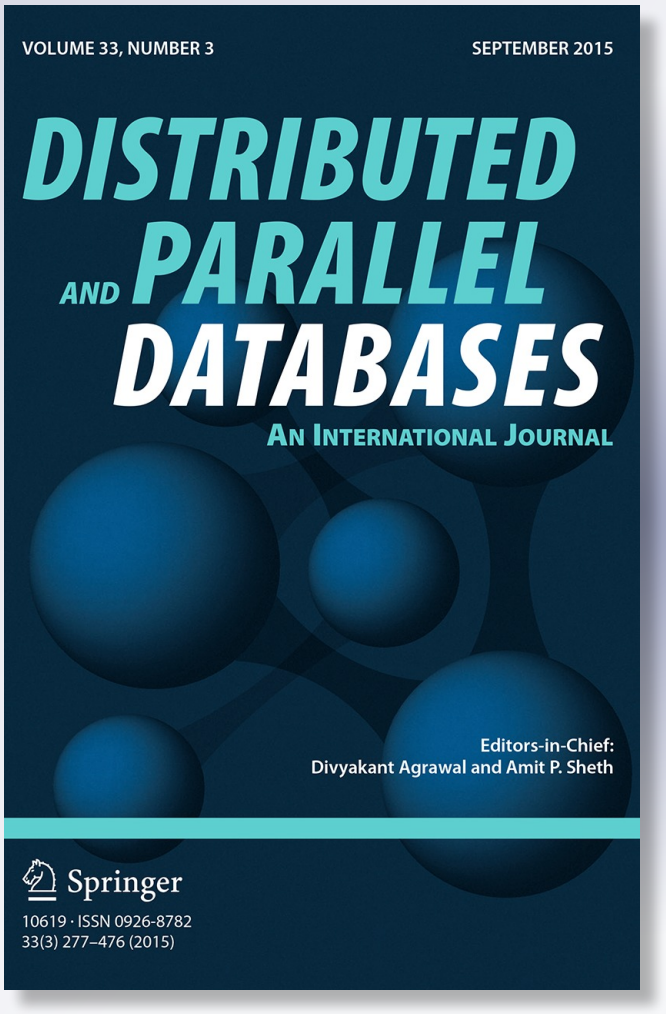
Efficient top-(k,l) range query processing for uncertain data based on multicore architectures

Guoqing Xiao, Kenli Li, Keqin Li & Xu Zhou

Distributed and Parallel Databases
An International Journal

ISSN 0926-8782
Volume 33
Number 3

Distrib Parallel Databases (2015)
33:381-413
DOI 10.1007/s10619-014-7156-8



Springer

Distrib Parallel Databases (2015) 33:381–413
DOI 10.1007/s10619-014-7156-8

Efficient top-(k,l) range query processing for uncertain data based on multicore architectures

Guoqing Xiao · Kenli Li · Keqin Li · Xu Zhou

Published online: 2 October 2014

Abstract Query processing over uncertain data is very important in many applications due to the existence of uncertainty in real-world data. In this paper, we first elaborate a new and important query in the context of an uncertain database, namely uncertain top-(k,l) range (UTR) query, which retrieves l uncertain tuples that are expected to meet score range constraint $[CR_1, CR_2]$ and have the maximum top-k probabilities but no less than a user-specified probability threshold q. In order to enable the UTR query answer faster, we put forward some effective pruning rules to reduce the UTR query space, which are integrated into an efficient UTR query procedure. What's more, to improve the efficiency and effectiveness of the UTR query, a parallel UTR (PUTR) query procedure is presented. Extensive experiments have verified the efficiency and effectiveness of our proposed algorithms. It is worth to notice that, comparing to the UTR query procedure, the PUTR query procedure performs much more efficiently and effectively.

G. Xiao · Kenli Li (✉) · Keqin Li · X. Zhou
College of Information Science and Engineering, Hunan University, Changsha 410082, China
e-mail: lkl@hnu.edu.cn

G. Xiao
e-mail: s12101024@hnu.edu.cn

Keqin Li
e-mail: likq@hnu.edu.cn; lik@newpaltz.edu

X. Zhou
e-mail: happypanda2006@126.com

Kenli Li
National Supercomputing Center in Changsha, Changsha 410082, China

Keqin Li
Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

Keywords Uncertain data · Top-k query · Range query · Parallel optimization

1 Introduction

Due to data noise, data leakage, transmission delay, and inaccuracy or incompletion in measurement etc, uncertain data exist widely in information retrieval, mobile object tracking, Web services, and various other applications. In particular, some large-scale applications, for instance, sensor networks and RFID, can generate a large amount of uncertain data. This has created a need for uncertain data management algorithms and applications [3,10], among which a pivotal technique in this respect is the query processing over uncertain database, such as top-k query [7,12,15–18,20,21,25,26,29]. With the rapid development of data collection methods and the practical applications, the issue of uncertain data query has drawn large amounts of attention in both academia and industry [5].

Top-k query is a classic problem in the area of information retrieval. For a user-defined scoring function and one given query, an algorithm needs to return k objects which have the largest scores [11]. However, in uncertain data management, data records are typically represented by probability distributions rather than deterministic values [3]. Therefore, traditional definite top-k query cannot respond to uncertain top-k query, thereby we have to redefine query semantics of top-k for uncertain query. As for uncertain top-k query, the interaction between score and probability determine the answers. Different combination of the two factors may generate various uncertain query semantics, among which most are based on possible worlds semantics [1,23].

The top-k range reporting problem has been studied in various areas, such as information retrieval, data streams, OLAP, etc. Given a data set D, a score interval $[CR_1, CR_2]$ and an integer k, a query reports, among all the tuples of D in $[CR_1, CR_2]$, the k tuples with the best scores [24]. It finds applications in many situations where people want to ascertain a few best objects, among only a subset of the data set meeting a range predicate. For instance, a customer of a store database may be interested in discovering the k best rated articles whose prices are in an appointed range. Likewise, for promotion purposes, the manager of a company would like to find the k employees with the best performance, among those salesmen whose salaries are in a designated range.

In uncertain data management, the range-based query processing also has attracted recently more and more attention in various practical applications as a result of the uncertainty, such as moving object tracking [13], location-based services [4,22], and computer games [30] etc. For the uncertainty of objects, or privacy reasons, the records we want to query usually locate in a finite range region, such as an interval range. For instance, in the coal mine surveillance application, sensors are deployed in the tunnel to collect data such as atmospheric pressure, density of gas, as well as temperature and humidity. In order to keep the safety of workers, these metrics must maintain in an accredited range, e.g., the temperature is in range [36, 86 °F], humidity in [80, 100 %], and atmospheric pressure in [103, 108 hpa]. The manager would like to report those data in the range with at least 75 % probability so as to grasp the circumstance in the coal mine, which is a top-k range reporting problem with a user-specified probability threshold.

As another example, consider a meteorology system that monitors the temperatures, humidity, UV indexes, etc [27]. In a large number of regions, the corresponding readings are taken by sensors in local areas, and transmitted to a central database periodically. The database content may not exactly reflect the current atmospheric status, e.g., the actual temperature in a region may have changed since it was last measured. A similar inquiry in a meteorology system may "identify those data whose temperatures are in range [75, 80 °F], humidity in [40, 60 %], and UV indexes [4.5, 6] with at least 70 % likelihood".

Although conventional range query in the context of deterministic data set has been well studied [2,24], its solutions cannot apply directly to uncertain data without considering the probabilistic requirements. To the best of our knowledge, there is very little work that has studied range-based uncertain top-k queries with probability threshold. In this paper, we firstly study the uncertain top-(k,l) range query (UTR query) problem, i.e., for a user-specified probability threshold q and an appointed score range constraint $CR = [CR_1, CR_2]$, the query returns l tuples which meet the range constraint CR, i.e., $s(t) \in CR$, and their top-k probabilities are no less than q.

At present, the vast majority of uncertain top-k queries mainly concentrate on the study of serial uncertain top-k query algorithms. As far as we know, very few techniques refer to parallel uncertain top-k query. However, with the explosion of Internet information content, people need to query more and more data sets. This requires top-k query to return results in time. Furthermore, due to the existence of probability dimension, uncertain data query is processed in a possible world space which grows exponentially. Although many query algorithms use pruning, indexing and other heuristics techniques to improve the efficiency, the consumption of time and space cannot be ignored. It is necessary to use more optimized techniques to improve the query efficiency. In this paper, we put forward a parallel implementation on uncertain top-(k,l) range (UTR) query algorithm, denoted by PUTR query, on the basis of a divide and conquer strategy on a multicore architecture. In particular, we use OpenMP [26] to parallelize the UTR query processing algorithm we proposed. To the best of our knowledge, no previous work has studied the PUTR query problem in the context of uncertain databases.

Our contributions made in this paper are summarized as follows.

- We first develop a new and crucial query, i.e., uncertain top-(k,l) range query by taking range query and top-k query into overall consideration in the context of uncertain databases.
- In order to improve the performance of the proposed algorithm, we present some effective pruning techniques to reduce the search space, which are integrated into an efficient UTR query procedure.
- A parallel implementation of UTR query, i.e., PUTR query, based on multicore architecture is proposed for the purpose of improving further the performance of UTR query as the data size increases rapidly.
- Extensive experiments are conducted over both real-world and synthetic data to evaluate the performance of the proposed algorithms. The experimental results show that our query algorithms perform well.

The rest of this paper is organized as follows. Section 2 mainly discuss the existing related studies. Section 3 mainly introduces an uncertain data model and its corresponding possible world semantics, and further gives the formal definitions concerning UTR query processing. Section 4 presents several pruning rules to reduce the UTR query search space. Section 5 presents the parallel optimization of UTR query. Section 6 evaluates the proposed algorithms with experiments. Finally, in Sect. 7, we make a conclusion with directions for future work.

2 Related work

In essence, a range-based uncertain top-k query inherits the characteristics of an uncertain top-k query and a range-based query. As such, we review the existing studies on these two queries in this section.

2.1 Uncertain top-k query processing

While research works on conventional top-k queries are mostly based on some deterministic scoring functions, the new factor of tuple membership probability in uncertain database makes evaluation of probabilistic top-k queries very complicated since the top-k answer set depends not only on the ranking scores of candidate tuples but also their probabilities [27]. As for uncertain databases, there exist all kinds of uncertain top-k query semantics, among which the most influential include U-Top$k$ [25,26], U-kRanks [20,25], PT-k [15–17], Global-Top$k$ [29], Expected Rank [7,18], E-Score Rank [7,18], c-typical-Top$k$ [12], and PTkS [21] etc. U-Top$k$ returns the most probable k-tuple vector with the maximum aggregated probability of being top-k over all possible worlds. U-kRanks returns a list of k tuples such that the ith-ranked tuple has the highest aggregated probability in all possible worlds. These two query algorithm proposed in [25] are inefficient due to lacking of pruning rules with an increasing possible world space. PT-k query returns those tuples whose top-k probabilities across all possible worlds are no less than a given probability threshold q, which makes for approving performance without unfolding all possible worlds. Furthermore, a sampling method is developed to quickly compute an approximation with quality guarantee to the answer set by extracting a small sample of the uncertain dataset. Though the sampling method can lower the accuracy of answers, it can improve efficiency to a large degree. In [29], Zhang etc proposed a Global-Top$k$ query semantic that returns k highest-ranked tuples on the basis of their probabilities of being top-k answer set in all possible worlds. In [7], the expected rank of each tuple over all possible worlds is regarded as the ranking function for obtaining the answer set. E-Score Rank query takes the E-Score of each tuple as the ranking function to find the final answers. In [12], the authors developed c-typical-Top$k$ query which returns a set of k-tuple vectors with the maximum probability of typical scores. Lian and Chen [21] proposed the probabilistic top-k star (PTkS) query, which aims to retrieve k objects in an uncertain database that are "closest" to a static/dynamic query point, considering both distance and probability aspects. In [14], the authors presented the problem of Top-k frequent itemsets mining in sliding

windows. Li etc presented a unified approach to ranking and top-k query processing in probabilistic databases by viewing it as a multi-criteria optimization problem [19].

From these definitions, we can see that a pivotal problem of uncertain top-k query is the calculation of possible world probability. Provided there is no good pruning technology, the performance of query may be comparatively low with a possible world space which grows exponentially. Consequently, it is necessary to exploit some efficient pruning strategies to reduce the computing of top-k probability for improving the performance of algorithm.

