Scalable Probabilistic Similarity Ranking in Uncertain Databases

Thomas Bernecker, Hans-Peter Kriegel, Nikos Mamoulis, Matthias Renz, and Andreas Zuefle

Abstract—This paper introduces a scalable approach for probabilistic top-k similarity ranking on uncertain vector data. Each uncertain object is represented by a set of vector instances that is assumed to be mutually exclusive. The objective is to rank the uncertain data according to their distance to a reference object. We propose a framework that incrementally computes for each object instance and ranking position, the probability of the object falling at that ranking position. The resulting rank probability distribution can serve as input for several state-of-the-art probabilistic ranking models. Existing approaches compute this probability distribution by applying the *Poisson binomial recurrence* technique of quadratic complexity. In this paper, we theoretically as well as experimentally show that our framework reduces this to a linear-time complexity while having the same memory requirements, facilitated by incremental accessing of the uncertain vector instances in increasing order of their distance to the reference object. Furthermore, we show how the output of our method can be used to apply probabilistic top-k ranking for the objects, according to different state-of-the-art definitions. We conduct an experimental evaluation on synthetic and real data, which demonstrates the efficiency of our approach.

Index Terms—Uncertain databases, probabilistic ranking, similarity search.

1 Introduction

In the past two decades, there has been a great deal of interest in developing efficient and effective methods for similarity search and mining in spatial, temporal, multimedia, and sensor databases. At the same time, improvements in our ability to capture and store data have led to massive data sets with complex structured data, which require special methodologies for efficient and effective data exploration tasks. In this work, we introduce a scalable approach for probabilistic similarity ranking on uncertain vector data.

Similarity ranking is a hot topic in database research because it plays a major role in a large number of emerging applications, such as data retrieval, decision support systems, and data mining that require exploratory querying on the aforementioned databases. For example, clustering and ranking have a mutual reinforcement property for search engines. While search engines use clustering to identify groups of relevant objects, ranking is used to report the most important first. A ranking query orders the objects in a database with respect to their similarity to a reference object. In a spatial database context, nearest neighbor queries rank the contents of a spatial object set (e.g., restaurants) in increasing order of their distance to a reference location. In a database of images, a similarity query ranks the feature vectors of images in increasing order of their distance (i.e., dissimilarity) to a query image. Such types of similarity queries are, in particular, important for many data mining applications including classification, clustering, and outlier detection. One direct use of such queries in data mining is in classification tasks, where k-NN queries are often used for classifying data items of unknown labels to class labels corresponding to the most similar labeled item. Clustering is also a relevant application, where the nearest neighbor search is used for assignment to clusters, e.g., k-medoids. In addition, a number of outlier detection methods are based on similarity queries, e.g., the detection of k-NN outliers that are defined as objects having the highest k-NN distances.

More recently, it has been recognized that many applications dealing with spatial, temporal, multimedia, and sensor data have to cope with uncertain or imprecise data. Uncertainty in the data can be caused due to a number of reasons. First, recording data involves uncertainty by nature either caused by imprecise sensors or by imprecision induced by the discretization, which is necessary to record the data. For instance, positions of moving individuals concurrently tracked by multiple sensor devices are usually inconsistent. This problem is also inherent in sensor networks collecting data such as temperature, humidity, etc. Often, objects in relational databases are redundantly represented by multiple tuples due to inconsistent data observations or to ensure privacy protection. One approach to achieve more reliable information from the data recording process is to record the data based on multiple observations, e.g., observations derived from multiple (preferably independent) sensors. Consequently, the observed object or state of a process is recorded as a set of possible instances, e.g., a set of images or a set of alternative positions. Second, uncertainty obviously occurs in prediction tasks, e.g., weather forecasting, stock market prediction, and traffic jam prediction. Here again, the consideration of a number of possible instances, i.e., alternative prediction results may

- *T. Bernecker, H.-P. Kriegel, M. Renz, and A. Zuefle are with the Institut für Informatik, Ludwig-Maximilians-Universität München, Oettingenstr. 67, 80538 München, Germany. E-mail: {bernecker, kriegel, renz, zuefle}@dbs.ifi.lmu.de.*
- *N. Mamoulis is with the Department of Computer Science, University of Hong Kong, Rm. 301 Chow Yei Ching Building, Pokfulam Road, Hong Kong. E-mail: nikos@cs.hku.hk.*

Manuscript received 8 Apr. 2009; revised 9 Aug. 2009; accepted 28 Sept. 2009; published online 30 Apr. 2010.
Recommended for acceptance by R. Cheng, M. Chau, M. Garofalakis, and J.X. Yu.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-2009-04-0337.
Digital Object Identifier no. 10.1109/TKDE.2010.78.

