query result under the corresponding possible world is valid with probability $P(W_i)$. As a result, many certain queries are applicable to uncertain databases by aggregating query results under different possible worlds. However, it is worth noting that this simple query processing strategy based on possible worlds semantics does not scale up because the number of possible worlds exponentially increases by data set size N.

2.3 U-Skyline Query

Following the possible worlds semantics, we can formally define the U-Skyline query with the U-Skyline probability.

Definition 4. ***The U-Skyline Probability*** *of a candidate skyline S is the accumulated possible world probabilities of all possible worlds that have S as their skyline*

$$P_{\text{u-sky}}(S) = \sum_{\forall W \in \mathcal{PW}, S = \mathrm{SKY}(W)} P(W). \quad (3)$$

Note that this U-Skyline probability is quite different from the nondominated probability[3] in P-Skyline [30]. The U-Skyline probability is defined for a tuple set, while the

3. Also called the skyline probability in some literature.

nondominated probability, as shown below, is defined for each individual tuple

$$P_{\text{non-dom}}(t_i) = \sum_{\forall W \in \mathcal{PW}, \exists t_x \in W, t_x \prec t_i} P(W).$$

Therefore, a U-Skyline query, aiming to find a tuple set S^* that has the highest U-Skyline probability among all other tuple sets, is defined as follows:

Definition 5. ***The U-Skyline Query*** *retrieves a tuple set from D, so that its U-Skyline probability is maximized*

$$\text{U-SKY}(D) = S^* = \arg\max_{S_i \subseteq D} P_{\text{u-sky}}(S_i). \quad (4)$$

2.4 Literature Review

A skyline query for certain database has been studied extensively in the literature [10], [35], [15], [9], [23], [29], [24], [37], [36]. Most of the previous research focuses on how to answer the query in a computationally efficient way. While [10], [15], [9] focus on skyline computation on the fly, many approaches utilize index structures such as R-tree [23], R*-tree [29], and ZB-tree [24]. Lin et al. [26] extend the skyline query to find top-k representative skylines, aiming to extract the tuples that are more important to users. Because all of these approaches are geared toward certain data sets, we cannot simply apply these techniques directly to uncertain databases.

In contrast with these certain data sets, uncertain data and queries have recently garnered considerable research attention. Related research issues include uncertain data modeling [4], [6], [31], [33], data management [28], query semantics and processing [17], [12], [32], [20], indexing techniques [14], [36], and systems issues [7], [21], [11], [13]. A possible worlds semantics is provided in [4], and is adopted in most studies in uncertain databases. For uncertain queries, there are a lot of studies that tackle on the uncertain Top-k query, including *U-Topk* and *U-kRanks* [34], *PT-Topk* [20], *PK-Topk* [22], and *Expected-kRanks* [16]. In general, the U-Top$k$ and U-kRanks investigate the global appearance probabilities of the query results, PT-Top$k$ and PK-Top$k$ evaluate each tuple's Top-k probability as a qualification, and Expected-kRanks focuses on the averaged Top-k rank within the global possible worlds. Each query semantics is applicable and useful for some applications, and various processing techniques and extensions [39], [27], [38] are studied.

Uncertain skyline queries have been studied in several recent works [30], [40], [41], [8] mainly based on the P-Skyline semantics [30]. The P-Skyline semantics is similar to PT-Top$k$, where individual data tuples are evaluated based on their probabilities to participate in any skyline. Once a tuple's nondominated probability (a.k.a. skyline probability) is above a threshold, the tuple is included in the result. The P-Skyline query semantics was adopted in [40], which assumes tuple level uncertainty and no x-relationship. The P-Skyline processing time has been recently improved to subquadratic in [8]. Moreover, P-Skyline in data stream are studied in [41] and sliding windows scenarios, which decomposes the skyline probability into probability terms for tuples that arrive before or later. With this decomposition, the authors discover that only a small subset of data tuples shall be kept to answer the query. Other than this work, a reversed skyline is studied in [25], which examines the problem of identifying reference points with the same skyline result.

Notice that these previous studies are based on P-Skyline. In this paper, we propose an alternative skyline operator for uncertain databases, namely, U-Skyline. The U-Skyline is not yet another query semantics, but a novel query operator designed to maintain the nice properties of conventional skyline queries, including *nondominance*, *incomparability*, and *coverage*.

3 U-Skyline Probability and Integer Programming Model

In this section, we investigate the U-Skyline query probabilities. Through a close analysis (see Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2012.33), we found the problem of U-Skyline query processing is NP-Hard. Thus, we formulate the U-Skyline query processing problem into an integer programming problem.

3.1 U-Skyline Probability

U-Skyline aims to find a tuple set with the highest U-Skyline probability. Therefore, simplifying the U-Skyline probability computation is important. From (3), the U-Skyline probability for a candidate skyline S is obtained by enumerating all possible worlds. This brute force strategy is infeasible due to the exponential running time complexity $O(2^N)$. An idea to improve the U-Skyline probability computation is to combine the probability computation for multiple possible worlds that share the same skyline sets. The U-Skyline probability function (3) can be simplified as

$$P_{\text{u-sky}}(S) = \prod_{\forall t_i \in S} P(t_i) \cdot \prod_{\forall t_j \notin S, S \not\prec t_j} \overline{P(t_j)} \cdot \prod_{\forall t_k \notin S, S \prec t_k} 1, \quad (5)$$

where $\prod_{t_i \in S} P(t_i)$ presents the probability that all tuples in S appear, $\prod_{\forall t_j, S \not\prec t_j} \overline{P(t_j)}$ denotes the probability that all tuples *not* dominated by S disappear, and $\prod_{\forall t_k, S \prec t_k} 1$ presents the part for all the other tuples that are dominated by S. For tuples dominating some tuples in S or not dominated by any tuples in S, they must not appear due to the skyline definition. Every tuple must be either a skyline tuple or get dominated by a skyline tuple [10]. Because the skyline must contain *all* the nondominated tuples, it is quite different to the nondominated probability defined for a single tuple. For the tuples dominated by S, whether they appear or disappear does not affect the validity of S. Thus, their corresponding terms are always one.

Based on (5), we find that each tuple $t_i \in D$ is now associated with a probability value: $P(t_i)$, $\overline{P(t_i)}$, or 1. Given a candidate skyline subset S, the corresponding probability assignment of t_i is then determined. For simplicity, we denote the probability assignment of tuple $t_i \in D$ for a given skyline set S as $\texttt{PT}(t_i|S)$, called the *probability term* (PT). Therefore, (5) can be further transformed as $P_{\text{u-sky}}(S) = \prod_{t_i \in D} \texttt{PT}(t_i|S)$, described as follows:

$$\mathtt{PT}(t_i|S) = \begin{cases} P(t_i), & \text{if } t_i \in S, \\ \overline{P(t_i)}, & \text{if } t_i \notin S \text{ and } S \not\prec t_i, \\ 1, & \text{if } t_i \notin S \text{ and } S \prec t_i. \end{cases} \quad (6)$$

3.2 Integer Programming Formulation

With the simplified U-Skyline probability computation, we formulate the U-Skyline query as an optimization problem. Here, we derive two integer programming formulations for the U-skyline query. With these formulations, any integer programming solver can be applied to find U-Skyline. To do this, we introduce three binary variables σ_i, ω_i, and δ_i. For each tuple t_i, they represent whether t_i is within candidate skyline $S(\sigma_i = 1)$, disappears from the world $(\omega_i = 1)$, or is dominated by $S(\delta_i = 1)$. Since the optimization objective is to maximize the U-Skyline probability, our objective function can be simplified according to (6):

$$\begin{aligned} P_{\text{u-sky}}(S) &\Leftrightarrow \log(P_{\text{u-sky}}(S)) \\ &\Leftrightarrow \sum_{t_i \in D} (\sigma_i \cdot \log P(t_i) + \omega_i \cdot \log \overline{P(t_i)} + \delta_i \cdot 0). \end{aligned} \quad (7)$$

Equation (7) still have three terms, and the objective function minimizes the additive costs $|\log P(t_i)|$ and $|\log \overline{P(t_i)}|$ associated with σ_i and ω_i, respectively. On the other hand, the optimization constraints of the U-skyline query are

$$\sigma_i + \omega_i + \delta_i = 1, \qquad \forall\, t_i \in D, \quad (8a)$$

$$\delta_i \leq \Sigma_{\forall t_j \prec t_i} \sigma_j, \qquad \forall\, t_i \in D, \quad (8b)$$

$$\sigma_i \leq \omega_j, \qquad \forall\, t_i \in D, \forall t_j \prec t_i. \quad (8c)$$

The first constraint (8a) guarantees that each tuple t_i must be one of the three choices. Since $\delta_i = 1$ incurs no cost in the objective function, the optimization process is inclined to assign tuples with $\delta_i = 1$. Note that we should not ignore the last term in (7). Also, the term δ_i cannot be omitted in constraint (8a). If we remove δ_i, constraint (8a) becomes $\sigma_i + \omega_i = 1$. In other words, either σ_i or ω_i must be 1. This constraint is then incorrect because it enforces that every tuple t_i must be either in skyline set S or disappearing in the world. Yet it does not allow tuple t_i to be dominated by a skyline tuple. The solution is not correct because every tuple is not dominated by any other tuple. As a result, this integer programming formulation is improper. In addition, constraint (8b) enforces that δ_i can only be 1 unless *at least* one tuple t_j dominating t_i is a skyline tuple. From the perspective of the U-Skyline probability, this means once t_i is dominated by a skyline tuple t_j, its appearance does not affect the U-Skyline probability. Finally, constraint (8c) guarantees that for every skyline tuple t_i, other tuples dominating t_i must disappear from the world, ensuring that every skyline tuple is not dominated by any other skyline tuple.