2.2 Range-based query processing

Range-based query processing has recently obtained more and more attention in a variety of practical applications due to the uncertainty, such as moving object tracking [13], location-based services [4,22], and computer games [30] etc. For the uncertainty of object, or privacy reasons, the records we want to query usually locate in a finite range region, such as an interval range. Although conventional range query in the context of deterministic dataset has been well studied [2,24], its solutions cannot apply directly to uncertain data without considering the probabilistic requirements. Brodal [2] and Tao [24] developed a static and dynamic structure for the top-k range reporting problem, respectively. In [14], Hu and Lee firstly presented the RkNN solution for rectangular ranges. Lin et al. proposed the first range-based skyline query in LBS [22]. In [6], Cheng et al. firstly put forward probabilistic range query based on one-dimensional space. Tao et al. studied the uncertain range query for arbitrary probability distribution function in multi-dimensional space [27]. Dai et al. [9] and Yiu et al. [28] studied the probabilistic spatial range queries with R-tree for multi-dimensional data in the context of uncertain dataset. R-tree is a good index structure and is efficient for high dimensional data, which can be presented by attribute-level uncertainty. However, in this paper, what we consider is the tuple-level uncertainty which is inefficient indexed by R-tree since it is only one dimension data. To the best of our knowledge, there is very little work that has studied uncertain top-k query problem based on an interval range. In this paper, we propose an uncertain top-k range query based on score attribute range, which retrievals the uncertain database by appointing an interval range.

3 Preliminaries

In this section, we first introduce some fundamental knowledge with respect to an uncertain data model and its corresponding possible world semantics. After that, we will give the formal definitions with regard to uncertain top-(k,l) range query processing.

3.1 Uncertain data model

The fundamental difference between a traditional deterministic database and an uncertain database is that an uncertain relation represents a set of possible relation instances, rather than a single one [3]. Suppose an uncertain database DB, which is composed of a set of n tuples t_i ($1 \leq i \leq n$). The uncertainty of every tuple t_i in the uncertain database DB is mainly represented by a confidence, i.e., its existence probability $P(t_i)$ in DB. For a given score ranking function $s(t)$, the score of each tuple t_i is denoted by $s(t_i)$. In fact, many uncertain data processing, including top-k query processing, pay attention to mainly two types of uncertainty, i.e., tuple-level uncertainty and attribute-level uncertainty. For tuple-level uncertainty, every tuple is uncertain while its attribute value (i.e. score) is deterministic. For attribute-level uncertainty, every tuple is deterministic while its attribute value is uncertain, and every attribute value corresponds to a probability. In a probabilistic database, there may exist some generation rules between tuples, such as exclusion or coexistence, but in most cases tuples are independent of each other. In this paper, we only consider tuple-level uncertainty with all tuples mutually independent. In practice, almost all tuples are mutually independent in the context of tuple-level uncertainty.

There exist many works on modeling uncertain data. One of the most popular is the model based on possible world semantics [1,23], where an uncertain database is regarded as a set of possible world instances associated with their probabilities. Each possible world W is a subset of uncertain database tuples, and the set of all worlds is denoted by the possible world space Ω. The probability of each world is computed as the joint probability of the existence of the world's tuples and the absence of all other database tuples. Since all tuples are mutually independent, we can obtain

$$P(W) = \prod_{t \in W} P(t) \prod_{t \notin W} \overline{P(t)}, \tag{1}$$

where $\overline{P(t)} = 1 - P(t)$, and $\sum_{W \in \Omega} P(W) = 1$.

3.2 Problem definition

With the aforementioned introductions and possible world semantics, in this subsection, we will put forward the query semantics concerning uncertain top-(k,l) query based on a given probability threshold q and Refs. [16,25,29].

Definition 1 (*Score dominating*) Let $s(t)$ be a score ranking function. For arbitrary two tuples t_i and t_j, if $s(t_i) > s(t_j)$, then $t_i \succ_s t_j$.

Definition 2 (*Top-k probability* [16,29]) Let DB be an uncertain database with possible world space Ω, k be a positive integer, $s(t)$ be a score ranking function, and $\text{Top}_k(W)$ be a set of k tuples in the front of possible world W on the basis of scoring function $s(t)$. Then the probability of any tuple t in DB $P_{\text{top-}k}(t, DB)$ can be defined as the summation of the probabilities of all possible worlds whose *top-k* answer set $\text{Top}_k(W)$ contains t , i.e.,

$$P_{\text{top-}k}(t, DB) = \sum_{W \in \Omega,\, t \in \text{Top}_k(W)} P(W). \tag{2}$$

Note that top-k answer sets may be of cardinality less than k for some possible worlds. We call such possible worlds as small worlds [29].

Definition 3 (*Uncertain top-(k,l) range query, UTR query*). Let DB be an uncertain database, l be a positive integral number, CR be a given score range constraint $[CR_1, CR_2]$, q be a user-specified probability threshold, and $P_{\text{top-}k}(t, DB)$ be the top-k probability of tuple t in DB. Then the top-(k,l) query over uncertain database DB returns l tuples which meet constraint CR and have top-k probabilities at least q, i.e.,

$$\{t | s(t) \in CR, P_{\text{top-}k}(t, DB) \geq q\}, \tag{3}$$

and

$$|\{t | s(t) \in CR, P_{\text{top-}k}(t, DB) \geq q\}| = l. \tag{4}$$

3.3 Compute top-k probability

Although there exist some differences between the semantics of top-(k,l) query and global top-k query in [29], we find that they have the same computing method with respect to top-k probability for any tuple t. That is, for an uncertain database DB with cardinality n, a positive integer k, a score ranking function $s(t)$, and $t_1 \succ_s t_2 \succ_s \cdots \succ_s t_{n-1} \succ_s t_n$, the top-$k$ probability of tuple t_i holds the following recursion,

$$P_{\text{top-}k}(t_i, DB) = \begin{cases} P(t_i), & 1 \leq i \leq k; \\ [P_{\text{top-}k}(t_{i-1}, DB) \cdot \frac{\overline{P(t_{i-1})}}{P(t_{i-1})} & \\ +P_{\text{top-}(k-1)}(t_{i-1}, DB)] \cdot P(t_i), & i > k; \end{cases} \tag{5}$$

where $\overline{P(t_{i-1})} = 1 - P(t_{i-1})$.

For the proof, the reader is referred to Appendix B in [29]. The commonly used symbols and their descriptions are summarized in Table 1.

4 Pruning rules

On the basis of semantics of uncertain top-(k,l) query, we can know that the calculation of top-k is very large with an exponentially growing world space. Therefore, it is necessary to put forward some pruning techniques for reducing the computing of top-k. In this section, we will introduce several theorems and lemmas on effective pruning rules.

What we care about is those tuples that satisfy the score range constraint. So firstly we can remove the tuples which do not meet the score constraint before calculating the probability of top-k of tuples.

Table 1 Frequently used symbols and their descriptions

Symbols	Descriptions
DB	An uncertain database
$\|DB\|$	The cardinality of database DB
SDB	An uncertain sub-dataset
t	A tuple in DB
$P(t)$	Tuple t existing probability in DB
W	Some possible world
$P(W)$	The probability of W
$s(t)$	A score ranking function
$\text{Top}_k(W)$	The set of k tuples in the front of W based on $s(t)$
Ω	Possible world space
k, l	Two positive integers
q	A user-specified probability threshold
CR=[CR_1,CR_2]	The score range constraint
$P_{\text{top-}k}(t, DB)$	The top-k probability of tuple t in DB
$P_{\text{Gtop-}k}(t, DB)$	The global top-k probability of tuple t in DB
$P_{\text{Ltop-}k}(t, SDB)$	The local top-k probability of tuple t in SDB
p	The number of cores (threads)
L_i	The local top-(k,l) answer set
Q	The global top-(k,l) result set
PR	The pruning ratio

Lemma 1 (Pruning Rule 1) *Let $s(t)$ be a score ranking function, CR be a score interval range, DB be an uncertain database. For any tuple $t \in DB$, if $s(t) \notin CR$, then we can remove t immediately from the DB without calculating its top-k probability.* □

Theorem 1 *The top-k probability of tuple t in an uncertain database DB, denoted by $P_{top\text{-}k}(t, DB)$, is at most its presence probability $P(t)$, i.e., $P_{top\text{-}k}(t, DB) \leq P(t)$.* □

The conclusion is obvious.

Lemma 2 (Pruning Rule 2) *Let q be a user-specified probability threshold, $P(t)$ be the presence probability of tuple t, $P_{top\text{-}k}(t, DB)$ be t's top-k probability. If $P(t) < q$, then t can be excluded immediately from the DB without computing $P_{top\text{-}k}(t, DB)$.* □

According to Lemmas 1 and 2, before the calculation of top-k probability of a tuple, we can prune some tuples potentially based on a given score range constraint CR and a probability threshold q, respectively. They aim at decreasing the number of the uncertain data set. The following Lemma 3 can remove further some tuples from the uncertain data set.

Lemma 3 (Pruning Rule 3) *Given an uncertain database DB with cardinality n, and* $t_1 \succ_s t_2 \succ_s \cdots \succ_s t_n$. *Let* DB_{t_i} *be the subset* $\{t_1, t_2, \cdots, t_i\}$, $P(DB_{t_i}, j)$ *denote the probability of any* j *tuples appearing in the set* DB_{t_i}. *If* $\sum_{j=0}^{k-1} P(DB_{t_i}, j) < q$, *then we can prune the tuples which rank lower than* t_i, *i.e., for all* $1 \leq m \leq n-i, m \in N_+$, $P_{top\text{-}k}(t_{i+m}, DB) < q$, *where* q *is a user-specified probability threshold.*

Proof It is clear that

$$\sum_{j=0}^{k-1} P(DB_{t_{i-1}}, j) = \sum_{W \in \Omega,\, |t_s | t_s \in W \wedge t_s \succ_s t_i| \leq k-1} P(W), \tag{6}$$

holds. Based on the proof of Theorem 1, we know

$$P_{\text{top-}k}(t_i, DB) = P(t_i) \sum_{\substack{W \in \Omega \\ |t_s | t_s \in W \wedge t_s \succ_s t_i| \leq k-1}} P(W). \tag{7}$$

As a result, we can obtain

$$P_{\text{top-}k}(t_i, DB) = P(t_i) \sum_{j=0}^{k-1} P(DB_{t_{i-1}}, j). \tag{8}$$

Then, for all $1 \leq m \leq n-i, m \in N_+$

$$P_{\text{top-}k}(t_{i+m}, DB) = P(t_{i+m}) \sum_{j=0}^{k-1} P(DB_{t_{i+m-1}}, j). \tag{9}$$

Next, we can get $P_{\text{top-}k}(t_{i+m}, DB) < q$ by induction.

(1) Suppose $m = 1$, then

$$\sum_{j=0}^{k-1} P(DB_{t_{i+m-1}}, j) = \sum_{j=0}^{k-1} P(DB_{t_i}, j) < q. \tag{10}$$

As a result of $0 \leq P(t_{i+m}) \leq 1$, we know that $P_{\text{top-}k}(t_{i+m}, DB) < q$ holds.

(2) Suppose $1 < m \leq n-i, m \in N_+$. Without loss of generality, we assume that $\sum_{j=0}^{k-1} P(DB_{t_{i+l-1}}, j) < q$ when m=l. Then, for m=l+1, we can obtain

$$\begin{aligned} \sum_{j=0}^{k-1} P(DB_{t_{i+m-1}}, j) &= \sum_{j=0}^{k-1} P(DB_{t_{i+l}}, j) \\ &= P(t_{i+l}) \sum_{j=1}^{k-1} P(DB_{t_{i+l-1}}, j-1) \end{aligned}$$

$$
\begin{aligned}
&+ (1 - P(t_{i+l})) \sum_{j=0}^{k-1} P(DB_{t_{i+l-1}}, j) \\
&= P(t_{i+l}) \sum_{j=0}^{k-2} P(DB_{t_{i+l-1}}, j) \\
&\quad + (1 - P(t_{i+l})) \sum_{j=0}^{k-1} P(DB_{t_{i+l-1}}, j) \\
&\leq P(t_{i+l}) \sum_{j=0}^{k-1} P(DB_{t_{i+l-1}}, j) \\
&\quad + (1 - P(t_{i+l})) \sum_{j=0}^{k-1} P(DB_{t_{i+l-1}}, j) \\
&= P(t_{i+l}) \sum_{j=0}^{k-1} x_{t_{i+l-1},j} + (1 - P(t_{i+l})) \sum_{j=0}^{k-1} x_{t_{i+l-1},j} \\
&= \sum_{j=0}^{k-1} P(DB_{t_{i+l-1}}, j) < q. \qquad (11)
\end{aligned}
$$

Consequently, we can know that $\sum_{j=0}^{k-1} P(DB_{t_{i+m-1}}, j) < q$ also holds when $m{=}l{+}1$. As a result of $0 \leq P(t_{i+m}) \leq 1$, we can get $P_{\text{top-}k}(t_{i+m}, DB) < q$, too. In conclusion, for a given probability threshold q, if $\sum_{j=0}^{k-1} P(DB_{t_i}, j) < q$, then for $\forall\ 1 \leq m \leq n - i, m \in N_+$, $P_{\text{top-}k}(t_{i+m}, DB) < q$ holds. Accordingly, we can prune the tuples which rank lower than t_i. □

According to the third pruning technique, we can obtain a compact set, i.e., an upper bound on the answer set. In subsequent experiments, we show that the pruning technology has excellent efficiency and effectiveness with the pruning ratio at least 99 % if the size of the data set is large.