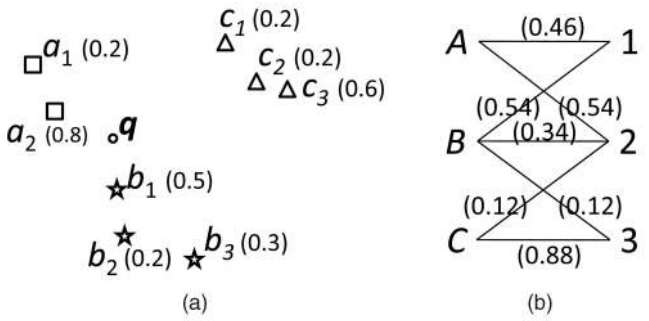

Fig. 1. Object instances and rank probability graph. (a) Object instances. (b) Bipartite graph.

help to improve the reliability of implications based on the predictions. For example, the traffic density on a single road segment can be well predicted for a given time in the future if we predict and incorporate all possible locations of all individuals at that time as proposed in [16]. The third motivation for uncertain data is privacy preserving issues. In contrast to the above reasons, privacy preserving applications often require uncertainty in the data in order to shield the exact information of objects or individuals. For example, often some digits of credit card numbers on receipts are hidden in order to avoid that the complete number is visible to a third party. As a consequence, there is a need to adapt storage models and indexing/search techniques to deal with uncertainty. There is already a volume of research on probabilistic data models [3], [20], [21], [2].

In this paper, we focus on similarity ranking of uncertain vector data. Prior work in this direction includes [7], [9], [24], [6], [14], [15], [10], [22]. In a nutshell, there are two models for capturing uncertainty of objects in a high-dimensional space. In the *continuous* uncertainty model, the uncertain values of an object are represented by a continuous probability distribution function (pdf) within the vector space. This type of representation is often used in applications where the uncertain values are assumed to follow a specific pdf, e.g., a Gaussian distribution [6]. Similarity search methods based on this model involve expensive integrations of the pdfs, thus special approximation techniques for efficient query processing are typically employed [24]. In the *discrete* uncertainty model, each object is represented by a discrete set of alternative values, and each value is associated with a probability [14]. The main motivation of this representation is that, in most real applications, data are collected in a discrete form (e.g., information derived from sensor devices). In this paper, we adopt the discrete uncertainty model, which also complies with the x-relations model used in the *Trio* system [1].

Consider, for example, a set of 3 two-dimensional objects A, B, and C (e.g., locations of mobile users) and their corresponding uncertain instances $\{a_1, a_2\}$, $\{b_1, b_2, b_3\}$, and $\{c_1, c_2, c_3\}$, as shown in Fig. 1a. Each instance carries a probability (shown in brackets) and instances of the same object are mutually exclusive. In addition, the sum of the probabilities of each object's instances cannot exceed 1. Assume that we wish to rank the objects A, B, and C according to their distances to the query point q shown in the figure. Clearly, several rankings are possible. In specific, each combination of object instances defines an order. For example, for combination $\{a_1, b_1, c_1\}$, the object ranking is (B, A, C), while for combination $\{a_2, b_3, c_1\}$, the object ranking is (A, B, C). Each combination corresponds to a *possible world* [1], whose probability can be computed by multiplying the probabilities of the instances that comprise it, assuming independent existence probabilities between the instances of different objects.

The example illustrates the ambiguity of ranking in uncertain data. On the other hand, most applications require the definition of a nonambiguous object ranking. For example, assume that a robbery took place at location q and the objects correspond to the positions of suspects that are sampled around the time that the robbery took place. The probabilities of the samples depend on various factors (e.g., time difference of the sample to the robbery event, errors of capturing devices, etc.). As an application, we may want to define a definite probabilistic proximity ordering of the suspects to the event, in order to prioritize interrogations.

Various top-k query approaches have been proposed generating unambiguous rankings from probabilistic data. Examples are U-top$k$ [23], U-kRanks [23], PT-k [13], Global top-k [29], and expected rank [10]. A summary of these ranking models can be found in [10]. All of them attempt to weigh the objects based on their probability to be in each of the first k ranks, but they use different ways to define the weights.

A common module in most of these approaches is the computation for each object instance x the probability P_i that i objects are closer to q than x for all $1 \leq i \leq k$. The resulting probabilities are aggregated to build the probability of each object at each rank. For example, the U-kRanks query reports the ith result as the object that is the most likely to be ranked ith over all possible worlds. For this computation, we obviously need the probabilities of all instances to be ranked ith over all possible worlds. The probability that an object is ranked at a specific position i can be computed by summing the probabilities of the possible worlds that support this occurrence. In our example, the probability that object A occurs as first one is 0.46 and the probability that object B is the first is 0.54. All possible occurrences and the corresponding probabilities are represented by the object-rank bipartite graph, which is shown in Fig. 1b. Nonexisting edges imply zero probability, i.e., it is not possible that the object occurs at the corresponding ranking position. In this example, all instances of A precede all those of C, so C cannot occur as first object and A cannot be ranked to the last position.