The integer programming formulation with the above constraints is valid, but in practice, we find this formulation is not scalable. Because every t_j dominating t_i needs to be considered in constraint (8b), this formulation leads to the out-of-memory problem with more than 3,000 tuples, (as shown later in Fig. 5d). To solve this problem, we propose another formulation that considers only each t_j that *directly dominates* t_i.

Definition 6 (Direct Dominance). *A tuple t_j directly dominates t_i, denoted as $t_j \prec_d t_i$, if and only if a) $t_j \prec t_i$, and b) $\nexists t_x$, s.t. $t_j \prec t_x \prec t_i$.*

Direct dominance removes the cascading dominance among tuples. Since only the direct dominating tuples are considered, every t_i is associated with fewer direct dominating tuples. Therefore, we propose the following constraints, where constraint (8b) is replaced with constraints (9b), (9d), and (9e) as

$$\sigma_i + \omega_i + \delta_i = 1, \qquad \forall\, t_i \in D, \quad (9a)$$

$$\delta_i \leq \Sigma_{\forall t_j \prec_d t_i} (\sigma_j + \delta_j), \qquad \forall\, t_i \in D, \quad (9b)$$

$$\sigma_i \leq \omega_j, \qquad \forall\, t_i \in D, \forall t_j \prec_d t_i, \quad (9c)$$

$$\omega_i \leq \omega_j, \qquad \forall\, t_i \in D, \forall t_j \prec_d t_i, \quad (9d)$$

$$\sigma_i + \omega_i = 1. \qquad \forall\, t_i \in D, \nexists t_j \prec_d t_i. \quad (9e)$$

This second formulation tightens the constraints so that the integer programming solver's search space is reduced. Constraint (9b) indicates that as long as one of the direct dominating tuples is either one of the skyline ($\sigma_j = 1$), or dominated by the skyline ($\delta_j = 1$), t_i can be safely assigned with $\delta_i = 1$. Constraint (9d) states that the direct dominating tuple t_j of each disappearing tuple t_i must also vanish, so that all tuples dominating t_i do not appear in the world. The last constraint guarantees that every tuple which is not dominated by other tuple must be a skyline tuple or disappear.

Once this integer programming formulation is settled, we can simply parse the input data into a dominance or direct dominance relation. A general integer programming solver is then applied to find U-Skyline. Since the first problem formulation does not scale, we use the second formulation as a baseline later in our experiment (Section 8).

4 U-Skyline Processing Algorithms

In this section, we first propose a dynamic programming algorithm to obtain U-Skyline from uncertain data sets, and then improve this algorithm with pruning and early termination techniques to avoid unnecessary computation.

An intuitive idea for processing U-Skyline (brute-force) is to enumerate all possible candidate skylines, and evaluate their U-Skyline probabilities (see (5)) to answer the query.

The running time of this brute-force search algorithm, shown in Algorithm 1 is $O(M \cdot N)$, where M is the number of possible candidate skylines. Since a candidate skyline is an incomparable tuple subset of the whole data set D, M can be approximated as $O(a^N)$ with $1 < a < 2$. This brute force search algorithm has a $O(a^N \cdot N)$ running time.

Algorithm 1: Candidate Skyline Search Algorithm

Input: Dataset D
Output: U-Skyline Dataset $S^* = \arg\max_{S_i \subseteq D} P_{u-sky}(S_i)$

1 **for** *Any Candidate Skyline* $S_i \subseteq D$ **do**
2 $\quad P_{\text{u-sky}}(S_i) = \prod_{t_j \in D} \mathtt{PT}(t_j|S_i)$
3 **return** $S^* = \arg\max_{S_i} P_{u\text{-}sky}(S_i)$

4.1 Dynamic Construction of Candidate Skyline

Since Algorithm 1 blindly exhausts the solution space of U-Skyline, we introduce an access order of data tuples to generate candidate skylines and to share the U-Skyline probability computations among different skylines. The access order is governed by a monotonic function [9], [30] as defined below.

Definition 7 (Monotonic Function). *$f(\cdot)$ is a function that maps each tuple to a single numerical value, so that $\forall t_i \prec t_j$, $f(\cdot)$ guarantees $f(t_i) \leq f(t_j)$. One example is $f(t) = \sum_i (t.a_i)$, where $t.a_i$ represents the ith attribute of tuple t.*

The tuple access order is based on the chosen monotonic function $f(\cdot)$. Notice that some monotonic functions, such as $\min_i(t.a_i)$, may lead to ambiguity when there is a tie in tuples' scores, but other functions, such as $\sum_i(t.a_i)$, do not have this problem. In this paper, we do not adopt $\sum_i(t.a_i)$ because it does not have some desired properties for skyline pruning and early termination (to be introduced). Therefore, we choose the function $\min_i(t.a_i)$ as the monotonic function to sort the tuples but use $\sum_i(t.a_i)$ as the tie breaker. It's not required to break the tie of $\sum_i(t.a_i)$. Thus, the final sorted access order ensures that all tuples dominating a tuple t_i have been examined earlier when t_i is being examined. This also ensures that to determine the U-Skyline probability term $\texttt{PT}(t_i|S)$ (see (6)), we only need to consider tuples ahead of t_i. Meanwhile, this access order facilitates dynamic construction of candidate skylines. Let $\mathbb{SC}$ denote a set containing all candidate skylines, which is initialized as $\{\emptyset\}$. While accessing a tuple t_i, our algorithm shall test if merging t_i into any existing candidate skyline $S_j \in \mathbb{SC}$ can form a new candidate skyline $S_j^+ = S_j \cup \{t_i\}$. There are two cases to be considered. If $S_j \prec t_i$, i.e., $S_j \cup \{t_i\}$ is an invalid candidate skyline, then no new candidate skyline is constructed. Otherwise, if $S_j \not\prec t_i$, a new candidate skyline $S_j^+ = S_j \cup \{t_i\}$ is created. We use $\mathbb{SC}_i$ to denote all the candidate skylines created *while accessing* t_i. Accordingly, $\mathbb{SC} = \cup_i \mathbb{SC}_i$.

4.2 Recursive Computation of U-Skyline Probability

Next, we incrementally compute the U-Skyline probability. Here, we first introduce an auxiliary probability, called the *partial U-Skyline probability*.

Definition 8 (Partial U-Skyline Probability). *Suppose that the data set D is sorted according to a monotonic function as D^S. Let D^S_{i-} denote the first i tuples. The U-Skyline probability for a candidate skyline S_j under D^S_{i-} is defined as a partial U-Skyline probability $P^{i-}_{\text{u-sky}}(S_j)$ because it includes only probability terms of tuples in D^S_{i-}, (see (6)).*

$$P^{i-}_{\text{u-sky}}(S_j) = \prod_{t_i \in D^S_{i-}} \texttt{PT}(t_i|S_j). \tag{10}$$

According to (6), for each newly accessed tuple t_i, we only need to determine t_i's U-Skyline probability term for each candidate skyline S_j. More specifically, there are two cases that must be considered. First, if $S_j \prec t_i$, t_i's U-Skyline probability term $\texttt{PT}(t_i|S_j) = 1$ because t_i is dominated, and thus $P^{i-}_{\text{u-sky}}(S_j) = P^{(i-1)-}_{\text{u-sky}}(S_j)$. Note that, in this case, no new candidate skyline can be created by combining t_i with S_j. Second, if $S_j \not\prec t_i$, then $\texttt{PT}(t_i|S_j) = \overline{P(t_i)}$ because S_j is a valid skyline only when t_i does not appear, so $P^{i-}_{\text{u-sky}}(S_j) = P^{(i-1)-}_{\text{u-sky}}(S_j) \cdot \overline{P(t_i)}$. In this case, a new candidate skyline $S_j^+ = S_j \cup \{t_i\}$ must be constructed. Also, because t_i is a part of the candidate skyline S_j^+, $\texttt{PT}(t_i|S_j^+) = P(t_i)$. Because the only difference between S_j^+ and S_j is the new tuple t_i, and because t_i *cannot* dominate any tuple that comes ahead of it, all the U-Skyline probability terms for previously visited tuples do not change after t_i is added to S_j, i.e., $\forall x < i, \texttt{PT}(t_x|S_j) = \texttt{PT}(t_x|S_j^+)$. Therefore, we have $P^{i-}_{\text{u-sky}}(S_j^+) = P^{(i-1)-}_{\text{u-sky}}(S_j^+) \cdot P(t_i) = P^{(i-1)-}_{\text{u-sky}}(S_j) \cdot P(t_i)$.

4.3 Dynamic Programming Algorithm

We summarize the above access order and the corresponding U-Skyline incremental computation into a dynamic programming algorithm (see Algorithm 2), which serves as a basic technique to find U-Skyline from uncertain data.