The three pruning techniques aforementioned all aim at cutting down the size of the uncertain data set for reducing the query search space. However, these pruning strategies do not consider the properties of top-k probability of tuples. One important innovation of this paper is the demonstration of the mathematical properties of top-k probability of tuples. In Sect. 5, we will give out several other pruning rules based on these properties.

5 PUTR query

As what mentioned before, at present very few techniques refer to parallel uncertain top-k query. And with the explosion of Internet information content, people need to query more and more data sets and return results timely. Consequently, it is necessary to propose a high performance query algorithm for uncertain top-k. In this section,

we first introduce briefly some basic knowledge on OpenMP programming and give several theorems and lemmas concerning pruning rules based on the properties of top-k probability. After that, we present the parallelization on uncertain top-(k,l) range (i.e., PUTR) query processing based on divide and conquer strategy on the multicore architecture.

With the development of single-core to multi-core/many-core in processors, programmers are required to find the parallelism in a program and to achieve them explicitly. In this paper, we tackle the parallelization problem of UTR query on the basis of OpenMP parallel programming platform. The OpenMP [8] (Open Multiprocessing) is an API that multi-platform shared memory multi-processing programming in C, C++, and Fortran, on most processor architectures and operating systems. It consists of a set of compiler directives, library routines, and environment variables that influence running time behavior. It is an implementation of multithreading, a method of parallelizing whereby a master thread (a series of instructions executed consecutively) forks a specified number of slave threads and a task is divided among them, then the threads run concurrently. More detailed information about the usage of OpenMP can be found on the website.[1]

Definition 4 (*Global/local top-k probability*) If tuple t is a member of the final top-(k,l) answer set, then we call its top-k probability as global top-k probability in DB, denoted by $P_{\text{Gtop-}k}(t, DB)$. On the other hand, if t is a member of the local top-(k,l) answer set in a sub-database of SDB, then we call its top-k probability as local top-k probability in the sub-database SDB, denoted by $P_{\text{Ltop-}k}(t, DB)$.

Theorem 2 *Given an uncertain database DB with cardinality n, and* $t_1 \succ_s t_2 \succ_s \cdots \succ_s t_n$. *Then the top-k probability of tuple* t_i *in the DB* $P_{top\text{-}k}(t_i, DB)$ *increases with k, but no more than* $P(t_i)$.

The conclusion is obvious.

Lemma 4 (Pruning Rule 4) *Given a probability threshold q, an uncertain database DB with cardinality n, and* $t_1 \succ_s t_2 \succ_s \cdots \succ_s t_n$. *Then there exists a lower bound LB of top-k probability of tuple* t_i, *i.e.,* $P(t_i)\prod_{t_j \succ_s t_i}(1 - P(t_j))$, *such that we can add* t_i *into the local top-(k,l) answer set without computing its complete top-k probability when* $LB \geq q$. □

Note that, in the processing of UTR query, we may put directly the tuples whose lower bound LB are greater than or equal to q into final answer set for the purpose of making some comparisons with the PUTR query processing. So the number of tuples in top-(k,l) answer set of the UTR query may be larger than l on account of adding some tuples whose LB are no less than q. But in either case, the answer set of the UTR query always contains the PUTR query results. It can turn out to be true by our theoretical analysis and experimental results in the following sections.

Theorem 3 *Given an uncertain database DB with cardinality n, and* $t_1 \succ_s t_2 \succ_s \cdots \succ_s t_n$. *Then the global top-k probability of tuple* t_i $P_{Gtop\text{-}k}(t_i, DB)$ *is less than its local top-k probability* $P_{Ltop\text{-}k}(t_i, DB)$, *i.e.,* $P_{Gtop\text{-}k}(t_i, DB) \leq P_{Ltop\text{-}k}(t_i, DB)$.

[1] http://openmp.org

Proof In order to prove the theorem, we can study it by two cases below.

Case 1 If the tuples from t_j $(1 \leq j \leq i \ and \ i - j \geq k)$ to t_i are divided just right into certain sub-database SDB, then any tuple with score lower than the score of t_i does not have any influence on the top-k probability in that its presence in a possible world will not affect the presence of t_i in the top-k answer set of that world. Thus, in this case we can obtain

$$P_{\text{Gtop-}k}(t_i, DB) = P_{\text{Ltop-}k}(t_i, DB). \tag{12}$$

Case 2 If tuple t_i is divided into some sub-database SDB with maximum score, then on the basis of Equation (4), we can obtain

$$P_{\text{Ltop-}k}(t_i, DB) = P(t_i). \tag{13}$$

And according to Theorem 1, we have

$$P_{\text{Gtop-}k}(t_i, DB) \leq P(t_i). \tag{14}$$

Therefore,

$$P_{\text{Gtop-}k}(t_i, DB) \leq P_{\text{Ltop-}k}(t_i, DB). \tag{15}$$

Consequently, combining Case 1 and Case 2, for any tuple t_i, we can obtain

$$P_{\text{Gtop-}k}(t_i, DB) \leq P_{\text{Ltop-}k}(t_i, DB). \tag{16}$$

This completes the proof of Theorem 3. □

Lemma 5 (Pruning Rule 5) *Let q be a probability threshold, $P_{Ltop\text{-}k}(t_i, DB)$ be local top-k probability of tuple t. If $P_{Ltop\text{-}k}(t_i, DB) < q$, then t cannot be a member of the global top-k answer set.*

Proof As a result of $P_{\text{Ltop-}k}(t_i, DB) < q$, in accordance to Theorem 3, we can obtain $P_{\text{Gtop-}k}(t_i, DB) < q$. However, what we need are those tuples whose top-k probability no less than q. Therefore, tuple t can be removed immediately from the SDB without handling it in the merge process. Accordingly, it can not be a member of the global top-k answer set. □

In summary, we can decrease efficiently the searching space by removing some tuples based on these pruning methods we proposed, such as the score range constraint pruning (Pruning Rule 1, see Sect. 4), the probability threshold pruning (Pruning Rule 2, see Sect. 4), the upper bound of answer set pruning (Pruning Rule 3, see Sect. 4), the top-k probability lower bound pruning (Pruning Rule 4), and local top-k probability pruning (Pruning Rule 5).

Now we elaborate the parallel uncertain top-(k,l) range (i.e., PUTR) query processing algorithm. The algorithm is presented in Algorithm 1. Given an uncertain database DB with cardinality n, a processor with p cores, the PUTR query algorithm mainly contains the following two steps.

Step 1 (*Divide*)

(1) According to the number of threads p, we divide the uncertain database DB into p sub-databases SDB_i with cardinality n_i in tuple ID, where $1 \leq i \leq p$ and $\sum_{1 \leq i \leq p} n_i = n$, and the size of each sub-database is $\lceil \frac{n}{p} \rceil$ (line 4);
For example, given an uncertain database $DB=\{t_1, t_2, \ldots, t_n\}$, the subset $SDB_1=\{t_1, t_2, \ldots, t_{\lceil \frac{n}{p} \rceil}\}$ can be allocated into Thread P_0 as the first sub-database, the subset $SDB_i=\{t_{(i-1)\lceil \frac{n}{p} \rceil+1}, t_{(i-1)\lceil \frac{n}{p} \rceil+2}, \ldots, t_{i\lceil \frac{n}{p} \rceil}\}$ can be allocated into Thread P_{i-1} as the ith sub-database, the subset $SDB_p=\{t_{(p-1)\lceil \frac{n}{p} \rceil+1}, t_{(p-1)\lceil \frac{n}{p} \rceil+2}, \ldots, t_n\}$ can be allocated into Thread P_{p-1} as the (p-1)th sub-database.
(2) For each thread P_i, we first prune SDB_i based on the score range pruning and probability threshold pruning (Pruning Rules 1 and 2), then sort for remaining tuples in SDB_i in the light of the decreasing order based on the score ranking function $s(t)$ (lines 6–11);
(3) Apply the upper bound of answer set pruning (Pruning Rule 3) for every sub-dataset (lines 12–14);
(4) Apply the lower bound of top-k probability pruning (Pruning Rule 4) (lines 15–18);
(5) Compute local top-k probability of tuples, use local top-k probability pruning (Pruning Rule 5), and obtain eventually local top-(k,l) result sets L_i (lines 19–30).

Step 2 (*Merge*)

(1) The master thread P_0 gathers local answer sets L_i and combines them into a new dataset (line 31);
(2) Sort the tuples in the new dataset in accordance with the decreasing order based on the scoring function $s(t)$ (line 32);
(3) Repeat procedure (3) in Step 1 (lines 33–35);
(4) Compute global top-k probability of tuples and select l tuples which satisfy the definitions and requirements (lines 36–37);
(5) Return the global top-(k,l) answer set Q (line 38).

Figure 1 illustrates the fundamental framework of PUTR query processing algorithm.

An illustrative example In a sensor network deployed in a "smart uniform" of the US military, each sensor monitors pivotal biological parameters for determining the status of physiology of a soldier [12]. Each sensor reading comes from a confidence value, which shows the probability that the reading is valid. In the services of patient-centered, different patients can need various medical attention. Based on these information, the military staff may want to find the top-k soldiers for a range of medical attention and allocate the appropriate resources to deliver to the battlefield. Then the staff need to compute the top-k probability of every soldiers. Table 2 shows eight tuples that were reported around the same time and thus estimating the same value for each soldier, and illustrates the process of parallelization of top-(2,2) query of the eight mutually independent tuples on a dual-core processor.

The thread P_0 deals with the first four tuples while the P_1 processes the remaining four tuples based on their ID. At first, we can prune tuples t_4 and t_6 on the basis of

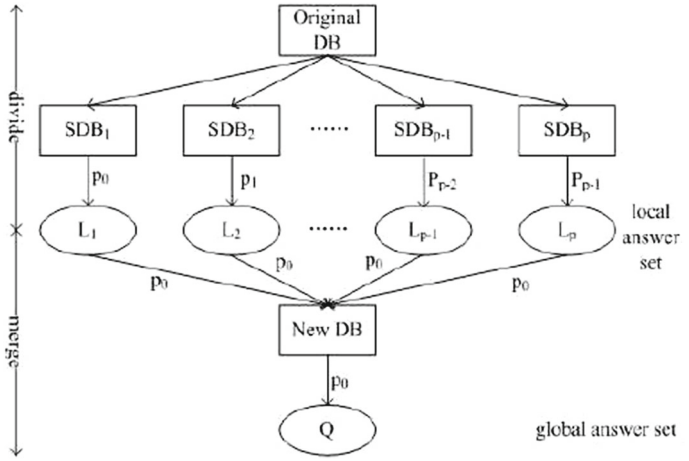


Fig. 1 The fundamental framework of PUTR query

Table 2 An example of 8 tuples of PUTR query

Tuple ID	Soldier ID	Time	Medical needs (score)	Prob.		
t_1	134	13:40	25	0.4		
t_2	135	13:40	18	0.6		
t_3	136	13:39	17	0.4		
t_4	137	13:41	20	0.1		
t_5	138	13:39	19	0.5		
t_6	139	13:40	9	0.6		
t_7	140	13:41	15	0.5		
t_8	141	13:41	20	0.4		
Thread	Tuple ID	Score	$\sum_{j=0}^{1} P(DB_{t_i}, j)$	LB	$P_{\text{Ltop-2}}(t, DB)$	L_i
P_0	t_1	25	1	0.4	–	$L_0 = \{t_1, t_2, t_3\}$
	t_2	18	0.76	0.36	0.6	
	t_3	17	–	0.096	0.4	
P_1	t_8	20	1	0.4	–	$L_1 = \{t_8, t_5, t_7\}$
	t_5	19	0.8	0.3	0.5	
	t_7	15	–	0.12	0.5	
Tuple ID	Score	Prob.	$\sum_{j=0}^{1} P(DB_{t_i}, j)$	$P_{\text{Gtop-2}}(t, DB)$	Q	
t_1	25	0.4	1	0.4	$\{t_1, t_8\}$	
t_8	20	0.4	0.84	0.4		
t_5	19	0.5	0.6	0.24		
t_2	18	0.6	$0.348 < 0.4$	0.252		
t_3	17	0.4	–	–		
t_7	15	0.5	–	–		

Pruning Rules 1 and 2 for a given probability threshold 0.4 and a score range CR=[15, 25]. In addition, we can put t_1 and t_8 immediately into local top-(2,2) answer set without computing their local top-2 probability because their lower bound LB are all greater or equal to than probability threshold 0.4 (Pruning Rule 4). Moreover, in the process of merging, it is easy to know $\sum_{j=0}^{1} P(DB_{t_1}, j) = 1 > 0.4$, $\sum_{j=0}^{1} P(DB_{t_8}, j) = 0.84 > 0.4$, $\sum_{j=0}^{1} P(DB_{t_5}, j) = 0.6 > 0.4$, $\sum_{j=0}^{1} P(DB_{t_2}, j) = 0.348 < 0.4$, so we can prune t_3 and t_7 directly based on Pruning Rule 3 without calculating their global top-2 probability. From the figure, we can see that the thread p_0 returns t_1, t_2 and t_3 as its local top-(2,2) answer set, and the p_1 returns t_8, t_5 and t_7 as its local top-(2,2) answer set similarly. After that, as shown in the figure, we obtain global top-(2,2) results, i.e., $\{t_1, t_8\}$, on the basis of their global top-2 probability. That is, for a given medical attention range, soldiers 1 and 8 need the treatment the most.