In this paper, we propose a framework that, given a database with uncertain vector objects, computes the rank probabilities of the object instances (e.g., a_1) in linear time to the total number of instances of all objects. Here, we assume that the instances are accessed in increasing distance order to the query object q (e.g., with the help of a nearest neighbor search algorithm [12]). As these can be aggregated on the fly, our framework also computes the rank probabilities of the objects (e.g., A) at the same cost. This is a great improvement, over the state of the art [27], which computes these probabilities in quadratic time.

Analogously to the *Trio* [1] system, we define an uncertain database as a set of uncertain objects (x-tuples), each including a number of alternatives associated with probabilities. Here, we consider uncertain vector objects in a d-dimensional vector space, i.e., each object is assigned to

multiple alternative positions associated with a probability value. Let us note that this model assumes independence among the uncertain objects.

Definition 1 (Uncertain Vector Objects). *An uncertain vector object X corresponds to a finite set of points in a d-dimensional vector space, called object instances, each associated with a probability value, i.e., $X = \{(x, P(X = x)),$ where $x \in \mathbb{R}^d$, and $P(X = x) \in [0, 1]\}$ is the probability that X has position x. The probabilities of the object instances represent a discrete probability distribution of the alternative points such that the condition $\sum_{(X,P(X=x))\in X} P(X = x) \leq 1$ holds. The collection of instances of all objects forms the uncertain database $\mathcal{D}$.*[1]

Since the number of possible worlds is exponential in the number of uncertain objects, it is impractical to enumerate all of them in order to find the rank probabilities of all object instances. Recently, it has been shown in [26] that we can compute the probabilities between all object instances and ranks in $O(kn^2)$ time, where n is the number of object instances required to be accessed until the solution is confirmed. This solution can be applied to all problems that comply to the x-relation model (including our problem). In this paper, we propose a significant improvement of this approach, which reduces the time complexity to $O(kn)$.

In Section 5, we discuss in detail how our method can be used as a module in various models that rank the objects according to the rank probabilities of their instances.

Although in the paper we focus on databases of uncertain vector objects as in Definition 1, our results apply, in general, to x-relations as defined in [1], which model mutual exclusiveness constraints between existentially uncertain tuples (i.e., object instances in our model).[2] Thus, our method is general and it can be used irrespective of whether we have uncertain objects or existentially uncertain tuples with exclusiveness constraints, expressed by x-tuples.

1.1 Contributions and Outline

The main contributions of this paper can be summarized as follows:

- We propose a framework based on iterative distance browsing that efficiently supports probabilistic similarity ranking in uncertain vector databases.
- We present a novel and theoretically founded approach for computing the rank probabilities of each object. We prove that our method reduces the computational cost of the rank probabilities from O(kn^2), achieved by the best currently known method, to O(kn).
- We show how diverse state-of-the-art probabilistic ranking models can use our framework to accelerate computation.
- We conduct an experimental evaluation using real and synthetic data, which demonstrates the applicability of our framework and verifies our theoretical findings.

1. Note that the condition $\sum_{(x,P(X=x))\in X} P(X = x) < 1$ implies existential uncertainty, meaning that the object may not exist at all.

2. The general model based on uncertain tuples uses a score function instead of a distance function in order to define an order of the tuples.

TABLE 1
Notations Used in This Work

Table of Notations	
$\mathcal{D}$	an uncertain database
N	the cardinality of $\mathcal{D}$
q	a query vector in respect to which a probabilistic ranking is computed
k	the ranking depth that determines the number of ranking positions of the ranking query result
D	a distance browsing of $\mathcal{D}$ with respect to q
X,Y,Z	uncertain vector objects, each corresponding to a finite set of alternative vector point instances
x,y,z	vector point instances belonging to objects X,Y,Z respectively.
$P(X = x)$	the probability that an uncertain vector object X matches a given vector point instance x.
$P_i(X)$	the probability that object X is assigned to the i-th ranking position i, i.e. the probability that exactly (i-1) objects in $(\mathcal{D} \setminus \{X\})$ are closer to q than X
$P_i(x)$	the probability that an instance x of object X is assigned to the i-th ranking position i, i.e. the probability that exactly $i-1$ objects in $(\mathcal{D} \setminus \{X\})$ are closer to q than x
AOL	*Active Object List*
$\mathcal{S}$	a set of objects that have already been seen, i.e. the set that contains an object X *iff* at least one instance of X has already been returned by the distance browsing D
$P_{i,S,x}$	the probability that exactly i objects $X \in S$ are closer to q than an object instance x
$P_x(Z)$	the probability that object Z is closer to query point q than the vector point x; computable using Lemma 1

The rest of the paper is organized as follows: In Section 2, we survey existing work in the field of managing and querying uncertain data. In Section 3, we introduce our framework for computing the rank probabilities of uncertain object instances, followed by the details regarding the efficient incremental rank probability computation for each object instance. The complete algorithm for computing the rank probabilities for all instances and the corresponding objects is presented in Section 4. We experimentally evaluate the efficiency of our approach in Section 6 and conclude the paper in Section 7. All notations used throughout this paper are listed in Table 1.