Algorithm 2: Dynamic Search Algorithm

Input: Dataset D
Output: U-Skyline Dataset $S^* = \arg\max_{S \subseteq D}(P_{u-sky}(S))$

```
1  D^S ← sort(D) based on monotonic function;
2  SC = SC_0 ← {∅};
3  for t_i ∈ D^S, i = 1, 2, ··· , N do
4      SC_i = ∅;
5      for t_x ≠≺ t_i, x = 1, 2, ··· , i − 1 do
6          for S_j ∈ SC_x do
7              if S_j ≠≺ t_i then
8                  S_j^+ ← S_j ∪ {t_i};
9                  P_u-sky(S_j^+) ← P_u-sky(S_j) · P(t_i);
10                 P_u-sky(S_j) ← P_u-sky(S_j) · P̄(t_i);
11                 SC_i ← SC_i ∪ S_j^+;
12     SC = SC ∪ SC_i
13 return S* = arg max_{S_i ∈ ∪SC} P_u-sky(S_i)
```

A running example for Algorithm 2 is illustrated below. The data set has 5 tuples, sorted as $\{a, b, c, e, d\}$. The candidate set $\mathbb{SC} = \mathbb{SC}_0$ is first initiated as $\{\emptyset\}$, where the corresponding partial U-Skyline probability is 1. While exploring the tuple a, $\mathbb{SC}_a$ is derived from $\mathbb{SC}_0$ by including a into $\emptyset$. The partial U-Skyline probability of $\emptyset$ is updated by multiplying $\overline{P(a)} = 0.6$, because $\emptyset$ can only be a valid skyline when a disappears. Following this strategy, when we access the third tuple c, we need to scan all the candidate sets from $\mathbb{SC}_0$ to $\mathbb{SC}_b$. For example, since the candidate $\emptyset$ cannot dominate c, the partial U-Skyline probability of $\emptyset$ is updated by multiplying $\overline{P(c)} = 0.6$, and a new candidate skyline $\{c\}$ is constructed with the partial U-Skyline probability $P_{\text{u-sky}}(\emptyset) \cdot P(c) = 0.3 \cdot 0.4 = 0.12$. On the other hand, for $\{a\}$, since c is dominated by it, its U-Skyline probability remains 0.2. Also, from $\{b\}$, a new candidate skyline $\{b, c\}$ is constructed. However, $\{a, b\}$ is not updated because it dominates c.

The running time for Algorithm 2 still depends on the number of unique candidate skylines. However, since the U-Skyline probability computation for each candidate skyline only takes a constant time, the running time is

reduced to $O(M) \approx O(a^N)$ $(1 < a < 2)$. In the following, we aim to improve this algorithm by reducing the search space of the candidate skylines.

4.4 Skyline Pruning and Early Termination

In this section, we improve the dynamic skyline processing algorithm by pruning the redundant candidate skylines during its execution and to stop the skyline search as early as possible.

In Algorithm 2, when t_i is visited, all the candidate skylines' U-Skyline probabilities are *partial*. Notice that partial U-Skyline probabilities are *larger or equal* to final true U-Skyline probabilities, because

$$P_{\text{u-sky}}(S_j) = \prod_{t_i \in D} \texttt{PT}(t_i|S_j) \leq \prod_{t_i \in D^S_{i-}} \texttt{PT}(t_i|S_j) = P^{i-}_{\text{u-sky}}(S_j). \tag{11}$$

In other words, all the partial U-Skyline probabilities are *non-increasing* during query processing. Thus Algorithm 2 must wait until all the tuples are explored before determining the candidate skyline with the highest U-Skyline probability.

To compare the U-Skyline probabilities *before* all the tuples are explored, an idea is to first find the partial U-Skyline probability that has already reached its final value. From (5), once the remaining tuples are dominated by a candidate skyline S_j, all the remaining tuples' U-Skyline probability terms are 1. In other words, S_j's partial U-Skyline probability reaches its final value $(P^{i-}_{\text{u-sky}}(S_j) = P_{\text{u-sky}}(S_j))$. To do this, we extend the bounding concept from SaLSa [9], and apply the minimum attribute value as the monotonic function of $f(t) = \min_i t.a_i$. More specifically, for each candidate S_j, we can define a dominating score lower bound as follows.

Definition 9 (Dominating Score Lower Bound). *Let $L(S_j)$ be the score lower bound of candidate S_j as*

$$L(S_j) = \min_{t_i \in S_j}(\max_{1 \leq m \leq d} t_i.a_m). \tag{12}$$

Lemma 1. *For any tuple $t_i \in D$ with $f(t_i) > L(S_j)$, $S_j \prec t_i$ must hold.*

Proof. Suppose t_x is the tuple in S_j having $\max_{1 \leq m \leq d} t_x.a_m = L(S_j)$. For any tuple t_i with $f(t_i) = \min_{1 \leq m \leq d} t_i.a_m > L(S_j)$, we have $\forall 1 \leq m \leq d, t_i.a_m > L(S_j) > t_x.a_m$, which means $t_x \prec t_i$. □

With the definition of dominating score lower bound $L(S_j)$, and Lemma 1, we are able to determine a candidate skyline's U-Skyline probability without waiting until the end. Once the processing algorithm visits a tuple t_i with $f(t_i) > L(S_j)$, we immediately know that not only t_i, but all remaining tuples after t_i are dominated by S_j. Since all of remaining tuples' U-Skyline probability terms are 1, we know that the current partial U-Skyline probability of S_j is the same as the final U-Skyline probability: $P^{i-}_{\text{u-sky}}(S_j) = P_{\text{u-sky}}(S_j)$. In other words, we know that the U-Skyline probability for S_j is *finalized*. In addition, this finalized U-Skyline probability can be used to prune other in-progress candidate skylines with the following theorem.

Theorem 1 (Candidate Skyline Pruning). *During the dynamic skyline search, if a candidate skyline S_i's partial skyline probability is lower than a* finalized *U-Skyline probability of $P_{\text{u-sky}}(S_j)$, the candidate S_i and any candidate containing S_i are suboptimal and can be discarded.*

Proof. Because the true U-Skyline probability of S_i is always lower than S_j's skyline probability, S_i is suboptimal and can be discarded. In addition, adding any other data tuples to S_i will not produce the optimal U-Skyline answer. To see this, without loss of generality, we consider a candidate skyline S^+_i, which is extended from S_i by including tuple t_j. Because we have $P^{j-}_{\text{u-sky}}(S_i \cup \{t_j\}) \leq P^{j-}_{\text{u-sky}}(S_i) > P_{\text{u-sky}}(S_j)$, the partial U-Skyline probability of S^+_i is also lower than $P_{\text{u-sky}}(S_j)$. Therefore, by induction, any candidate extended from S_i will have an even lower U-Skyline probability and is subject to pruning. □

Armed with Theorem 1, we can seamlessly include the candidate skyline pruning technique and early termination into the dynamic skyline search framework.

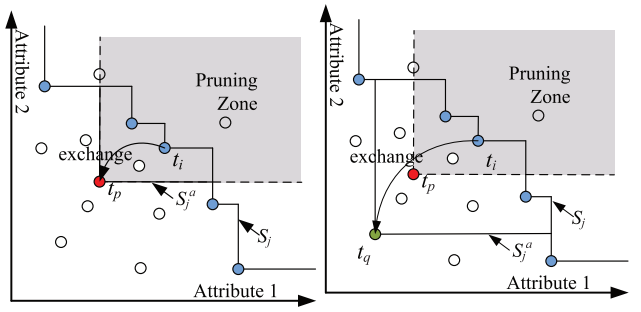
Algorithm 3 follows the dynamic processing framework (Algorithm 2). Here, lines 8-10 check if the current searching candidate skyline S_j is finalized. If so, it is removed from the candidate skyline set, and used to update the best U-Skyline found. Lines 11-12 check if the skyline S_j is already worse than some *finalized* skylines. If so, it is discarded without further processed according to Theorem 1. If both of these two conditions are not satisfied, then S_j is processed against t_i, which is the same as Algorithm 2.

Algorithm 3: Pruning and Early Termination Algorithm

Input: Dataset D
Output: U-Skyline Dataset $S^* = \arg\max_{S \subseteq D}(P_{u-sky}(S))$

```
1  D^S ← sort(D);
2  𝕊ℂ = 𝕊ℂ_0 ← {∅}, L(∅) = ∞, bestUSkyP ← 0;
3  for t_i ∈ D^S, i = 1, 2, ··· , N do
4      𝕊ℂ_i ← ∅;
5      for t_x ≺ t_i, x = 1, 2, ··· , i − 1 do
6          for S_j ∈ 𝕊ℂ_i do
7              if f(t_i) > L(S_j) then
8                  Delete S_j from 𝕊ℂ_x;
9                  Update bestUSkyP, bestUSky ← S_j;
10             else if P_u-sky(S_j) ≤ bestUSkyP then
11                 Delete S_j from 𝕊ℂ_x;
12             else if S_j ⊀ t_i then
13                 S_j^+ ← S_j ∪ {t_i};
14                 P_u-sky(S_j^+) ← P_u-sky(S_j) · P(t_i);
15                 P_u-sky(S_j) ← P_u-sky(S_j) · P̄(t_i);
16                 𝕊ℂ_i ← 𝕊ℂ_i ∪ S_j^+;
17     𝕊ℂ = 𝕊ℂ ∪ 𝕊ℂ_i;
18 return bestUSky
```

We use the same example to illustrate this algorithm. The data tuples are sorted as $\{a, b, c, e, d\}$ and processed accordingly. The pruning effect comes into play when tuple d is accessed. Since $L(\{a, b\}) = 20$, $\{a, b\}$ dominates any tuple t_i with a minimum attribute value $(f(t_i))$ higher than 20. Therefore, because $f(d) = 21 > L(\{a, b\})$, $P^{d-}_{\text{u-sky}}(\{a, b\})$ is finalized while d is visited, with $P_{\text{u-sky}}(\{a, b\}) = 0.2$. Thus, not only $\{a, b\}$ is removed from the candidate set, but all the candidate skylines with partial U-Skyline probabilities lower than 0.2 are also pruned. Since all the



(a) Single Exchange Potential (b) Collaborative Exchange Potential

Fig. 4. Exchange potential.

other pending U-Skyline candidates have lower *partial* U-Skyline probabilities than $\{a, b\}$ at the time, the search is terminated immediately.