Theorem 4 *The algorithm of parallel uncertain top-(k,l) range query processing, i.e., PUTR query processing, can report the global optimal l tuples answer set.*

Proof Let A_l be the global top-(k,l) answer set, $\text{Sub}_i A_l$ be the local top-(k,l) answer set in sub-database SDB_i, $\text{Sub}_i \overline{A_l}$ be the intersection of A_l and $\text{Sub}_i A_l$, i.e.,

$$\text{Sub}_i \overline{A_l} = A_l \cap \text{Sub}_i A_l, \tag{17}$$

where $1 \leq i \leq p$.

First of all, we can get $\text{Sub}_i \overline{A_l} \subseteq \text{Sub}_i A_l$ by contradiction. Suppose there exists a tuple t which belongs to $\text{Sub}_i \overline{A_l}$ but not $\text{Sub}_i A_l$. Let U be the set of tuples in global data space DB which their top-k probabilities are no less than that of t, i.e.,

$$U = \{t' | t' \in DB, P_{\text{top-}k}(t') \geq P_{\text{top-}k}(t)\}. \tag{18}$$

Let $\text{Sub}_i U$ be the set of tuples in local data space SDB_i which their top-k probabilities are no less than that of t, i.e.,

$$\text{Sub}_i U = \{t' | t' \in SDB_i, P_{\text{top-}k}(t') \geq P_{\text{top-}k}(t)\}. \tag{19}$$

Evidently,

$$\text{Sub}_i U = U \cap SDB_i \subseteq U. \tag{20}$$

According to the definition of top-(k,l) query, we can have

$$\text{Sub}_i A_l \subseteq \text{Sub}_i U. \tag{21}$$

Consequently, we can obtain

$$|U| \geq |\text{Sub}_i U| \geq |\text{Sub}_i A_l| > l. \tag{22}$$

Algorithm 1 . PUTR Query Processing

Input:

An uncertain database DB with the cardinality n, a score ranking function $s(t)$, a user-specified probability threshold q, a score range CR, two integral numbers k and l, p threads

Output:

PUTR query answer set Q

/***Divide***/

1: initialize the top-(k,l) answer set $Q \leftarrow \emptyset$;
2: initialize the local top-(k,l) answer set $L_i \leftarrow \emptyset$ $(1 \leq i \leq p)$;
3: initialize the local candidate top-(k,l) answer set $S_i \leftarrow \emptyset$ $(1 \leq i \leq p)$;
4: divide database DB into p sub-databases SDB_i with cardinality n_i in ID, where $1 \leq i \leq p$ and $\sum_{1 \leq i \leq p} n_i = n$;
5: **for** each thread P_i and sub-database SDB_i, where $1 \leq i \leq p$, **par– do**
6: **for** each tuple t_{ij} in SDB_i, where $1 \leq j \leq n_i$ **do**
7: **if** ($s(t_{ij}) \notin CR$ or $P(t_{ij}) < q$) **then**
8: remove t_{ij} from the SDB_i;
9: **end if**
10: **end for**
11: sort for remaining tuples in SDB_i in the decreasing order of the scoring function $s(t)$;
12: **if** $\sum_{j=0}^{k-1} P(DB_{t_i}, j) < q$ **then**
13: prune the tuples which rank lower than t_{ij} from the SDB_i;
14: **end if**
15: compute the lower bound LB of local top-k probability of tuple t_{ij} in SDB_i;
16: **if** $LB \geq q$ **then**
17: put t_{ij} into into local top-(k,l) answer set L_i, where $1 \leq i \leq p$;
18: **end if**
19: compute local top-k probability of tuple t_{ij} $P_{Ltop-k}(t_{ij}, SDB_i)$ for the remaining tuples in the SDB_i using Algorithm 2;
20: **if** $P_{Ltop-k}(t_{ij}, SDB_i) < q$ **then**
21: remove t_{ij} from the SDB_i;
22: **else**
23: put t_{ij} into local candidate answer set S_i;
24: **end if**
25: **if** $|S_i| < L$ **then**
26: $L_i \leftarrow S_i$;
27: **else**
28: pick out l tuples with the maximum $P_{Ltop-k}(t_{ij}, SDB_i)$ from S_i and insert them into the local top-(k,l) answer set L_i;
29: **end if**
30: **end for**

/***Merge***/

31: the master thread P_0 collects local top-(k,l) answer set L_i, and groups them into a new database DB;
32: sort the new dataset DB in the decreasing order of the scoring function $s(t)$;
33: **if** $\sum_{j=0}^{k-1} P(DB_{t_i}, j) < q$ **then**
34: prune the tuples which rank lower than t_i from the DB;
35: **end if**
36: compute top-k probability of tuple t $P_{top-k}(t, DB)$ in DB using Algorithm 2;
37: select l tuples which have the maximum top-k probabilities but no less than q, and put them into the global answer set Q;
38: **return** Q.

Algorithm 2 . Compute top-k probability of tuples in uncertain database

Input:

An uncertain database DB with the cardinality n is sorted in the decreasing order based on the score ranking function $s(t)$, a user-specified probability threshold q, two integral numbers k and l

Output:

The dynamic programming table $q(0...k;1...|DB|)$

```
1: for i = 1 to |DB| do
2:     q(0,i) = 0;
3: end for
4: for ktemp = 1 to k do
5:     q(ktemp,1) = P(t_1);
6: end for
7: for i = 2 to |DB| do
8:     for ktemp = 1 to k do
9:         q(ktemp,i) = P(t_i)q(ktemp,i-1)(1-P(t_{i-1}))/P(t_{i-1}) + q(ktemp-1,i-1);
10:    end for
11: end for
12: return q(0...k;1...|DB|).
```

This leads to $t \notin A_l$ which contracts our assumption.

In addition, for the reason that A_l is the set of the optimal l tuples in global space DB, and the following holds,

$$A_l \subseteq \cup_{1 \le i \le p} \mathrm{Sub}_i A_l. \tag{23}$$

Thus, for arbitrary a tuple $t \in A_l$, it certainly belongs to the set consisted of the optimal l tuples from the set $\cup_{1 \le i \le p} \mathrm{Sub}_i A_l$. Consequently, we can obtain global top-(k,l) answer set by sorting for the set $\cup_{1 \le i \le p} \mathrm{Sub}_i A_l$ based on their global top-k probabilities. □

Theorem 5 *The time complexity of PUTR query processing algorithm is* $max[O(n/p), O(n_1^2/p^2), O(kn_2/p), O((n_2/p)\log(n_2/p)), O(p^2(l+x)^2)]$*, where, n, n_1, n_2 are the cardinality of original uncertain database, compact database based on score value constraint pruning and probability threshold pruning, and dataset on the grounds of the upper bound of answer set pruning and top-k probability lower bound pruning, respectively. p is the number of cores (threads), x is the maximum number of tuples got on the basis of probability lower bound pruning in the process of local execution.*

Proof The time of PUTR query processing algorithm mainly includes two parts, that is, the local processing time and the merging processing time. It is easy to know that the time complexity of merging is $O(p^2(l+x)^2)$, where p is the number of cores (threads), x is the maximum number of tuples got on the basis of probability lower bound pruning in the process of local execution. On the other hand, for given p threads and the dataset with cardinality n, each thread concurrently deals with $\lfloor n/p \rfloor$ data with the same program. Therefore, the time complexity of local top-(k,l) processing is $max[O(n/p), O(n_1^2/p^2), O(kn_2/p), O((n_2/p)\log(n_2/p))]$, where n, n_1 and n_2 as

shown in Theorem 5. Consequently, the time complexity of PUTR query processing algorithm is $max[O(n/p), O(n_1^2/p^2), O(kn_2/p), O((n_2/p)\log(n_2/p)), O(p^2(l+x)^2)]$.

Generally in practice, the values of p and l should typically be far much less than n, n_1 and n_2. So the time complexity of the algorithm may be $max[O(n/p), O(n_1^2/p^2), O(kn_2/p), O((n_2/p)\log(n_2/p))]$ when n, n_1 and n_2 are very big. □

6 Experimental evaluation

In this section, we demonstrate the efficiency and effectiveness of UTR query and PUTR query processing algorithms through a series of simulations over both real-world and synthetic data. We take query running time, speedup, and pruning ratio PR (as defined hereinafter) as the primary performance metric under various parameters. All experiments were run on a PC with a 2.13 GHz Intel® Xeon® CPU E5506, 2 Quad Cores CPU with 8 GB of main memory, and a 1 TB hard disk, running Windows Win7 64 bit Operating System. Our algorithms were implemented in Microsoft Visual Studio 2010.

Definition 5 (*Pruning Ratio, PR*) Let NUM_{bef} be the number of data set before pruning, NUM_{aft} be the size of the data set after pruning .We call

$$PR = \frac{NUM_{\text{bef}} - NUM_{\text{aft}}}{NUM_{\text{bef}}} \tag{24}$$

pruning ratio, denoted as PR. It represents the efficiency of pruning techniques.

6.1 Results on the real-world data set

We use the International Ice Patrol (IIP) Iceberg Sightings Database[2] which contains 13,095 tuples to evaluate the efficiency and effectiveness of top-k queries on uncertain data in real-world applications. This data set was used in previous works on ranking queries in uncertain data [15–17]. We apply our UTR query and PUTR query on the uncertain data set by setting $k = l = 10$, $q = 0.3$ and CR = [100,500]. The ranking order is the number of days of iceberg drift descending order. The detailed information about the result set of UTR query and PUTR query is showed in Tables 3 and 4, including attribute scores (drifted days), presence probabilities, top-10 probabilities and the corresponding ranks.