2 Related Work

The potential of uncertain data processing has achieved increasing interest in diverse application fields, e.g., sensor monitoring [8], traffic analysis, location-based services [25], etc. Till date, uncertain data management has been established as an important branch of research within the database community, with increasing tendency. Existing approaches in this field of modeling of, managing of, and query processing on uncertain data can be categorized into diverse directions, including probabilistic databases [3], [20], [21], [2], indexing of uncertain data [9], [24], [6], [28], and probabilistic query processing [7], [11], [6], [14], [5], [27], [23].

In [6], the Gauss-tree is introduced, which is an index for managing large amount of uncertain objects with their uncertain attribute represented by a Gaussian distribution function. Objects which have the highest probability of being located inside a given query range are reported efficiently. Note that this definition is semantically different from the problem studied in this paper. In contrast, the approaches for managing uncertain vector objects proposed in [7], [9], [24] support arbitrarily shaped probability distribution functions for uncertain object attributes. Since the above-mentioned approaches focus on probability computations based on query predicates according to a given query range, they are not applicable to our problem. Although Yiu et al. [28] study probabilistic ranking of objects according to their distance from a reference query point, the solutions are limited to existentially uncertain spatial data with a single alternative.

To the best of our knowledge, only Bernecker et al. [5] address the probabilistic ranking according to our problem definition. There, a divide-and-conquer method for accelerating the computation of the ranking probabilities is proposed. Although the proposed approach achieves a significant speedup compared to the naive solution incorporating each possible database instance, its runtime is still exponential. Related to our ranking problem, significant work has been done in the field of probabilistic top-k query processing. Soliman et al. [23] were the first who studied such problems on the x-relations model of [3]. They proposed two ways of ranking uncertain tuples. In the first, *uncertain top-k* (U-Top$k$) query, the objective is to find the k-permutation of the most likely tuples to be the top-k. In our setting, this corresponds to finding the top-k most probable object instances (belonging to different objects) in all possible worlds. The *uncertain k-ranks query* (U-kRanks) reports a probabilistic ranking of the tuples (again, *not* the x-tuples). However, an efficient approach for this problem is only given for the case where the tuples are mutually independent, which does not hold for the x-relation model. At the same time, Re et al. [20] proposed an efficient but approximative probabilistic ranking based on the concept of Monte-Carlo simulation. Later, Yi et al. [27] proposed the first efficient exact probabilistic ranking approach for the x-relation model, for both cases of single-alternative x-tuples only, i.e., x-tuples with only one uncertain instance, and multialternative x-tuples. They proposed dynamic programming-based methods for the computation of uncertain ranking queries, which have much lower costs than the previously best known results. Furthermore, they proposed early stopping conditions for accessing the tuples. Their methods for U-Top$k$ and U-kRanks queries have O($nlogk$) and O(kn^2) time complexities, respectively. The cost of the U-kRanks algorithm is dominated by the computation of the probability of each accessed tuple to be in each of the k first ranks. In this paper, we also use this as a module of finding the object-rank probabilities. However, we propose an improvement of their O(kn^2) algorithm that does the same work in O(kn) without increasing the memory requirements.

In a recent paper, Cormode et al. [10] reviewed alternative top-k ranking approaches for uncertain data, including the U-Top$k$ and U-kRanks queries, and argued for a more robust definition of ranking, namely the *expected* rank for each tuple (or x-tuple). This is defined by the weighted sum of the ranks of the tuple in all possible worlds, where each world in the sum is weighed by its probability. The k tuples with the lowest expected ranks are argued to be a more appropriate definition of a top-k query than previous approaches. Nevertheless, we found by experimentation that such a definition may not be appropriate for ranking objects (i.e., x-tuples), whose instances have large variance (i.e., they are scattered far from each other in space). In general, the result of this ranking method is similar to the brute-force approach that would take the mean of the instances for each object and rank these means. On the other hand, approaches that take into consideration the rank probabilities (e.g., U-kRanks) would be more suitable for such data. This is the reason why we focus on the computation of rank probabilities in this paper. Another piece of recent related work is [22], where the goal is to rank uncertain objects (i.e., x-tuples) whose score is uncertain and can be described by a range of values. Based on these ranges, the authors define a graph that captures the partial orders among objects. This graph is then processed to compute U-kRanks and other queries. Although this work has similar objectives to ours, it operates on a different input, where the distribution of uncertain scores is already known, as opposed to our work which dynamically computes this distribution by performing a linear scan over the ordered object instances.

3 Probabilistic Ranking Framework

Our framework basically consists of two modules, which are performed in an iterative way:

- The first module (*distance browsing*) incrementally retrieves the instances of all objects in order of their distance to q. This can be achieved with the help of a multidimensional index (e.g., an R^*-tree index [17]), using an incremental nearest neighbor search algorithm [12].
- The second module computes the probabilistic ranking $P_i(x)$ of each object instance x reported from the distance browsing for all $1 \leq i \leq k$. This step is the main focus of this paper because of its potentially high computational cost. A naive solution would take into account all possible worlds that include the instance and update the probabilities accordingly; however, as discussed before, there already exists an efficient solution that can perform this computation in quadratic time and linear space [26]. In this paper, we improve this method to a linear time and space complexity algorithm. The key idea is to use the probabilistic ranks of the previous object instance to derive those of the currently accessed one in O(k) time. Section 3.2 describes the details of this improvement.