5 Data Set Reduction and Partition

While the algorithms proposed earlier efficiently find U-Skyline from an uncertain data set, here we propose two additional techniques, namely, input data set reduction and reduced data set partition, to speed up U-Skyline search by reducing the input data set and applying the strategy of divide-and-conquer in query processing.

5.1 Input Data Set Reduction

A *reduced set* (i.e., $S(D)$) is a subset of the whole data set (D), so that the U-Skyline for the reduced set is the *same* as the U-Skyline in the entire data set, i.e., U-SKY($S(D)$) = U-SKY(D). Therefore, a smaller reduced set shrinks the problem scale exponentially (see Fig. 3).

First, we introduce *exchange potential*. As shown in Fig. 4a, the core idea is to find a tuple t_p, so that all its dominated tuples (e.g., t_i) can be discarded during query processing without affecting the U-Skyline answer. As such, any candidate skyline containing a dominated tuple t_i has a lower U-Skyline probability than an alternative candidate constructed by substituting t_i with t_p .

Definition 10 (Single Exchange Potential). *Single exchange potential (denoted as $r_{p,p}$) is defined for* one tuple t_p. *For any candidate skyline S_j containing a tuple $t_i \succ t_p$, exchanging the tuple t_i to t_p in S_j shall increase its U-Skyline probability for* at least *the following* ratio:

$$r_{p,p} = P(t_p)/\overline{P(t_p)}. \tag{13}$$

Lemma 2 (Single Exchange Pruning). *For any tuple t_p in the data set, if its single exchange potential is higher than 1, $r_{p,p} > 1$, all the tuples* dominated *by t_p, can be safely discarded without affecting the U-Skyline answer.*

The proof of Lemma 2 can be found in Appendix, which is available in the online supplemental material. Because $r_{p,p} > 1$ if and only if $P(t_p) > 0.5$, the Single Exchange Pruning Lemma offers a strong criterion: *as long as a tuple has probability above 0.5, all the tuples in its dominant region can be safely discarded*. In Fig. 4a, the region dominated by t_p is denoted as the pruning region. Any tuples within the pruning region are excluded from the reduced set. In the following, we derive another exchange pruning strategy to consider the case with $P(t_p) \leq 0.5$.

Definition 11 (Collaborative Exchange Potential). *Collaborative exchange potential is defined for a pair of tuples t_p and t_q ($r_{p,q}$), where t_p is the pruning tuple, and t_q is the exchanging target (see Fig. 4b). The collaborative exchange potential represents the* least *U-Skyline probability* increment ratio *by substituting a tuple t_i* dominated by *t_p into t_q. Therefore, the collaborative exchange potential for t_p is as follows:*

$$r_{p,q} = \frac{P(t_q)}{P(t_p) \cdot \overline{P(t_q)}} \cdot \frac{1}{\prod_{\forall t_x, t_q \prec t_x \prec t_p} \overline{P(t_x)}}. \tag{14}$$

Lemma 3 (Collaborative Exchange Pruning). *All tuples dominated by t_p can be discarded without affecting the U-Skyline answer if there exists tuple $t_q \prec t_p$, and their collaborative exchange potential $r_{p,q} > 1$.*

The proof of Lemma 3 can be found in Appendix, which is available in the online supplemental material. Here, once $r_{p,q} > 1$, substituting any tuple dominated by t_p to t_q would always lead to a higher U-Skyline probability.

Based on Lemmas 2 and 3, we obtain the following theorem for pruning.

Theorem 2 (Reduced Set Pruning). *For any tuples t_p, as long as its single or collaborative exchange potential is higher than 1, all the tuples dominated by t_p can be discarded without affecting the final U-Skyline answer set.*

With Theorem 2, we design Algorithm 4 to find the reduced set $S(D)$. Specifically, Algorithm 4 sorts the data tuples according to the monotonic function, and scans each tuple individually. This sorting can guarantee that the exchange target t_q is always ranked ahead of the dominating tuple t_p, and pruned tuple t_i is always accessed after t_p. Therefore, for each tuple t_p, finding its maximum exchange potential actually requires $\frac{p(p+1)}{2}$ time, because line 8 scans the tuples between t_q and t_p for the correct exchange potential. Once we find its exchange potential larger than 1, the tuple is included into the pruning tuple set $PS(D)$. Therefore, when we encounter a tuple dominated by PS, according to Theorem 2, the tuple can be excluded from the reduced set. This reduced set algorithm runs in $O(N^3)$ time, because it needs $O(p^2)$ time to process each tuple t_p.

Algorithm 4: Reduced Set Pruning Algorithm

Input: Dataset D
Output: Reduced set $S(D)$, so that U-Sky($S(D)$) = U-Sky(D)
1 Pruning tuples: $PS(D) \leftarrow \emptyset$;
2 Reduced set: $S(D) \leftarrow \emptyset$;
3 $D^S \leftarrow$ `sort`(D) based on monotonic function;
4 **for** $t_p \in D^S$, $p = 1, 2, \cdots, N$ **do**
5 **if** $PS \not\prec t_p$ **then**
6 $S(D) \leftarrow S(D) \cup \{t_p\}$;
7 **for** $t_q \in D^S, q = 1, 2, \cdots, p$ **do**
8 **if** $r_{p,q} \geq 1$ **then** $PS \leftarrow PS \cup \{t_p\}$; break;
9 **return** $S(D)$

The example for a reduced set algorithm for data set in Fig. 1 is shown below. Similarly, the data set is first sorted

as $\{a, b, c, e, d\}$. Thus, when a is accessed, since there is no other tuple dominating a, its maximum exchange potential is the same as its single exchange potential as $r_{a,a} = P(a)/\overline{P(a)} \approx 0.667$. Therefore, a is included into the reduced set $S(D)$, but not the pruning set PS. The next tuple is b, since its maximum exchange potential is $P(b)/\overline{P(b)} = 1$. Thus, b is included into the $S(D)$ as well as PS. For tuple c, its exchange potential includes two cases: exchanging into either a or itself. For exchanging into a, we have $r_{c,a} = P(a)/(\overline{P(a)P(c)}) = 0.4/0.6/0.6 \approx 1.11 > 1$. Thus, c is also included in PS. For tuples d and e, since both are dominated by pruning tuple b, they are excluded from the reduced set. Therefore, the reduced set of the example is $\{a, b, c\}$. Using this reduced set in our processing framework, the total processing time improves because the algorithm handles fewer data tuples, and the search space is thereby reduced (see Fig. 3).

5.2 Reduced Data Set Partition and Conquest

In order to further reduce the input size, we further introduce a new technique based on divide and conquer to splitting the reduced set into a number of disjoint subsets. We process the U-Skyline for each subset independently and then merge the answers to form the final U-Skyline answer.

Consider the example in Fig. 1. Some tuple sets are independent with each other in dominance relations. Specifically, reduced set $\{a, b, c\}$ can be split into two independent sets, $\{a, c\}$ and $\{b\}$ because no part of $\{a, c\}$ has any dominance relationship with $\{b\}$. In addition, whether tuple a or c shows up does not influence b's U-Skyline probability term, i.e., $\texttt{PT}(b|\{a, b, c\}) = \texttt{PT}(b|\{b\})$. Similarly, tuple b's appearance also does not affect a or c's U-Skyline probability terms. This observation suggests that we can partition the reduced set into two groups: $\{a, c\}$ and $\{b\}$, evaluate the U-Skyline for each group separately, and then merge the U-Skyline groups together (shown in Fig. 3). To effectively partition the reduced set, we define the set independent partition as follows.

Definition 12 (Reduced Set Independent Partition). *For reduced set $S(D)$ of the data set D, an independent partition is represented as $\mathcal{P}(S(D)) = \{G_1, G_2, \ldots, G_M\}$ so that for any two tuples t_i and t_j from different partition groups, we have*

1. *$t_i \not\prec t_j$ and $t_j \not\prec t_i$.*
2. *$\nexists t_k \in S(D)$, so that $t_i \prec t_k \wedge t_j \prec t_k$ or $t_k \prec t_i \wedge t_k \prec t_i$.*

Lemma 4. *For a valid independent partition for $S(D)$ as $\mathcal{P}(S(D)) = \{G_1, G_2, \ldots, G_M\}$, an arbitrarily chosen candidate skyline S_i, and a tuple t_j ($t_j \in G_j$), $\texttt{PT}(t_j|S_i) = \texttt{PT}(t_j|(G_j \cap S_i))$ must hold.*

Lemma 4 is correct because for an arbitrarily chosen S_i, only tuples in $S_i \cap G_j$ may dominate t_j or be dominated by t_j. In other words, any other skyline tuples in other groups are irrelevant to t_j and $\texttt{PT}(t_j|S_i)$, because $\texttt{PT}(t_j|S_i)$ is determined by only $S_i \cap G_j$.