As shown in Table 3, the UTR query returns a set of 12 tuples $\{t_{6903}, t_{3610}, t_{3612}, t_{6928}, t_{4174}, t_{4020}, t_{8313}, t_{8412}, t_{8411}, t_{8409}, t_{8410}, t_{8408}\}$ as the top-(10,10) answer set, among which t_{6903} and t_{3610} (as shown in bold) are obtained by probability lower bound pruning (Pruning Rule 4). Table 4 illustrates the top-(10,10) result set, i.e., $\{t_{6903}, t_{3612}, t_{6928}, t_{4174}, t_{4020}, t_{8313}, t_{8412}, t_{8411}, t_{8409}, t_{8410}\}$, of PUTR query. From the results, we can see that the PUTR query results are included in the UTR query

[2] http://nsidc.org/data/g00807.html

Table 3 UTR query answer set in the IIP_2009 database

Tuples	$\mathbf{t_{6903}}$	$\mathbf{t_{3610}}$	t_{3612}	t_{6928}	t_{4174}	t_{4020}	t_{8313}	t_{8412}	t_{8411}	t_{8409}	t_{8410}	t_{8408}
Drifted days	**500**	**500**	495	488.7	439.5	427.6	423.5	455.5	435.2	431.6	431	430.9
Presence prob.	**0.8**	**0.6**	0.8	0.8	0.8	0.8	0.8	0.7	0.7	0.7	0.7	0.7
Top-10 prob.	–	–	0.8	0.8	0.8	0.8	0.8	0.7	0.7	0.7	0.7	0.7
Rank	**1**	**2**	3	4	5	6	7	8	9	10	11	12

Table 4 PUTR query answer set in the IIP_2009 database

Tuples	t_{6903}	t_{3612}	t_{6928}	t_{4174}	t_{4020}	t_{8313}	t_{8412}	t_{8411}	t_{8409}	t_{8410}	$\mathbf{t_{3610}}$	$\mathbf{t_{8408}}$
Drifted days	500	495	488.7	439.5	427.6	423.5	455.5	435.2	431.6	431	**500**	**430.9**
Presence prob.	0.8	0.8	0.8	0.8	0.8	0.8	0.7	0.7	0.7	0.7	**0.6**	**0.7**
Top-10 prob.	0.8	0.8	0.8	0.8	0.8	0.762236	0.7	0.7	0.7	0.7	**0.6**	**0.7**
Rank	1	3	4	5	6	7	8	9	10	11	**2**	**12**

Table 5 PT-k/Global-Top$k$ queries answer set over the IIP_2009 database

Tuples	t_{6903}	t_{3612}	t_{6928}	t_{4174}	t_{4020}	t_{8412}	t_{8411}	t_{8409}	t_{8410}	t_{8408}	$\mathbf{t_{3610}}$
Drifted days	500	495	488.7	439.5	427.6	455.5	435.2	431.6	431	430.9	**500**
Presence prob.	0.8	0.8	0.8	0.8	0.8	0.7	0.7	0.7	0.7	0.7	**0.6**
Top-10 prob.	0.8	0.8	0.8	0.8	0.766956	0.7	0.7	0.7	0.7	0.7	**0.6**
Rank	1	3	4	6	11	5	7	8	9	10	**2**

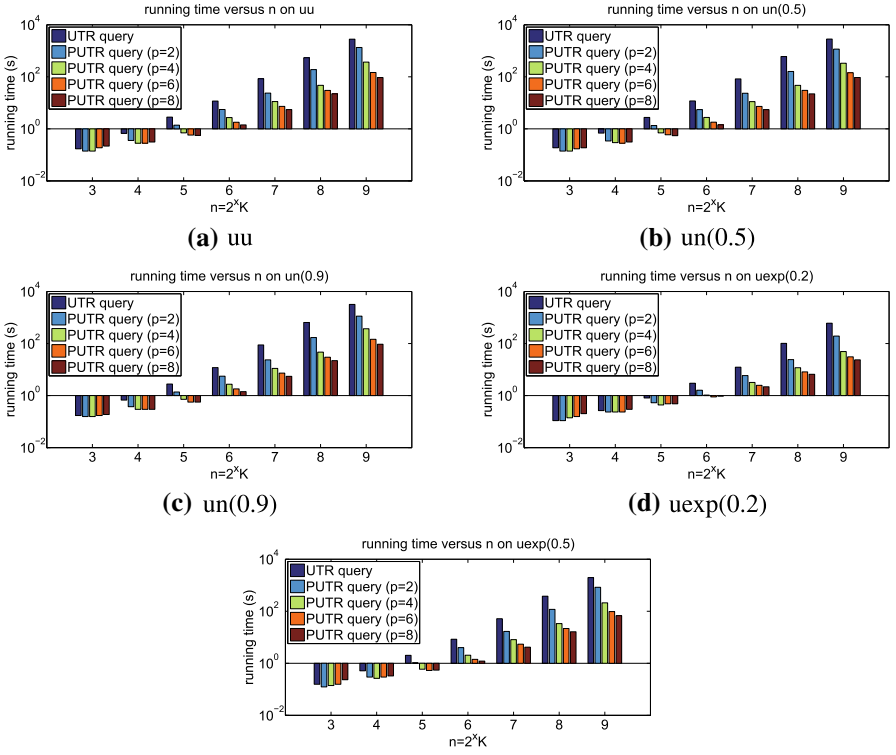
results. Although tuple t_{3601} (shown in bold) has the second largest score, it cannot be a member of the global answer set for the reason that its presence probability and the probability of being top-10 are all lower than that of tuples in the result set. Tuple t_{8408} (shown in bold) has the same top-10 probability with the tuples in the answer set, it cannot be a member of the global result set because its score are all smaller than that of tuples in the result set. From an experiment viewpoint, it verifies also the correctness of PUTR query, that is, it can correctly report global optimal l tuples users wanted.

In summary, our PUTR query captures those important tuples with better efficiency and effectiveness. In the following subsections, we will test the PUTR query on synthetic data sets for evaluating its scalability and performance further.

On the other hand, as a comparison task, we apply our UTR query and two other queries, i.e., PT-k query [15–17] and Global-Top$k$ query [29], on the uncertain dat aset. The ranking order is still the number of days of iceberg drift descending order. For UTR query, we set $k = l = 10$, $q = 0.3$ and $CR = [100,500]$, that is, the researchers would want to observe those records whose drifted days are located at between 100 and 500, and their probabilities of being top-10 are no less than 0.3. We set $k = 10$ and $q = 0.3$ for PT-k query and $k = 10$ for Global-Top$k$ query. The detailed information about the result sets of these three queries are showed in Tables 3 and 5, including attribute scores (drifted days), presence probabilities, top-10 probabilities and the corresponding ranks.

Table 6 The experimental data sets under various distributions

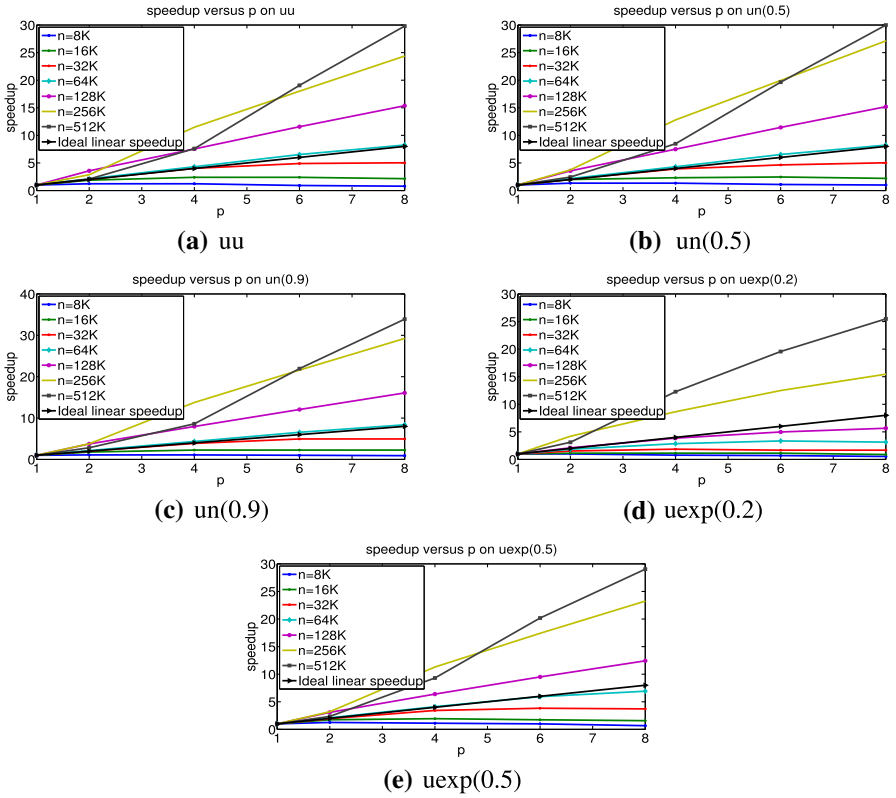
Datasets	Meanings
uu	The score and probability follow uniform distribution
un(0.5)	The score follows uniform distribution, and the probability follows normal distribution with mean value equal to 0.5, variance equal to 0.2
un(0.9)	The score follows uniform distribution, and the probability follows normal distribution with mean value equal to 0.9, variance equal to 0.2
uexp(0.2)	The score follows uniform distribution, and the probability follows exponent distribution with mean value equal to 0.2
uexp(0.5)	The score follows uniform distribution, and the probability follows exponent distribution with mean value equal to 0.5



(a) uu **(b)** un(0.5) **(c)** un(0.9) **(d)** uexp(0.2) **(e)** uexp(0.5)

Fig. 2 Running time versus n for various queries (different distributions)

All tuples with top-10 probability at least 0.3 and drifted days between 100 and 500 are reported by the UTR query. As shown in Table 3, the UTR query returns a set of 12 tuples $\{t_{6903}, t_{3610}, t_{3612}, t_{6928}, t_{4174}, t_{4020}, t_{8313}, t_{8412}, t_{8411}, t_{8409}, t_{8410},$



(a) uu **(b)** un(0.5)

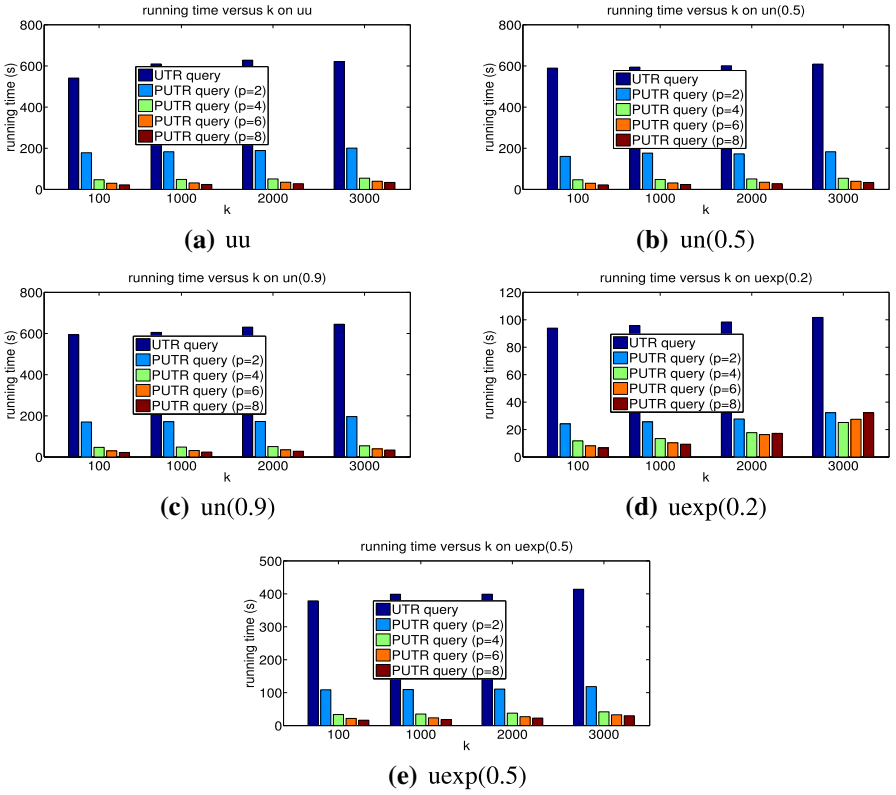
(c) un(0.9) **(d)** uexp(0.2)

(e) uexp(0.5)

Fig. 3 The speedup for different n (various probability distributions)

$t_{8408}\}$ as the top-(10,10) answer set, among which t_{6903} and t_{3610} (shown in bold) are obtained by probability lower bound pruning (Pruning Rule 4). Table 5 illustrates the results of PT-k query and Global-Top$k$ query. Although tuple t_{3601} (as shown in bold) has the relatively low presence probability, it is regarded as a member of the answer set for the reason that its score (drifted days) is larger or equal than that of tuples in the result set. However, the tuple t_{3601} is not included in the answer set of the PT-k query and Global-Top$k$ query despite the fact that t_{3601} has the largest score in the given score range. In other words, the two queries lose some relatively important tuples. From an real experiment viewpoint, it verifies also the effectiveness of UTR query, that is, the query can accurately report those optimal tuples users wanted.

The pruning ratio of the score range pruning and probability threshold pruning (i.e., Pruning Rules 1 and 2), the upper bound pruning of answer set (i.e., Pruning Technique 3) and probability lower bound pruning (i.e., Pruning Strategy 4) is 78.68, 99.46 and 13.33 %, respectively. The upper bound pruning of result set has a super pruning effect with the pruning ratio at least 99 %. The effects of Pruning Techniques 1 and 2 rests with the score range and probability threshold user selected. Although the Pruning Rule 4 has a relatively lower pruning effect, it contributes to our UTR



(a) uu **(b)** un(0.5) **(c)** un(0.9) **(d)** uexp(0.2) **(e)** uexp(0.5)

Fig. 4 Running time versus k for various queries (different distributions)

query capturing those important tuples missed by some uncertain top-k queries such as the PT-k query and Global-Top$k$ query.