Our framework is illustrated in Fig. 2. The computation of the probability distributions is iteratively processed within a loop. First, we initialize a distance browsing among the object instances starting from a given query point q. Other orders used for the instance browsing, e.g., descending probability as discussed in [27], might possibly lead to faster algorithms if the probability distribution favors them.

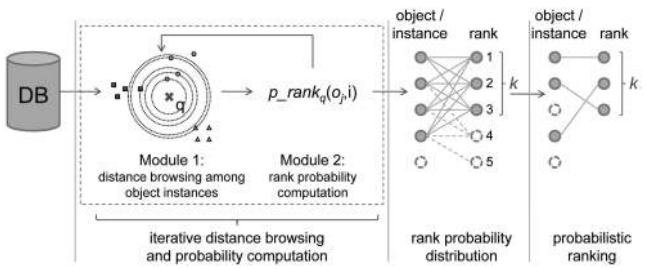

Fig. 2. Framework for probabilistic similarity ranking.

However, the distance-based order is somewhat natural for NN search around a query point, as there exist efficient search modules that support it. Furthermore, the distance-based sorting supports spatial pruning techniques in order to reduce the candidate set as far as possible due to the restricted memory. For each object instance fetched from the distance browsing (Module 1), we compute the corresponding rank probabilities (Module 2) and update the rank probability distributions generated from the probabilistic ranking routine.

Note that the rank probabilities of the object instances (i.e., tuples in the x-relations model) reported from the second module can be optionally aggregated into the rank probabilities of the objects (i.e., x-tuples in the x-relations model). The probability that an uncertain vector object $X = \{(x_1, P(X = x_1)), \ldots, (x_s, P(X = x_s))\}$ is at the ith ranking position according to the distance to a reference query object q (or generally according to a score function s(x)) is

$$P_i(X) = \sum_{x \in X} P(X = x) \cdot P_i(x).$$

Our framework can be used to compute the object-based rank probabilities by maintaining a list of objects from which instances have been seen so far and successively aggregate the rank probabilities by means of the instance-based rank probabilities reported from the framework.

Finally, in a postprocessing step, the rank probability distributions computed by our framework can be used to generate a definite ranking of the objects or object instances. The objective is to find a nonambiguous ranking, where each object or object instance is uniquely assigned to one rank. Here, one can plug-in any user-defined ranking method that requires rank probability distributions of objects in order to compute unique positions. In Section 5, we illustrate this for several well-known probabilistic ranking queries that make use of such distributions. In particular, we demonstrate that by using our framework, we can process such queries in $O(nlogn + k \cdot n)$ time,[3] as opposed to the existing approaches that require $O(k \cdot n^2)$ time.

3.1 Dynamic Probability Computation

Consider an uncertain object X, defined by m probabilistic instances $X = \{(x_1, P(X = x_1)), \ldots, (x_m, P(X = x_m))\}$. The probability that X is assigned to a given ranking position i is equal to the chance that exactly $i - 1$ objects $Z \in (\mathcal{D} \setminus X)$ are closer to the query object q than the object X. This can be computed by aggregating the probabilities over all instances $(x, P(X = x))$ of X that exactly $i - 1$ objects Z are closer to q than the instance $(x, P(X = x))$. Formally,

$$P_i(X) = \sum_{(x, P(X=x)) \in X} (P_i(x) \cdot P(X = x)). \quad (1)$$

Based on the above formula, we can compute the probabilities for an object X to be assigned to each of the ranking positions $i \in \{1, \ldots, k\}$ by computing the probabilities $P_i(x)$ for all instances $(x, P(X = x))$ of X. As mentioned above, we perform this computation in an iterative way, i.e., whenever we fetch a new object instance $(x, P(X = x))$, we compute all probabilities $P_i(x) \cdot P(X = x)$ for all $i \in \{1, \ldots, k\}$. Thereby, in a list, we store the current *probability state* according to all ranking positions $i \in \{1, \ldots, k\}$ for each object for which we already have accessed some instances and for which we expect to obtain further instances in the remaining iterations. Whenever the probabilities according to a new object instance are computed, we update the list by adding the new probabilities to the current probability state.