Theorem 3. *The U-Skyline of $S(D)$ is the same as the union of U-Skylines of all the independent partitions $\mathcal{P}(S(D))$*

$$\text{U-SKY}(D) = \text{U-SKY}(S(D)) = \bigcup_{G_i \in \mathcal{P}(S(D))} \text{U-SKY}(G_i).$$

Proof. We prove this theorem by showing that maximizing the global U-Skyline probability is equivalent to maximizing the U-Skyline probability for each independent partition. Starting from the U-Skyline probability of candidate skyline, we have

$$\begin{aligned} P_{\text{u-sky}}(S) &= \prod_{t_i \in S(D)} \texttt{PT}(t_i|S) \\ &= \prod_{G_j \in \mathcal{P}(S(D))} \left(\prod_{t_i \in G_j} \texttt{PT}(t_i|(S \cap G_j)) \right) \\ &= \prod_{G_j \in \mathcal{P}(S(D))} P_{\text{u-sky}}(S \cap G_j). \end{aligned} \tag{15}$$

In (15), we first partition the U-Skyline probability terms into different groups. Then, we apply Lemma 4 to replace each probability term. After this, the U-Skyline probability of an arbitrary candidate skyline S is divided into the product of the individual group's U-Skyline probability. Therefore, maximizing the entire U-Skyline probability is equivalent to maximizing each group's U-Skyline probability. □

Given Theorem 3, we design an algorithm to partition the reduced set, as shown in Algorithm 5.

Algorithm 5: Reduced Set Independent Partitions

Input: Reduced Set $S(D)$
Output: Independent Partitions $\mathcal{P}(S(D))$

```
1 S(D)^S ← sort(S(D));
2 ∀i = 1, 2, ··· , |S(D)|, G_i = {t_i};
3 for t_i ∈ S(D)^S, i = 1, 2, ··· , |S(D)| do
4     ∀G_j ≺ t_i, merge G_j s into a single group;
5 return All groups as P(S(D))
```

For the same example in Fig. 1 with $S(D) = \{a, b, c\}$. The independent partition algorithm would first initialize the tuples into three independent groups $\{a\}, \{b\}, \{c\}$ in line 2, and explore the dominance relationships among them. The access order is $\{a, b, c\}$. When the algorithm accesses tuple c, it discovers tuple c is dominated by a. Therefore, $\{a\}$ and $\{c\}$ are merged together as $\{a, c\}$. After the entire algorithm is completed, it returns independent partitions $\{a, c\}$ and $\{b\}$. In this way, following processing algorithm ends up with much less search space than handling the reduced set as a whole (see Fig. 3).

6 Extensions

In this section, we first devise an approximation heuristic for U-Skyline query processing and then extend U-Skyline to support tuples with multiple instances.

6.1 Approximation Algorithms for U-Skyline

Since the searching for the U-Skyline is NP-hard, a natural direction worthy for further investigation is to find an approximated U-Skyline answer set. Our idea is to find an approximated answer by limiting the search space explored in the proposed U-Skyline algorithms. Once the searched skyline space exceeds a predefined threshold λ, a

proactive thread will be triggered to dump inferior candidate skylines.

In the dynamic search algorithm (Algorithm 2) and all of the extensions, one important property is that all the candidate skylines are buffered with their partial U-Skyline probabilities. This buffered set increases when a newly accessed tuple is incomparable to an existing candidate skyline. Our pruning and early termination techniques in Algorithm 3 only discard candidate skylines when they are proved not to be the optimal solution. In this approximation heuristic, however, we introduce a candidate skyline limit λ, such that, whenever the candidate set size exceeds λ, the candidate skyline with the lowest partial U-Skyline probability is discarded without further processing. We choose the candidates with the minimum partial U-Skyline probabilities because these candidates have smaller chances to appear in good solutions. The time complexity of the approximation algorithm is $O(\lambda N^2)$, where λ stands for the buffer size for candidate skylines, and each tuple thereby is tested for the dominance relation against at most λ candidate skylines, each with at most N tuples. By tuning λ, we are able to achieve good balance between approximation accuracy and processing efficiency.

6.2 Multiinstance Support for U-Skyline

In the above query processing techniques, we assume that each tuple only has one instance. In the following, we extend U-Skyline to support multiple instances associated with each tuple. More specifically, each tuple contains a set of tuple instances $t_i = \{t_i^1, t_i^2, \ldots, t_i^{N_i}\}$. Each tuple instance t_i^j is associated with d attributes $\{t_i^j.a_1, t_i^j.a_2, \ldots, t_i^j.a_d\}$ and a probability $P(t_i^j)$. Therefore, the tuple's probability is defined as the sum of the probabilities of all tuple instances, $P(t_i) = \sum_j P(t_i^j) \leq 1$. Also, the appearances of tuple instances need to be mutually exclusive for each tuple. In other words, if t_i^x appears, $t_i^y, y \neq x$ cannot appear. To support multi-instance tuples in U-Skyline, each candidate skyline S is allowed to have multiple skyline instances $\Delta_S = \{S^1, S^2, \ldots, S^{M_S}\}$, and M_S denotes the number of possible instances for S. To generate a skyline instance S^k from S, each tuple in S must contribute one and only one instance ($S^k = \{t_i^k | \forall t_i \in S\}$). Also the generated skyline instance S^k must be a valid skyline which does not have dominance relation between any two tuple instances. Under this definition, the U-Skyline probability for S is the sum of U-Skyline probability for all valid skyline instances from Δ_S

$$P_{\text{u-sky}}(S) = \sum_{S^k \in \Delta_S} \left(\prod_{\forall t_i^k \in S^k} P(t_i^k) \cdot \prod_{\forall t_j \notin S} \left(1 - \sum_{\forall t_j^x S^k \not\prec t_j^x} P(t_j^x) \right) \right). \tag{16}$$

Compared to the single-instance case in (5), the multi-instance U-Skyline probability includes a sum operator for all the possible skyline instances in Δ_S. Moreover, the probabilities of all nonappearing instances are required to be summed up in the second term because their appearances are mutual-exclusive.

The DP algorithm (Section 4.3) are performed in the similar way as the previous version. The candidate skyline kept in DP algorithm needs to be replaced with candidate skyline instances (e.g., $S^k \in \Delta_S$). A similar procedure for skyline instance construction can be applied because each skyline instance consists of tuple instances with only one possible value. The difference is that the computed U-Skyline probabilities for skyline instances must be summed up to find U-Skyline probabilities following the new U-Skyline probability definition (3).

Similarly, the pruning and early termination (Section 4.4) can be applied to consider candidate skyline instances. Because the role of a skyline instance is similar to a candidate skyline under the single-instance U-Skyline query, the U-Skyline probability of one skyline instance is also finalized when all dominating tuples are visited. Since the U-Skyline probability for a candidate skyline is the sum of the probabilities of all possible skyline instances, the candidate skyline's probability is obtained after considering all its instances. Therefore, P&ET can also be applied to a candidate skyline instance with the partial U-Skyline probability smaller than one finalized U-Skyline probability.

The input Set Reduction optimization can be extended to multi-instance scenarios by considering tuple instance's exchange potentials. Because a tuple has multiple values, only tuple instances can be used as pruning references here. For example, a tuple t_p's instance t_p^k in a candidate skyline instance S^k has a single exchange potential $r_{t_p^k, t_p^k}$ and a collaborative exchange potential $r_{t_p^k, t_q^x}$ with another tuple instance t_q^x of tuple t_q^x, where $t_q^x \prec t_p^k$. The exchange potential of tuple instances is calculated in the similar way as introduced in Section 5.1, except that the mutually exclusive relation among tuple instances needs to be handled for calculating collaborative exchange potentials. Once we identify a pruning tuple instance t_p^k, a tuple t_i can be pruned when all of t_i's tuple instances are dominated by t_p^k. This is because if each tuple instance of t_i is inferior to t_p^k, any candidate skyline including t_i is inferior to an alternative skyline having t_p for U-Skyline probability.

Finally, the Set Partition technique can also be extended. For multi-instance tuples, two tuple sets can be decoupled only when all of their tuple instances do not dominate each other.

Although the introduced techniques can be extended under the multi-instance scenario, we are aware that the problem complexity significantly increases. This behavior is similar to P-Skyline with multi-instances introduced in [30].

7 Empirical Study

We perform an extensive empirical study to examine the U-Skyline query under both synthetic and real data sets. All experiments are conducted on an IBM server 3550 (CPU: two Intel Xeon X5450 3.00 GHz (4 cores), RAM: 8G, OS: Windows Server 2008). For the integer programming problem approach, we use a commercial parallel optimizer CPLEX [3], which supports multithreaded parallel optimization to obtain U-Skyline answers. The running time of CPLEX is used as the baseline to compare with our

TABLE 1
Experiment Parameters

Parameter	Variation Range	Default Value
Dataset Size (N)	$10 \sim 100,000$	10,000
Dimensionality (d)	$2 \sim 6$	3
Correlation (r)	$-0.75 \sim 0.75$	0
Probability Center (α)	$0.2 \sim 0.8$	0.5

TABLE 2
Evaluated Algorithms

Technique Name	Abbreviation	Introduced Section
Dynamic Programming Search	DP	Section 4.3
Pruning and Early Termination	P&ET	Section 4.4
Input Dataset Reduction	SR	Section 5.1
Reduced Dataset Partition	SP	Section 5.2

proposed algorithms. In addition, we also compare the query answers obtained by U-Skyline and P-Skyline.