6.2 Results on the synthetic data set

Since uncertain top-(k,l) query algorithms need to balance scores and probabilities of tuples, the experimental data should consider different distributions with respect to the scores and probabilities. In this section, we conduct the experiments on some synthetic data sets under different distributions. Their scores all follow the uniform distribution between zero and one hundred, and their probabilities obey uniform distribution, normal distribution, and exponent distribution, respectively. There is no correlation between the score and the probability. Table 6 illustrates the experimental data sets we adopted under various distributions.

6.2.1 Effects of the cardinality of database n

In this series of experiments, we vary the number of uncertain data n from $8K$ to $512K$ ($K = 1000$). Figure 2 (the y axis is log scale) shows the execution time of

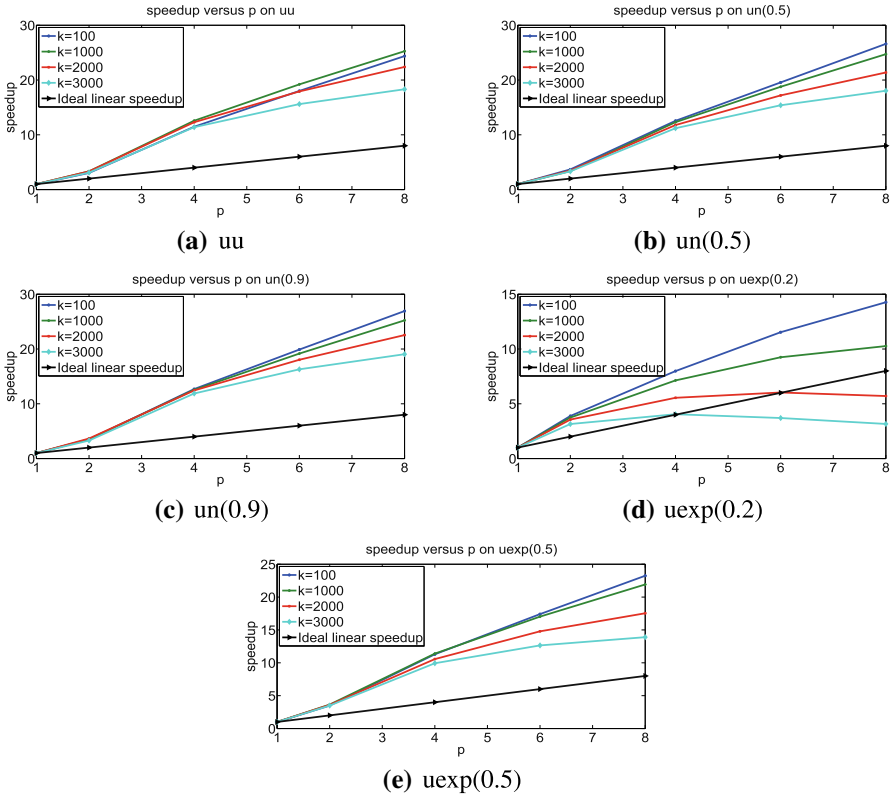
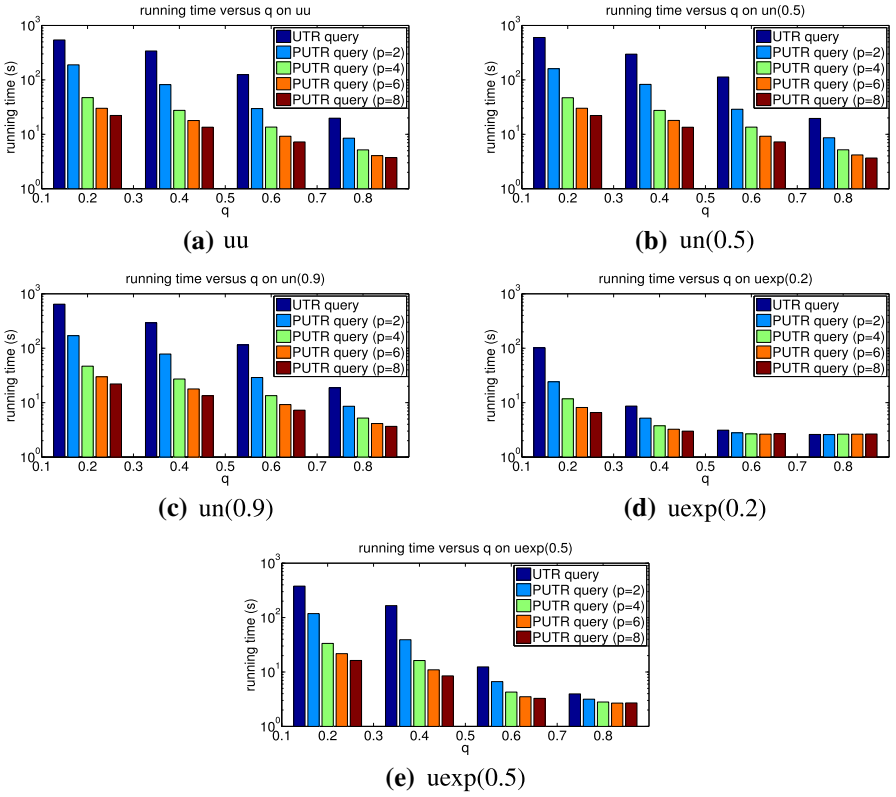


Fig. 5 The speedup for different k (various probability distributions)

various queries with the cardinality of data set n up to $512K$ under different data sets, where $CR = [10,90]$, $q = 0.2$ and $k = l = 100$. We can see from the pictures the time increases with n value, which is intuitive. Uu distribution pair and un distribution pair have relatively big running time while $uexp$ distribution pair (e.p., the $exp(0.2)$ distribution pair) has relatively small execution time for the UTR query. This can be explained based on the fact that a small quantity of tuples have relatively high probability under $uexp$ distribution, thereby the number of prunable tuples increase with n on the basis of the pruning strategies. However, for the un distribution pair where a considerable number of tuples are highly probable, the prunable tuples decrease based on the pruning techniques. On the other hand, there is a relatively longer execution time when a data set has a un distribution with larger mean value, because their probabilities, especially some relatively high probabilities, distribute so intensively that we can prune less tuples. As such, the execution time is shorter when a data set has a $uexp$ distribution with smaller mean value because the mean value forces probability to decay relatively fast leading to a small number of highly likely tuples. On the other side, as illustrated in the figure, the running time decreases as the number of threads and the size of data set increase obviously.



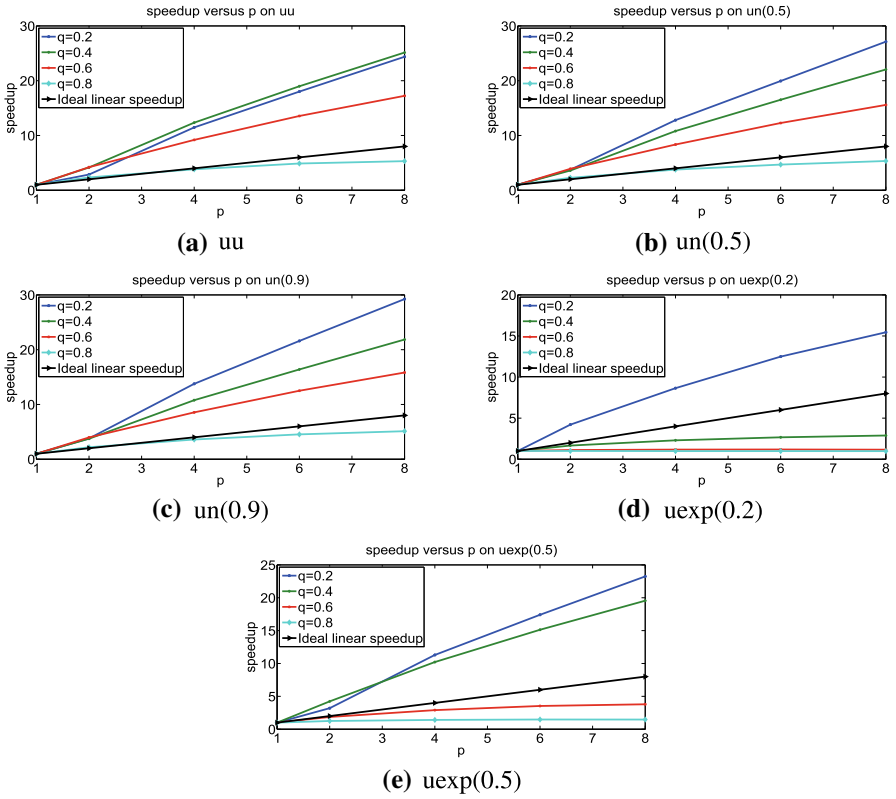
(a) uu **(b)** un(0.5) **(c)** un(0.9) **(d)** uexp(0.2) **(e)** uexp(0.5)

Fig. 6 Running time versus q for various queries (different distributions)

The change tendency of speedup of PUTR query for n under various threads and data sets is depicted in Fig. 3. Note that the black line denotes the ideal linear speedup, similarly hereinafter. From these graphs, we can know that the PUTR query can obtain super-linear speedup. One possible reason is that the data to be accessed in parallel reside in each core cache while the data visited serially cannot all stay in a single cache. So, compared to PUTR query, the UTR query takes more than p times time for the reason that the serial procedure has to access the slower part of storage system.

6.2.2 Effects of the parameter k

In this experiment, we study the influence of k value on the performance of UTR query and PUTR query under various types of data sets. We vary k value from $0.1K$ to $3K$. The experimental results are shown in Fig. 4. In this figure, we illustrates the query running time with k value up to $3K$, where $CR = [10,90]$, $q = 0.2$, $l = 100$ and $n = 512K$. From the figure, we can know that the execution time increases almost linearly as k value increases. The variation of running time of different distributions roughly has the similar form with Fig. 2 as the same causes aforementioned. Figure 5 illustrates the variation tendency of speedup for k under different threads and databases. On



(a) uu **(b)** un(0.5) **(c)** un(0.9) **(d)** uexp(0.2) **(e)** uexp(0.5)

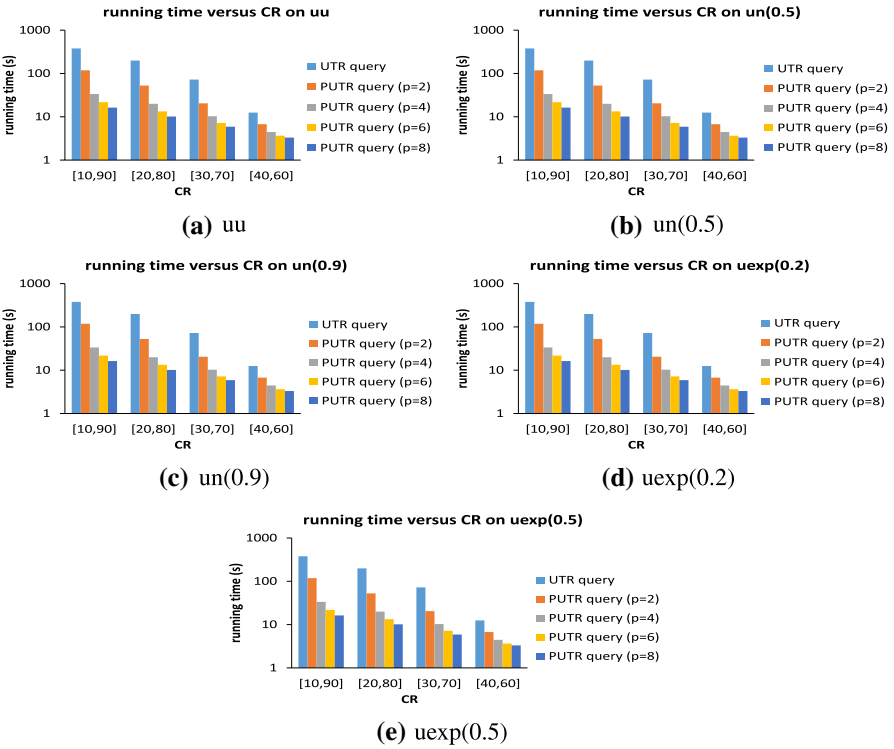
Fig. 7 The speedup for different q (various probability distributions)

the other hand, the speedup decreases as k value increases. We can see from these pictures, in general, the PUTR query can obtain super-liner speedup for the similar reasons aforementioned.