In the following, we show how to compute the probabilities $P_i(x) \cdot P(X = x)$ for all $i \in \{1, \ldots, k\}$ for a given object instance $(x, P(X = x))$ of an uncertain object X, which is assumed to be currently fetched from the distance browsing (Step 1). For this computation, we first need, for all uncertain objects $Z \in \mathcal{D}$, the probability $P_x(Z)$ that Z is closer to q than the current object instance x. These probabilities are stored in an *active object list* (AOL), which can easily be kept updated due to the following obvious lemma:

Lemma 1. *Let q be the query object and $(x, p(X = x))$ be the object instance of an object X fetched from the distance browsing in the current processing iteration. The probability that an object $Z \neq X$ is closer to q than x is*

$$P_x(Z) = \sum_{(z, P(Z=z)) \in Z} P(Z = z),$$

where $(z, P(Z = z))$ are the instances fetched in previous processing iterations.

Lemma 1 states that we can accumulate in overall linear space the sums of probabilities of all instances for each object, which have been seen so far, and use them to compute $P_x(Z)$ given the current instance x and any object Z in $\mathcal{D}$. In fact, we only need to manage in the list the probabilities of those objects for which we already have accessed an instance and for which we expect to access further instances in the remaining iterations.

Now let us see how we can use list AOL to efficiently compute the probabilities $P_i(x)$. Assume that $(x, P(X = x)) \in X$ is the current object instance reported from distance browsing. Let $\mathcal{S} = \{Z_1, \ldots, Z_j\}$ be the set of objects which has been seen so far, i.e., for which we already have seen at least one object instance. The probability that an object $X \in \mathcal{S}$ appears at ranking position i of the first j objects seen so far only depends on the event that $i - 1$ of the remaining $j - 1$ objects $Z \in \mathcal{S}$ $(Z \neq X)$ appear before X, no matter which of these objects fulfill this criterion. Let $\mathcal{S}$ denote the

3. Note that the $O(nlogn)$ factor is due to presorting the object instances according to their distances to the query object. If we assume that the instances are already sorted, then our framework can compute the probability distributions for the first k rank positions in $O(k \cdot n)$ time.

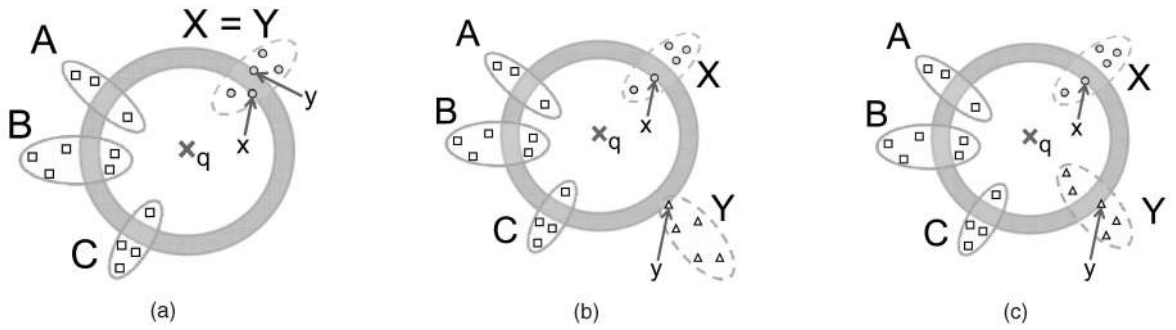

Fig. 3. Cases when updating the probabilities, assuming that x was the last processed instance and y is the current one. (a) Case 1: previous instance x and current instance y belong to the same object. (b) Case 2: instance y is the first returned instance of object Y. (c) Case 3: instance y is not the first returned instance of object Y and $X \neq Y$.

set of objects except for object X seen so far, i.e., $X \notin \mathcal{S}$. Furthermore, let $P_{i,\mathcal{S},x}$ denote the probability that exactly i objects of $\mathcal{S}$ are closer to q than the object instance x. Now, we can formulate the recursive function:

$$P_{i,\mathcal{S},x} = P_{i-1,\mathcal{S}\setminus\{Z\},x} \cdot P_x(Z) + P_{i,\mathcal{S}\setminus\{Z\},x} \cdot (1 - P_x(Z)),$$

where

$$P_{0,\emptyset,x} = 1 \quad \text{and} \quad P_{i,\mathcal{S},x} = 0, \quad \text{iff} \quad i > |\mathcal{S}| \vee i < 0. \qquad (2)$$

Let us note that the above recursion is also known as Poisson binomial recurrence[4] and has been used in this context by the authors of [26], [27], [13]. The approach in [13] applies the above recurrence on a slightly different problem, where independence is assumed among all the tuples.

The correctness of (2) can be shown by the following intuition: the event that i objects of $\mathcal{S}$ are closer to q than x occurs if one of the following conditions holds. In the case that an object $Z \in \mathcal{S}$ is closer to q than x, then $i-1$ objects of $\mathcal{S} \setminus \{Z\}$ must be closer to q. Otherwise, if we assume that object $Z \in \mathcal{S}$ is farther to q than x, then i objects of $\mathcal{S} \setminus \{X\}$ must be closer to q.