7.1 Synthetic Data Study

We evaluate the U-Skyline query with synthetic data using different cardinalities and dimensionality settings. To generate synthetic data, we first utilize a d-dimensional independent uniform random variable vector X to generate the correlated vector $Y = X \cdot \texttt{CHOL}(R)$, where R is the correlation matrix with $R = r \cdot U + (1 - r) \cdot I$ (diagonal elements are 1s and other elements are r), and $\texttt{CHOL}(R)$ is the Cholesky decomposition of the correlation matrix R [2]. Following this correlated data generation method, any two random variables within vector Y have correlation $r, (-1 < r < 1)$. Y is used as a generated tuple in our experiment. In addition, the probability of each tuple is uniformly distributed between $\alpha - \delta$ and $\alpha + \delta$, where α is the center and δ represents one side width, $\delta = \min(\alpha, 1 - \alpha)$. The generated probabilities are always between $[0, 1]$.[4]

The parameters in our experiments are summarized in Table 1. To compare the results in different scenarios, we vary the parameters shown in Table 1 one at a time while keeping the rest fixed. Under each test case, the U-Skyline properties (i.e., U-Skyline probability, U-Skyline size, reduced set size, and the largest set partition size) are studied. We also compare P-Skyline against U-Skyline from the aspects of precision and recall. The precision is the number of the same tuples in P-Skyline and U-Skyline divided by P-Skyline's size. The recall is the number of the same tuples divided by U-Skyline's size. In addition, we evaluate the running time and search spaces, i.e., candidate sizes, for all proposed techniques summarized in Table 2.

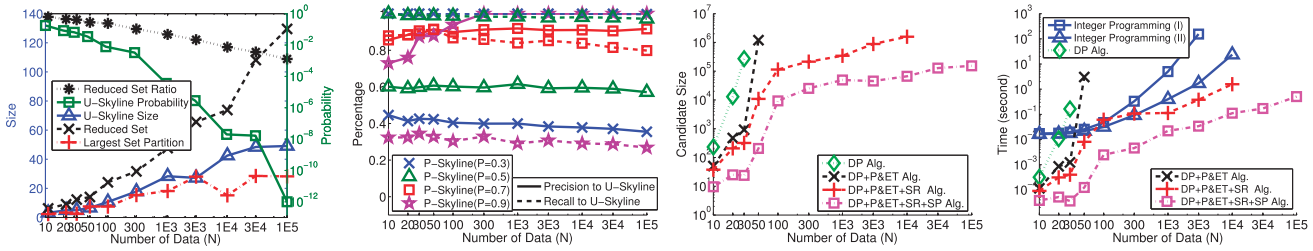
The first set of experiments compares the U-Skyline processing efficiency under different data set sizes. Here, all tuple values are independent, and their probabilities follow a uniform distribution between 0 and 1. In Fig. 5a, we compare the U-Skyline answer size, reduced input set size, reduced set ratio against entire data set, and the largest set partition size to demonstrate the proposed pruning techniques. First, compared to the U-Skyline answer size, the reduced set is usually about 2-3 times larger. In other words, after reduced set pruning, we still have some redundant tuples that must be processed but eventually are not included in the final U-Skyline result. However, compared to the entire data set size, the reduced set ratio is only about 1 percent and thereby demonstrates that the reduced set pruning is very effective. The set partition algorithm (SR) is able to further partition the reduced set $S(D)$ into smaller tuple groups, i.e., $\mathcal{P}(S(D)) = \{G_1, G_2, \ldots, G_M\}$. This enables us to process each smaller partition individually. Since the running time to process each partition group is exponential to its size, the largest set partition $\max|G_i|$ is the bottleneck to find the U-Skyline. As shown in Fig. 5a, the largest partition group size is reduced to a very small level. With this partition technique, we significantly reduce the search space of the U-Skyline query because the skyline search is restricted to each partition group. Thus, we do not need to explore the candidate skyline across partitions to get the correct U-Skyline answer. The average U-Skyline probabilities are also shown, and a smaller U-Skyline probability is obtained when the input data set size becomes large.

Fig. 5b compares P-Skyline and U-Skyline under different thresholds. The results indicate that a better precision can be obtained at the cost of lowering the recall for a higher P-Skyline threshold. The best balance between precision and recall can be reached by selecting 0.7 as the P-Skyline threshold, but U-Skyline still outperforms P-Skyline by 10 percent. Moreover, U-Skyline does not require a threshold for finding the optimal solutions.

In Fig. 5c, we study the number of candidate skyline sets explored during U-Skyline processing. This number represents the actual search space for each algorithm and is related to the running time of the algorithm. Among all the proposed techniques, the DP algorithm without any other techniques performs the worst, because it needs to enumerate all the candidate skylines among the entire data set, which exponentially grows with N. After applying P&ET, the candidate skyline size is smaller because some of them are removed before being extended to other candidate skylines. In contrast, as lots of redundant data are excluded from the reduced set, the input data set reduction technique can significantly reduce the candidate size and set partitioning further decreases the candidate skyline size by solving U-Skyline group by group.

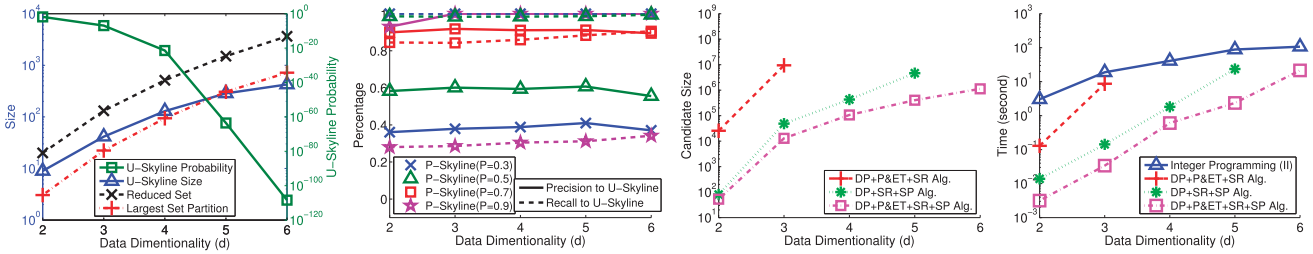
In Fig. 5d, integer programming methods serve as a baseline. First, the direct domination integer programming formulation (formulation II) always performs better than formulation I, because it requires fewer variables and constraints. Thus, we only use this formulation to compare our proposed techniques in the following experiments. Fig. 5d shows that our proposed single-thread algorithm (DP + P&ET + SR + SP) outperforms the parallel optimizer. Among the proposed algorithms, the DP algorithm has exponentially increasing running time and cannot handle data sets with more than 50 tuples. In contrast, DP+P&ET algorithm can solve U-Skyline query with fewer than 100 tuples. SR technique can significantly reduce the processing time, and SP will further reduce the running time and enable our proposed algorithm to handle more than 100,000 tuples.

4. An α close to 0 or 1 will lead to a smaller interval.



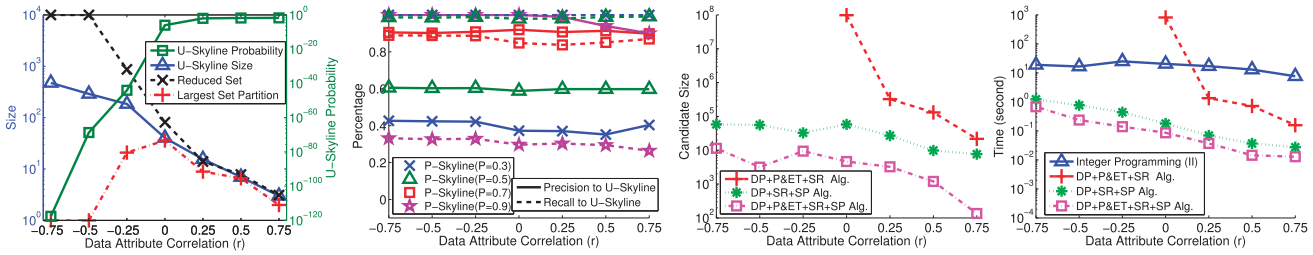
(a) Typical Statistics (b) Precision/Recall Comparision (c) Candidate Size (d) Query Execution Time

Fig. 5. Experimental results with different data size (N).



(a) Typical Statistics (b) Precision/Recall Comparision (c) Candidate Size (d) Query Execution Time

Fig. 6. Experimental results with different data dimensionality(d).



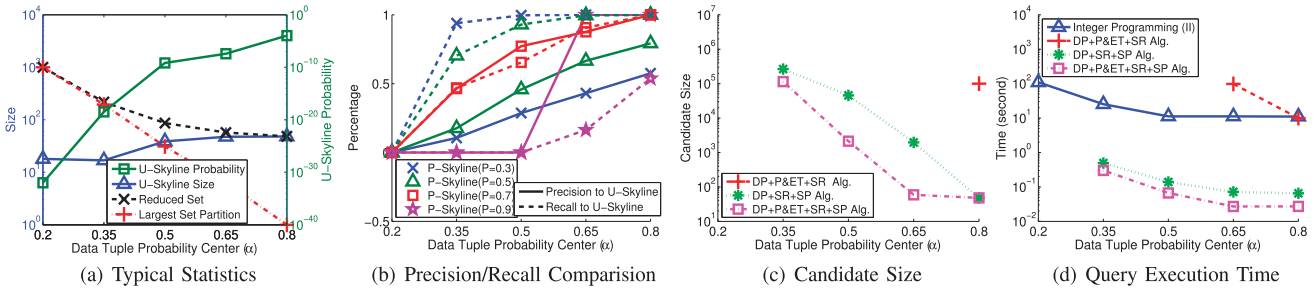
(a) Typical Statistics (b) Precision/Recall Comparision (c) Candidate Size (d) Query Execution Time

Fig. 7. Experimental results with different tuple attribute correlation(r).