6.2.3 Effects of the probability threshold q

In this train of experiments, we vary probability threshold q value from 0.2 to 0.8. The experimental results are shown in Fig. 6 (the y axis is log scale). In these figures, we illustrate the query running time with q value up to 0.8, where $CR = [10,90]$, $k = l = 100$ and $n = 512K$. From the figures, we can know that the execution time decreases as q value increases, which is intuitive. On the other hand, as is shown in the figure, the running time declines dramatically with the number of threads and the probability threshold increase.

The change tendency of speedup of PUTR query for q under various threads and data sets is depicted in Fig. 7. From these graphs, we can know that the PUTR query can obtain super-linear speedup for smaller q value for the probable reason aforementioned. In addition, the speedup declines as the probability threshold q increases. On the other side, the speedup increases as p increases. However, the performance of



(a) uu **(b)** un(0.5) **(c)** un(0.9) **(d)** uexp(0.2) **(e)** uexp(0.5)

Fig. 8 Running time versus CR for various queries (different distributions)

PUTR query can decline as q and p increase. Because the number of prunable tuples is so big, the data set decreases sharply. As what mentioned before, it requires more overhead for OpenMP to start and distribute more threads. As a specific example, Fig. 7 shows the performance of PUTR query is worse than that of UTR query for $uexp(0.2)$ distribution when q is larger than 0.8.

6.2.4 Effects of the score range CR

Now, we vary score rang constraint CR for simulations. Figure 8 (the y axis is log scale) shows the experimental results. We describe the running time for different score range in the picture. Of course, in practice, the users may set the score range in terms of their own demands. On the other side, the variation of running time of various kinds of distribution data sets roughly has the approximate trend with the Figs. 4 and 6 as the similar causes mentioned previously. In addition, as is shown in the picture, the running time declines notably with p increases.

Figure 9 manifests the change trend of speedup for CR under different threads and data sets. From these figures, we can see that the PUTR query can achieve super-linear speedup for some score range for the possible reasons aforementioned. The variation tendency of speedup for CR under different threads and databases is similar to that of Fig. 7 for the same reasons mentioned before.

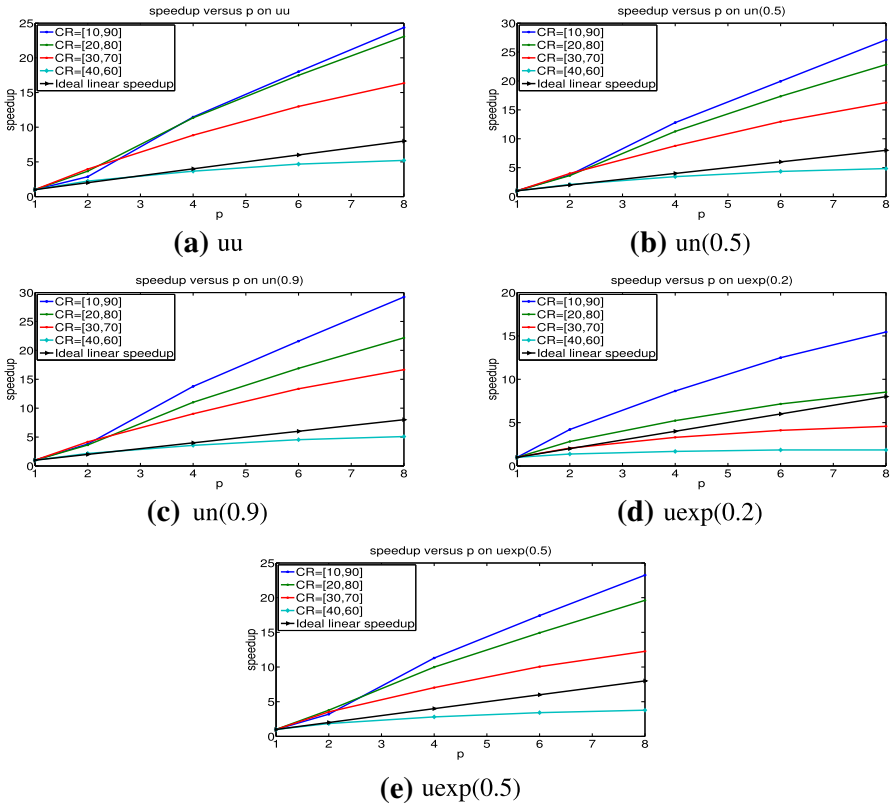
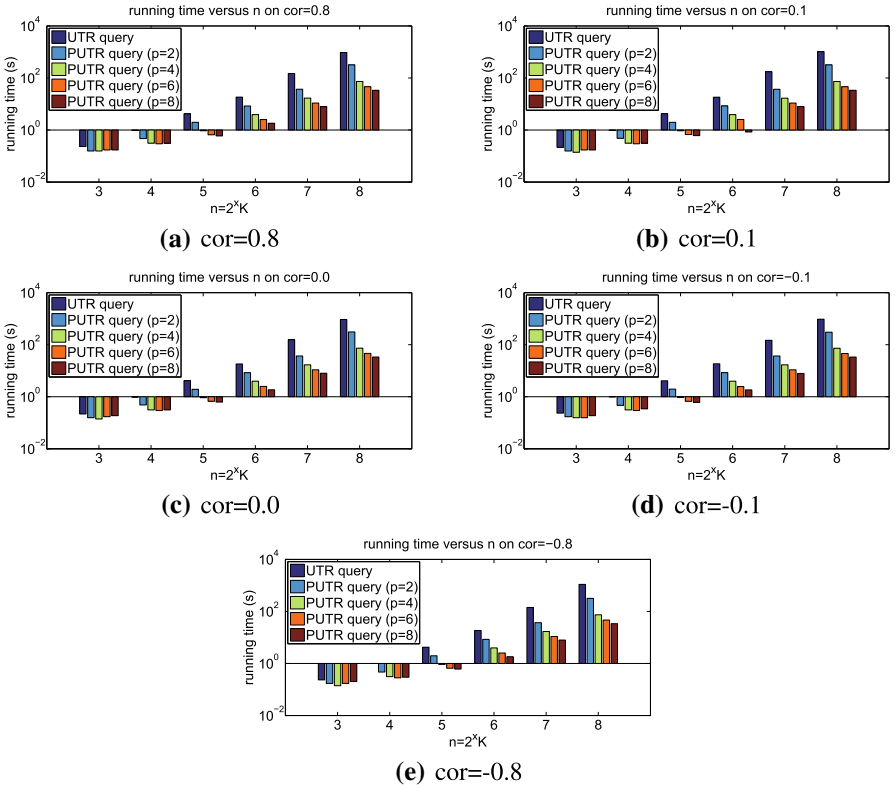


Fig. 9 The speedup for different CR (various probability distributions)

6.2.5 Effects of the score-probability correlations

In this series of experiments, we evaluate the effect of score-probability correlation on the performance of query. We use the synthetic uncertain databases of mutually independent tuples with the score and probability values of uncorrelation, positive correlation, and negative correlation. Given this, we generate bivariate normal data over score and probability, and control the correlation coefficient by adjusting bivariate covariance matrix. In this paper, we vary the correlation coefficient from -0.8 to 0.8 (mainly -0.8, -0.1, 0, 0.1, and 0.8). We evaluate the performance of UTR query and PUTR query under various parameter settings. Experimental results are shown in Figs. 10, 11, 12, and 13 (the y axis is log scale except Fig. 11).

At first, we study the influence of n on the query performance with the settings CR = [10,90], $q = 0.2$ and $k = l = 100$. In addition, we study the effect of k value on the performance of UTR query and PUTR query under various correlation coefficients. We vary k value from $0.1K$ to $2K$, where $CR = [10,90]$, $q = 0.2$, $l = 100$ and $n = 256K$. Another experiment illustrates the query execution time with q value up to 0.8, where



(a) cor=0.8 **(b)** cor=0.1

(c) cor=0.0 **(d)** cor=-0.1

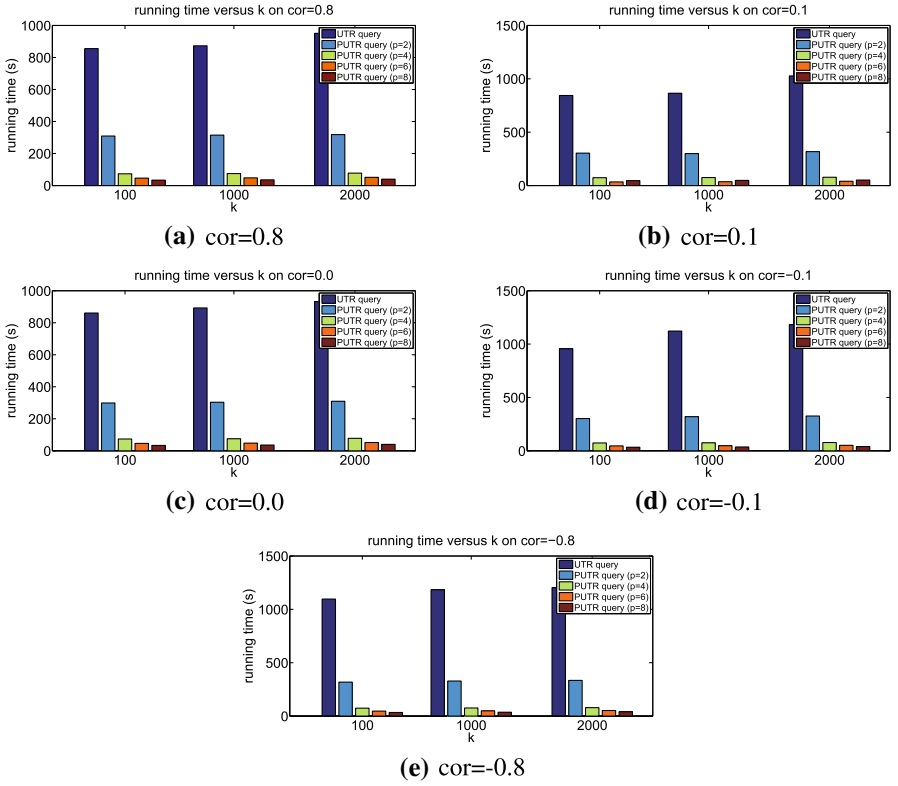
(e) cor=-0.8

Fig. 10 Running time versus n for various queries (different distributions)

$CR = [10,90]$, $k = l = 100$ and $n = 256K$. At last, we discuss the influence of different score range on performance of query under the setting of $q = 0.2$, $k = l = 100$ and $n = 256K$. We can know from these results that the positive correlation has positive effects on the performance of query while anti-correlation has negative influences on the performance. This can be explained based on the fact that for the data of positive correlation, high score tuples are attributed with high probability, which allows pruning considerably low probable tuples in advance to answer uncertain top-k range queries, while for negatively correlated data, more tuples need to be visited before concluding the answer set.

6.2.6 Pruning effects

In this subsection, we evaluate the efficiency and effectiveness of pruning techniques for different types of data used previously. Experimental results are shown in Tables 7 and 8. Table 7 illustrates the pruning ratios of pruning strategies under various kinds of probability distribution with the settings of $CR = [10,90]$, $q = 0.2$, $k = l = 100$ and $n = 512K$. Table 8 describes pruning ratios of these four pruning techniques under different bivariate gaussian data over score and probability with the settings



(a) cor=0.8 **(b)** cor=0.1

(c) cor=0.0 **(d)** cor=-0.1

(e) cor=-0.8

Fig. 11 Running time versus k for various queries (different distributions)

of $CR = [10,90]$, $q = 0.2$, $k = l = 100$ and $n = 256K$. From the tables, we can see that the Pruning Rule 3 (i.e., the upper bound pruning of answer set) has an exceptional pruning efficiency with the pruning rate at least 99.9 %. Pruning Rules 1 and 2 (i.e., the score range pruning and the probability threshold pruning), as the methods of predictive pruning, have also better pruning effects, which depends on the score range and probability threshold user selected. Although the Pruning Rules 4 and 5 (i.e., the probability lower pruning) have lower pruning effect, they contributes to our PUTR query capturing those important tuples with better efficiency and effectiveness.

In a word, extensive experiments based on synthetic data have very well verified the efficiency and effectiveness of our proposed algorithm for the uncertain parallel top-(k,l) range query, in terms of shorter execution time compared to UTR query, super-linear speedup, and excellent pruning ratio.