For each object instance $(x, P(X = x))$ reported from the distance browsing, we have to apply the recursive function as defined above. Specifically, we have to compute for each instance $(x, P(X = x))$ the probabilities $P_{i,\mathcal{S},x}$ for all $i \in \{0, \ldots, \min\{k, |\mathcal{S}|\}\}$ and for $j = |\mathcal{S}|$ subsets of $\mathcal{S}$. If $n = |\mathcal{D}|$, this has a cost factor of $O(k \cdot n)$ per object instance retrieved from the distance browsing, leading to a total cost of $O(k \cdot n^2)$. Assuming that k is a small constant, we have an overall runtime of $O(n^2)$.

In the following, we show how we can compute each $P_{i,\mathcal{S},x}$ in constant time by utilizing the probabilities computed for the previously accessed instance.

3.2 Incremental Probability Computation

Let $(x, P(X = x)) \in X$ and $(y, P(Y = y)) \in Y$ be two object instances consecutively returned from the distance browsing. Without loss of generality, let $(x, P(X = x))$ be returned before $(y, P(Y = y))$. Each of the probabilities $P_{i,\mathcal{S}\setminus\{Y\},y}$ ($i \in \{0, \ldots, |\mathcal{S} \setminus \{Y\}|\}$) can be computed from the probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$ in constant time. In fact, the probabilities $P_{i,\mathcal{S}\setminus\{Y\},y}$ can be computed by considering at most one recursion step backward.

4. To the best of our knowledge, the Poisson binomial recurrence was first introduced by Lange [18].

The following three cases have to be considered. The first two are easy to tackle and the third one is the most common and challenging one:

- **Case 1:** Both instances belong to the same object, i.e., $X = Y$.
- **Case 2:** Both instances belong to different objects, i.e., $X \neq Y$ and $(y, P(Y = y))$ is the first returned instance of object Y.
- **Case 3:** Both instances belong to different objects, i.e., $X \neq Y$ and $(y, P(Y = y))$ is not the first returned instance of object Y.

Now, we show how the probabilities $P_{i,\mathcal{S}\setminus\{Y\},y}$ for $i \in \{0, \ldots, |\mathcal{S} \setminus \{Y\}|\}$ can be computed in constant time considering the above cases that are illustrated in Fig. 3.

In the first case (cf., Fig. 3a), the probabilities $P_x(Z)$ and $P_y(Z)$ of all objects in $Z \in \mathcal{S} \setminus \{X\}$ are equal because the instances of objects in $\mathcal{S} \setminus \{X\}$ that appear within the distance range of q of y and within the distance range of x are identical. Since the probabilities $P_{i,\mathcal{S}\setminus\{Y\},y}$ and $P_{i,\mathcal{S}\setminus\{X\},x}$ only depend on $P_x(Z)$ for all objects $Z \in \mathcal{S} \setminus \{X\}$, it is obvious that $P_{i,\mathcal{S}\setminus\{Y\},y} = P_{i,\mathcal{S}\setminus\{X\},x}$ for all i.

In the second case (cf., Fig. 3b), we can exploit the fact that $P_{i,\mathcal{S}\setminus\{X\},x}$ does not depend on Y. Thus, given the probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$, we can easily compute the probability $P_{i,\mathcal{S}\setminus\{Y\},y}$ by incorporating the object X using the recursive equation (2):

$$P_{i,\mathcal{S}\setminus\{Y\},y} = P_{i-1,\mathcal{S}\setminus\{Y,X\},y} \cdot P_y(X) + P_{i,\mathcal{S}\setminus\{Y,X\},y} \cdot (1 - P_y(X)).$$

Since $\mathcal{S} \setminus \{Y, X\} = \mathcal{S} \setminus \{X, Y\}$ and no instance of any object in $\mathcal{S} \setminus \{X, Y\}$ appears within the distance range of q according to y but not within the range according to x (cf., Fig. 3b), the following equation holds:

$$P_{i,\mathcal{S}\setminus\{Y\},y} = P_{i-1,\mathcal{S}\setminus\{X,Y\},x} \cdot P_y(X) + P_{i,\mathcal{S}\setminus\{X,Y\},x} \cdot (1 - P_y(X)).$$

Furthermore, $P_{i-1,\mathcal{S}\setminus\{X,Y\},x} = P_{i-1,\mathcal{S}\setminus\{X\},x}$ because Y is not in the distance range according to x, and thus, $Y \notin \mathcal{S} \setminus \{X\}$. Now, the above equation can be reformulated:

$$P_{i,\mathcal{S}\setminus\{Y\},y} = P_{i-1,\mathcal{S}\setminus\{X\},x} \cdot P_y(X) + P_{i,\mathcal{S}\setminus\{X\},x} \cdot (1 - P_y(X)). \qquad (3)$$

All probabilities of the term on the right-hand side in (3) are known, and thus, $P_{i,\mathcal{S}\setminus\{Y\},y}$ can be computed in constant time assuming that the probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$ computed in the previous step have been stored for all $i \in \{0,\ldots,|\mathcal{S}\setminus\{X\}|\}$.