Fig. 6 investigates the impact of dimensionality with the data set size set to 10,000. From Fig. 6a, as the dimensionality increases, it is more difficult for tuples to dominate each other. Therefore the typical set sizes increase accordingly. When dimensionality is 2, the reduced set size is only about 30 as shown in Fig. 6a. As the U-Skyline size increases along with data dimensionality, the U-Skyline probability decreases accordingly. The size of U-Skyline is smaller than that of the classic skyline because U-Skyline needs to consider the probability of each tuple and thereby is inclined to discard the tuples with small probabilities, even though the tuples are included in the classic skyline. In Fig. 6b, the comparison of P-Skyline under different thresholds is also presented. The balanced precision and recall is found when P-Skyline threshold is 0.7. Figs. 6c and 6d compare the search space and running time. Here, the results of DP and DP + P&ET algorithms are not included, because they are unable to support such a large data set. In addition, DP + SR + SP algorithm is provided to demonstrate the effect of P&ET as compared to DP + P&ET + SR + SP algorithm. DP + P&ET + SR is not able to process data with the dimensionality more than 3 because the candidate skyline search space exponentially increases to 10^7 when dimensionality is 3, leading to memory exhaustion. In Fig. 6c, the result indicates that the search space of U-Skyline decreases according to SR and SP. Finally, the average query execution time is reported in Fig. 6d. With all the proposed algorithm improvements, our single-thread algorithm outperforms CPLEX, especially when d is small.

Fig. 7 shows the comparison of performance under different attribute correlations. For $r < 0$, the attributes are anticorrelated. In contrast, when $r > 0$, they are correlated and more likely to dominate each other, thereby leading to fewer skyline tuples. As shown in Fig. 7a, the reduced set size decreases as expected when tuples become more correlated. However, our experiment reveals more interesting results. SP technique, working on top of the SR, can partition the reduced set into independent smaller tuple subsets. For significantly anticorrelated data, (e.g., $\rho = -0.75, -0.5$), very few tuples dominate others, and therefore the SP technique can separate them into very small independent sets, (e.g., each with size 1 in extreme cases). Therefore, for anticorrelated data, although the reduced set is still large, the set partition sizes become quite small. Thanks to this, algorithms with SP can process these small partitions individually quickly, and merge the results together to get the optimal U-Skyline. In Fig. 7c, while the DP+P&ET+SR algorithm cannot handle the anticorrelated data, algorithms with SP can achieve it efficiently with significantly reduced search space. Therefore, even compared to the multicore parallel optimizer, our algorithms can significantly decrease the running time, as shown in Fig. 7d.

Fig. 8 shows the results with different tuple probabilities. Fig. 8a indicates that the reduced set size and the largest set partition size significantly increases when the probability



(a) Typical Statistics (b) Precision/Recall Comparision (c) Candidate Size (d) Query Execution Time

Fig. 8. Experimental results with different tuple probability(α).

becomes small, even though the sizes of the U-Skyline results remain almost the same for different probabilities. This is because it is difficult to remove tuples have small probabilities from the reduced set. Therefore, keeping these redundant tuples also reduces the chance to separate them into independent partitions. In contrast, as the tuple probabilities increase, e.g., $\alpha = 0.8$, most skyline tuples have attendance probabilities higher than 0.5. According to the single exchange potential, only skyline tuples can be included in the reduced set, and our algorithms thereby will perform better with fewer redundant tuples included. As shown in Figs. 8c and 8d, the search space and running time for U-Skyline can be significantly reduced as α increases. In Fig. 8b, it is shown that when tuples probabilities are small, it is almost impossible to approximate U-Skyline with P-Skyline. When tuples probabilities increase, both P-Skyline and U-Skyline are degenerated to classic skyline problem with the same results.

Finally, we compare the coverage of P-Skyline and U-Skyline in Fig. 9. Since U-Skyline never includes any two tuples with one dominating the other in the result, we demonstrate the difference of the coverage by evaluating P-Skyline with different qualifying thresholds (P) against U-Skyline. Here, we define the *not covered probability* of skyline result S as $P_{\text{not covered}}(S)$, which is the probability that one or more nondominated tuples ($t_i \notin S \wedge S \not\prec t_i$) exist. Thus a smaller $P_{\text{not covered}}(S)$ for each result set S leads to better coverage. As shown in Fig. 9, a larger qualifying threshold for P-Skyline generates fewer tuples qualified for P-Skyline but $P_{\text{not covered}}(S)$ also grows in this case. To improve the P-Skyline's coverage, a smaller qualifying threshold can be used, but the number of P-Skyline tuples would increase accordingly. In contrast, the U-Skyline can only generate one result for each data set, and for this data set (1,000 tuples with three dimensionality, tuple probabilities uniformly distributed between 0 and 1), the U-Skyline result size is around 20.

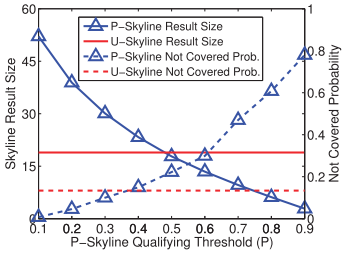


Fig. 9. Comparison of P-Skyline and U-Skyline.

To generate a U-Skyline with similar size, P-Skyline must choose a threshold around 0.5. However, the P-Skyline's result has an inferior uncovered probability in this case.

7.2 Real Data Set

Next, we test the U-Skyline processing algorithm using a real world data set of used cars. The data set is gathered from a used car website over a 10-month period (2006/1-2006/11) with 42,543 data objects [1]. From the data set, we not only find four numerical attributes for each car: id, mileage, make year and price, but also the date date_a when vehicle is posted on the website and the date date_d when it is deleted. To convert the data set into an uncertain data set, we obtain each vehicle's available probability with a hidden Markov model (HMM) as shown in Fig. 10. The transition graph assumes an independent selling probability for each vehicle, and assumes that once the car is sold, it shall not return to the available state.

With this model, for each used car posted on the website on a certain day, we are able to infer the probability that it is at the available state, or at the sold state. Using this probability, we can assign the available probability for each tuple. To consider different cardinalities (N), we multiply the sample date, and merge the used car data from these dates, until the merged data set reaches N. For each vehicle, we use the mileage, make year, and price as three numerical attributes. For different data set cardinalities, the result for the real data set is shown in Fig. 11. We can find that the results are consistent to Fig. 5. This validates that the practicality of the proposed U-Skyline query in real life applications.

8 Conclusion

In this paper, we propose a new uncertain skyline operator called U-Skyline. We show that processing U-Skyline is an NP-hard problem. Thus, we propose a dynamic programming framework for processing U-skyline and developed a

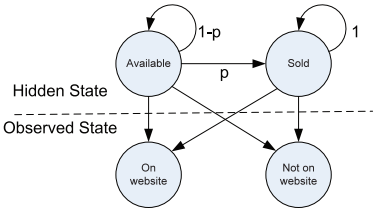
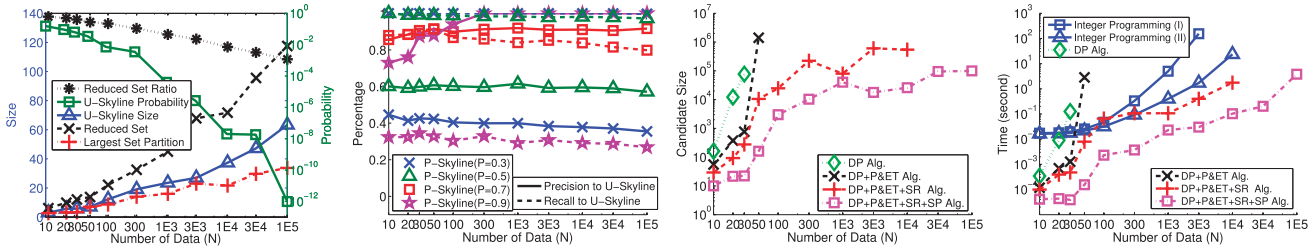


Fig. 10. Experimental results with real data set.



(a) Typical Statistics (b) Precision/Recall Comparision (c) Candidate Size (d) Query Execution Time

Fig. 11. Experimental results with real data set.

series of optimization techniques, including probability computational simplification, candidate skyline pruning and early termination, data set reducing, and reduced set partition, to alleviate the computational strain. We compare our U-Skyline processing algorithms to a commercial parallel integer programming solver, and conduct experiments to demonstrate the efficiency of our algorithms. Experimental results show that our algorithm (executed in a single thread) is 10-100 times faster than the parallel integer programming solver executed on an 8-core IBM 3,650 server.

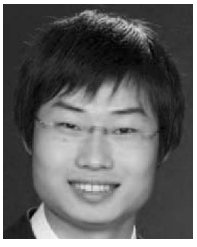
Acknowledgments

The work was supported in part by the National Science Council of Taiwan under Contract NSC100-2219-E-001-002, NSC100-2221-E-001-006-MY2, and NSC101-2628-E-001-003-MY3.