7 Conclusions

In real-world applications, uncertainty inherently exists in data. Therefore, it has recently become crucial to explore how to answer various queries over uncertain data

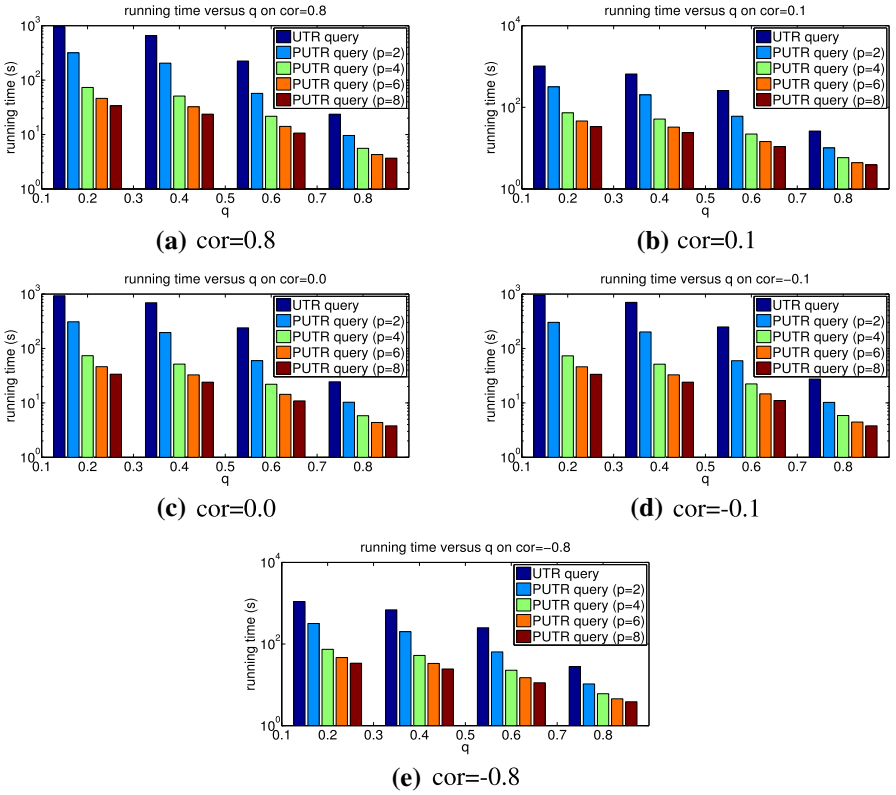
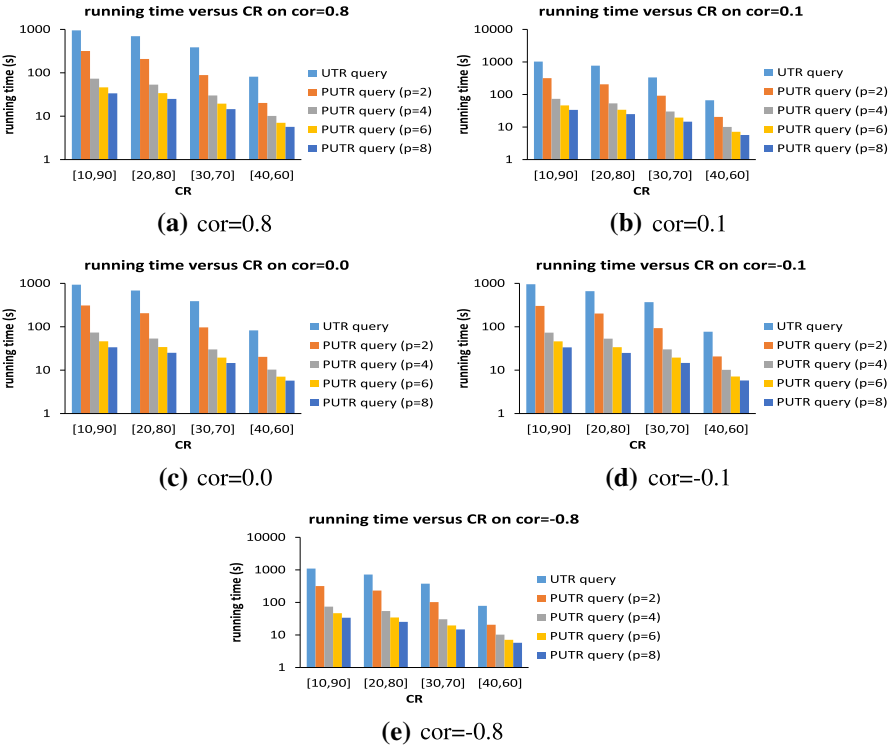


Fig. 12 Running time versus q for various queries (different distributions)

effectively and efficiently. Since traditional definite query processing methods usually assume precise data, they are not directly applicable to handling uncertain data. Thus, many query types in the uncertain database have to redefine query semantics in order to obtain accurate results from uncertain data, such as uncertain range query processing.

In this article, we present the definition of uncertain top-(k,l) range (UTR) query processing based on a given attribute score range and a probability threshold. In addition, we put forward some effective pruning rules to improve the performance of UTR query algorithm. Specially, based on the properties of top-k probability, we can find out an upper bound concerning the result set and a lower bound of top-k probability of tuples. What's more, with the rapid growth of Internet data, users need to obtain query results timely. Given this, a parallel implementation of UTR (PUTR) query on the grounds of the OpenMP on multicore architecture is developed for improving further the performance of UTR query. Extensive experiments have excellently verified the efficiency and effectiveness of our proposed approaches.

As for future work, we will extend mainly the query processing algorithm on the basis of the assumptions aforementioned, including tuple-level uncertainty with any



(a) cor=0.8 **(b)** cor=0.1 **(c)** cor=0.0 **(d)** cor=-0.1 **(e)** cor=-0.8

Fig. 13 Running time versus CR for various queries (different distributions)

Table 7 The pruning ratio under different probability distributions

Datasets/PR	$PR_{12}(\%)$	$PR_3(\%)$	PR_{45} (%)
uu	35.28	99.94	8.20
$un(0.5)$	35.38	99.95	8.09
$un(0.9)$	35.31	99.95	6.25
$uexp(0.2)$	45.80	99.93	5.56
$uexp(0.5)$	70.29	99.82	2.94

Table 8 The pruning ratio under various bivariate normal distributions

Datasets/PR	PR_{12} (%)	PR_3 (%)	PR_{45} (%)
Dataset with cor = 0.8	18.65	99.93	0.64
Dataset with cor = 0.1	18.47	99.92	1.21
Dataset with cor = 0.0	18.48	99.92	1.20
Dataset with cor = −0.1	18.54	99.92	0.61
Dataset with cor = −0.8	18.21	99.92	0.57

generation rules, attribute-level uncertainty. In addition, we will study some new extend queries by combining the uncertain top-k query with some other classical queries, e.g., group query, nearest neighbour query, skyline query, and subspace query etc.

Furthermore, we will study their performance of parallel optimization on the basis of some parallel programming platforms, e.g., MPI, GPU, MIC, and MapReduce etc.

Acknowledgments The authors would like to thank the three anonymous reviewers for their valuable and helpful comments on improving the manuscript. This research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos.61133005, 61432005), and the National Natural Science Foundation of China (Grant Nos.61370095, 61202109, 1472124), Project supported by the National Science Foundation for Distinguished Young Scholars of Hunan (12JJ1011).

References

1. Abiteboul, S., Kanellakis, P., Grahne, G.: On the representation and querying of sets of possible worlds. In: SIGMOD (1987)
2. Afshani, P., Brodal, G.S., Zeh, N.: Ordered and unordered top-k range reporting in large data sets. In: Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 390–400 (2011)
3. Aggarwal, C.C., Yu, P.S.: A survey of uncertain data algorithms and applications. IEEE Trans. Knowl. Data Eng. **21**, 609–623 (2009)
4. Chen, J., Cheng, R.: Efficient evaluation of imprecise location-dependent queries. In: Proceedings of the 23th International Conference on Data Engineering (2007)
5. Cheng, R., Kalashn, I.D., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: Proceeding of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 551–562. ACM Press, New York (2003)
6. Cheng, R., Xia, Y., Prabhakar, S., et al.: Efficient indexing methods for probabilistic threshold queries over uncertain data. In: Proceedings of the VLDB (2004)
7. Cormode, G., Li, F., Yi, K.: Semantics of ranking queries for probabilistic data and expected ranks. In: Proceedings of the International Conference on Data Engineering, pp. 305–316. IEEE Computer Society Press, Washington (2009)
8. Dagum, L., Menon, R.: OpenMP: an industry standard API for shared-memory programming. IEEE Comput. Sci. Eng. **5**(1), 46–55 (1998)
9. Dai, X.Y., Yiu, M.L., Mamoulis, N., Tao, Y.F., Vaitis, M.: Probabilistic spatial queries on existentially uncertain data. In: SSTD, pp. 400–417 (2005)
10. Ding, X.F., Jin, H.: Efficient and progressive algorithms for distributed skyline queries over uncertain data. IEEE Trans. Knowl. Data Eng. **24**(8), 1148–1162 (2012)
11. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top-k lists. SIAM J. Discret. Math. **17**(1), 134–160 (2004)
12. Ge, T., Zdonik, S., Madden, S.: Top-k queries on uncertain data: on score distribution and typical answers. In: Proceedings of the SIGMOD, pp. 375–388. ACM Press, New York (2009)
13. Gedik, B., Wu, K.L., Yu, P.S., Liu, L.: Processing moving queries over moving objects using motion-adaptive indexes. IEEE Trans. Knowl. Data Eng. **18**(5), 651–668 (2006)
14. Hu, H.B., Lee, D.L.: Range nearest-neighbor query. IEEE Trans. Knowl. Data Eng. **18**(1), 78–91 (2006)
15. Hua, M., Pei, J.: Ranking queries on uncertain data. VLDB J. **20**(1), 129–153 (2011)
16. Hua, M., Pei, J., Zhang, W.J., Lin, X.M.: Ranking queries on uncertain data: a probabilistic threshold approach. In: Proceedings of the SIGMOD, pp. 673–686. ACM Press, New York (2008)
17. Hua, M., Pei, J., Zhang, W.J., Lin, X.M.: Efficiently answering probabilistic threshold top-k queries on uncertain data. In: Proceedings of the International Conference on Data Engineering, pp. 1403–1405. IEEE Computer Society Press, Washington (2008)
18. Jestes, J., Cormode, G., Li, F.F., Yi, K.: Semantics of ranking queries for probabilistic data. IEEE Trans. Knowl. Data Eng. **23**(12), 1903–1917 (2010)
19. Li, J., Saha, B., Deshpande, A.: A unified approach to ranking in probabilistic databases. Proc. VLDB Endow. **2**(1), 502–513 (2009)
20. Lian, X., Chen, L.: Probabilistic ranked queries in uncertain databases. In: Proceedinggs of the EDBT, pp. 511–522. ACM Press, New York (2008)

21. Lian, X., Chen, L.: Shooting Top-k stars in uncertain databases. J. VLDB **20**(6), 819–840 (2011)
22. Lin, X., Xu, J.L., Hu, H.B.: Range-based skyline queries in mobile environments. IEEE Trans Knowl. Data Eng. **25**(4), 835–849 (2013)
23. Sarma, A.D., Benjelloun, O., Halevy, A., Widom, J.: Working models for uncertain data.In: ICDE (2006)
24. Sheng, C., Tao, Y.F.: Dynamic top-k range reporting in external memory. In: Proceedings of ACM Symposium on Principles of Database Systems (PODS), pp. 121–130 (2012)
25. Soliman, M.A., Ilyas, I.F., Chang, K.C.C.: Top-k query processing in uncertain databases. In: Proceedings of the International Conference on Data Engineering, pp. 896–905. IEEE Computer Society Press, Washington (2007)
26. Soliman, M.A., Ilyas, I.F.: Ranking with uncertain scores. In: Proceedings of the ICDE, pp. 317–328. IEEE Computer Society Press, Washington (2009)
27. Tao, Y.F., Cheng, R., Xiao, X.K.: Indexing multi-dimensional uncertain data with arbitrary probability density functions. In: Proceedings of the VLDB (2005)
28. Yiu, M.L., Mamoulis, N., Dai, X.Y., Tao, Y.F., Vaitis, M.: Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data. IEEE Trans. Knowl. Data Eng. **21**(1), 108–122 (2009)
29. Zhang, X., Chomicki, J.: On the semantics and evaluation of Top-K queries in probabilistic databases. Distrib. Parallel Databases **26**(1), 67–126 (2009)
30. Zhang, Z., Yang, Y., Tung, A.K.H., Papadias, D.: Continuous k-means monitoring over moving objects. IEEE Trans. Knowl. Data Eng. **20**(9), 1205–1216 (2008)