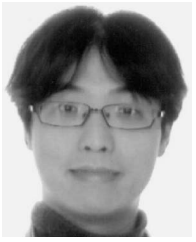
References

[1] Chen Li's Data Set, http://www.ics.uci.edu/~chenli/webobjects/, 2012.
[2] Generating Correlated Data, http://www.uvm.edu/~dhowell/StatPages/More_Stuff/Gener_Correl_Numbers.html, 2012.
[3] IBM ILOG CPLEX Optimizer, http://www.ibm.com/software/integration/optimization/cplex-optimizer/, 2012.
[4] S. Abiteboul, P. Kanellakis, and G. Grahne, "On the Representation and Querying of Sets of Possible Worlds," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '87)*, pp. 34-48, 1987.
[5] C.C. Aggarwal, "On Unifying Privacy and Uncertain Data Models," *Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE '08)*, pp. 386-395, 2008.
[6] C.C. Aggarwal and P.S. Yu, "A Survey of Uncertain Data Algorithms and Applications," *IEEE Trans. Knowledge and Data Eng.*, vol. 21, no. 5, pp. 609-623, May 2009.
[7] P. Agrawal, O. Benjelloun, A.D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom, "Trio: A System for Data, Uncertainty, and Lineage," *Proc. 32nd Int'l Conf. Very Large Data Bases (VLDB '06)*, pp. 1151-1154, 2006.
[8] M.J. Atallah and Y. Qi, "Computing All Skyline Probabilities for Uncertain Data," *Proc. 28th ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS '09)*, pp. 279-287, 2009.
[9] I. Bartolini, P. Ciaccia, and M. Patella, "SaLSa: Computing the Skyline without Scanning the Whole Sky," *Proc. 15th ACM Int'l Conf. Information and Knowledge Management (CIKM '06)*, pp. 405-414, 2006.
[10] S. Borzsonyi, K. Stocker, and D. Kossmann, "The Skyline Operator," *Proc. 17th Int'l Conf. Data Eng. (ICDE '01)*, pp. 421-430, 2001.
[11] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu, "MYSTIQ: A System for Finding More Answers by Using Probabilities," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '05)*, pp. 891-893, 2005.
[12] R. Cheng, D.V. Kalashnikov, and S. Prabhakar, "Evaluating Probabilistic Queries Over Imprecise Data," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '03)*, pp. 551-562, 2003.
[13] R. Cheng, S. Singh, and S. Prabhakar, "U-DBMS: A Database System for Managing Constantly-Evolving Data," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05)*, pp. 1271-1274, 2005.
[14] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J.S. Vitter, "Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data," *Proc. 13th Int'l Conf. Very Large Data Bases (VLDB '04)*, pp. 876-887, 2004.
[15] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with Presorting," *Proc. 19th Int'l Conf. Data Eng. (ICDE '03)*, 2003.
[16] G. Cormode, F. Li, and K. Yi, "Semantics of Ranking Queries for Probabilistic Data and Expected Ranks," *Proc. IEEE Int'l Conf. Data Eng. (ICDE '09)*, pp. 305-316, 2009.
[17] N. Dalvi and D. Suciu, "Efficient Query Evaluation on Probabilistic Databases," *Proc. 13th Int'l Conf. Very Large Data Bases (VLDB '04)*, pp. 864-875, 2004.
[18] A. Dempster, N. Laird, and D. Rubin, "Maximum Likelihood from Incomplete Data Via the em Algorithm," *J. Royal Statistical Soc. Series B (Methodological)*, vol. 39, pp. 1-38, 1977.
[19] T. Hofmann, "Probabilistic Latent Semantic Analysis," *Proc. Uncertainty in Artificial Intelligence Conf. (UAI)*, pp. 289-296, 1999.
[20] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '08)*, pp. 673-686, 2008.
[21] J. Huang, L. Antova, C. Koch, and D. Olteanu, "MayBMS: A Probabilistic Database Management System," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '05)*, pp. 1071-1074, 2009.
[22] C. Jin, K. Yi, L. Chen, J.X. Yu, and X. Lin, "Sliding-Window Top-k Queries on Uncertain Streams," *Proc. VLDB Endowment*, vol. 1, pp. 301-312, 2008.
[23] D. Kossmann, F. Ramsak, and S. Rost, "Shooting Stars in the Sky: An Online Algorithm for Skyline Queries," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02)*, pp. 275-286, 2002.
[24] K.C.K. Lee, B. Zheng, H. Li, and W.-C. Lee, "Approaching the Skyline in Z Order," *Proc. 33rd Int'l Conf. Very Large Data Bases (VLDB '07)*, pp. 279-290, 2007.
[25] X. Lian and L. Chen, "Monochromatic and Bichromatic Reverse Skyline Search Over Uncertain Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '08)*, pp. 213-226, 2008.
[26] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting Stars: The K Most Representative Skyline Operator," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE '07)*, pp. 86-95, 2007.
[27] X. Liu, M. Ye, J. Xu, Y. Tian, and W. Lee, "K-Selection Query Over Uncertain Data," *Proc. 15th Int'l Conf. Database Systems for Advanced (DASFAA)*, pp. 444-459, 2010.
[28] E. Michelakis, R. Krishnamurthy, P.J. Haas, and S. Vaithyanathan, "Uncertainty Management in Rule-Based Information Extraction Systems," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '05)*, pp. 101-114, 2009.
[29] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive Skyline Computation in Database Systems," *ACM Trans. Database System*, vol. 30, no. 1, pp. 41-82, 2005.
[30] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," *Proc. 33rd Int'l Conf. Very Large Data Bases (VLDB '07)*, pp. 15-26, 2007.
[31] S. Prithviraj and A. Deshpande, "Representing and Querying Correlated Tuples in Probabilistic Databases," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE '07)*, pp. 596-605, 2007.
[32] C. Re, N. Dalvi, and D. Suciu, "Efficient Top-K Query Evaluation on Probabilistic Data," *Proc. Int'l Conf. Data Eng. (ICDE '07)*, pp. 896-905, 2007.

[33] A.D. Sarma, O. Benjelloun, A. Halevy, and J. Widom, "Working Models for Uncertain Data," *Proc. 22nd Int'l Conf. Data Eng. (ICDE '06)*, p. 7, 2006.
[34] M.A. Soliman, I.F. Ilyas, and K.C. Chang, "Top-k Query Processing in Uncertain Databases," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE '07)*, pp. 896-905, 2007.
[35] K.-L. Tan, P.-K. Eng, and B.C. Ooi, "Efficient Progressive Skyline Computation," *Proc. 27th Int'l Conf. Very Large Data Bases (VLDB '01)*, pp. 301-310, 2001.
[36] Y. Tao, R. Cheng, X. Xiao, W.K. Ngai, B. Kao, and S. Prabhakar, "Indexing Multi-Dimensional Uncertain Data with Arbitrary Probability Density Functions," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05)*, pp. 922-933, 2005.
[37] A. Vlachou, C. Doulkeridis, and Y. Kotidis, "Angle-Based Space Partitioning for Efficient Parallel Skyline Computation," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '08)*, pp. 227-238, 2008.
[38] M. Ye, K. Lee, W. Lee, X. Liu, and M. Chen, "Querying Uncertain Minimum in Wireless Sensor Networks," *IEEE Trans. Knowledge and Data Eng.*, vol. 24, no. 12, pp. 2274-2287, Dec. 2012.
[39] M. Ye, W. Lee, D. Lee, and X. Liu, "Distributed Processing of Probabilistic Top-K Queries in Wireless Sensor Networks," *IEEE Trans. Knowledge and Data Eng.*, vol. 25, no. 1, pp. 76-91, Jan. 2013.
[40] H. Yong, J. ha Kim, and S. won Hwang, "Skyline Ranking for Uncertain Data with Maybe Confidence," *Proc. IEEE 24th Int'l Conf. Data Eng. Workshop (ICDEW '08)*, pp. 572-579, 2008.
[41] W. Zhang, X. Lin, Y. Zhang, W. Wang, and J.X. Yu, "Probabilistic Skyline Operator Over Sliding Windows," *Proc. IEEE Int'l Conf. Data Eng. (ICDE '09)*, pp. 1060-1071, 2009.

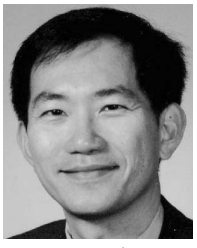


Xingjie Liu received the BE degree in electronic engineering from the Tsinghua University, Beijing, China, in June 2003. Currently, he is working toward the PhD degree in computer science at the Pennsylvania State University. His research interests spans a number of disciplines, including database query engine, query evaluation and optimization, distributed system, and statistical database analysis.



De-Nian Yang received the BS and PhD degrees from the Department of Electrical Engineering at National Taiwan University in 1999 and 2004, respectively. He is now an associate research fellow in the Institute of Information Science and Research Center for Information Technology Innovation, Academia Sinica, Taiwan. His research interests include mobile data management, social networks, and multimedia mobile networking. He received the best student paper award from the IEEE International Conference on Multimedia and Expo (IEEE ICME), the Emerging Technologies Prize from ACM SIGGRAPH Asia, the Career Development Award from Academia Sinica, the K. T. Li Young Researcher Award from IICM, the Research Exploration Award from the Pan Wen Yuan Foundation, and the Project for Excellent Junior Research Investigators from the National Science Council of Taiwan. He is a senior member of IEEE and a member of ACM.

Mao Ye received the bachelor's degree in computer science from the Nanjing University, China, in 2004. Currently, he is working toward the PhD degree in the Department of Computer Science and Engineering, Pennsylvania State University, University Park. He served as a research assistant at the City University of Hong Kong from September 2005 to August 2006. His research interests include data and knowledge engineering and distributed systems.



Wang-Chien Lee received the PhD degree from the Computer and Information Science Department at The Ohio State University. He worked as a principal member of the technical staff at Verizon/GTE Laboratories, Inc. He is an associate professor of computer science and engineering at Pennsylvania State University (Penn State). Currently, he leads the Pervasive Data Access (PDA) Research Group at Penn State University to pursue cross-area research in data management, pervasive/mobile computing, and networking. He is a member of the IEEE.