The third case (cf., Fig. 3c) is the general case, which is not as straightforward as the previous two cases and requires special techniques. Again, we assume that the probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$ computed in the previous step for all $i \in \{0,\ldots,|\mathcal{S}\setminus\{X\}|\}$ are known. Similar to Case 2, the probability $P_{i,\mathcal{S}\setminus\{Y\},y}$ is equal to:

$$P_{i,\mathcal{S}\setminus\{Y\},y} = P_{i-1,\mathcal{S}\setminus\{X,Y\},x}\cdot P_y(X) + P_{i,\mathcal{S}\setminus\{X,Y\},x}\cdot(1-P_y(X)). \quad (4)$$

Since the probability $P_y(X)$ is assumed to be known, now we are left with the computation of $P_{i,\mathcal{S}\setminus\{X,Y\},x}$ for all $i \in \{0,\ldots,|\mathcal{S}\setminus\{X,Y\}|\}$ by again exploiting (2):

$$P_{i,\mathcal{S}\setminus\{X\},x} = P_{i-1,\mathcal{S}\setminus\{X,Y\},x}\cdot P_x(Y) + P_{i,\mathcal{S}\setminus\{X,Y\},x}\cdot(1-P_x(Y)),$$

which can be resolved to

$$P_{i,\mathcal{S}\setminus\{X,Y\},x} = \frac{P_{i,\mathcal{S}\setminus\{X\},x} - P_{i-1,\mathcal{S}\setminus\{X,Y\},x}\cdot P_x(Y)}{1-P_x(Y)}. \quad (5)$$

With $i=0$, we have

$$P_{0,\mathcal{S}\setminus\{X,Y\},x} = \frac{P_{0,\mathcal{S}\setminus\{X\},x} - P_{-1,\mathcal{S}\setminus\{X,Y\},x}\cdot P_x(Y)}{1-P_x(Y)} = \frac{P_{0,\mathcal{S}\setminus\{X\},x}}{1-P_x(Y)},$$

because the probability $P_{-1,\mathcal{S}\setminus\{X,Y\},x}=0$ by definition (cf. (2)). The case $i=0$ can be solved assuming that $P_{0,\mathcal{S}\setminus\{X\},x}$ is known from the previous iteration step.

With the assumption that all probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$ for all $i \in \{1,\ldots,|\mathcal{S}\setminus\{X\}|\}$ and $P_x(Y)$ are available from the previous iteration step, we can use (5) to recursively compute $P_{i,\mathcal{S}\setminus\{X,Y\},x}$ $(1 \le i \le |\mathcal{S}\setminus\{X,Y\}|)$ using the previously computed $P_{i-1,\mathcal{S}\setminus\{X,Y\},x}$. Based on this recursive computation, we obtain all probabilities $P_{i,\mathcal{S}\setminus\{X,Y\},x}$ $(0 \le i \le |\mathcal{S}\setminus\{X,Y\}|)$, which can be used to compute the probabilities $P_{i,\mathcal{S}\setminus\{Y\},y}$ for all $0 \le i \le |\mathcal{S}\setminus\{X,Y\}|$ according to (4).

3.3 Runtime Analysis

Building on this case-based analysis for the cost of computing $P_{i,\mathcal{S}\setminus\{X\},x}$ for the currently accessed instance x of an object o, we now prove that we can compute the rank probabilities of all objects at cost $O(nk)$, where n is the number of object instances accessed. The following lemma suggests that the incremental cost per object instance access is $O(k)$:

Lemma 2. *Let $(x,P(X=x)) \in X$ and $(y,P(Y=y)) \in Y$ be two object instances consecutively returned from the distance browsing. Without loss of generality, let us assume that the instance $(x,P(X=x))$ was returned in the last iteration in which we computed the probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$ for all $0 \le i \le |\mathcal{S}\setminus\{X\}|$. The next iteration in which we fetch $(y,P(Y=y))$ the probabilities $P_{i,\mathcal{S}\setminus\{Y\},y}$ for all $0 \le i \le \min\{k,|\mathcal{S}\setminus\{Y\}|\}$ can be computed in $O(k)$ time and space.*

TABLE 2
Runtime Complexity Comparison of the Best Known Approaches to Our Own Approach

runtime table	no precomputed D	precomputed D
ours	O($n$log$n$+kn)	O(kn)
[26]	O(kn^2)	O(kn^2)
[5]	exponential	exponential
[23]	exponential	exponential

Proof. In Case 1, the probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$ and $P_{i,\mathcal{S}\setminus\{Y\},y}$ are equal for all $0 \le i \le min\{k,|\mathcal{S}\setminus\{Y\})|\}$. No computation is required ($O(1)$ time) and the result can be stored using at most $O(k)$ space.

In Case 2, the probabilities $P_{i,\mathcal{S}\setminus\{Y\},y}$ for all $0 \le i \le min\{k,|\mathcal{S}\setminus\{Y\})|\}$ can be computed according to (3) taking $O(k)$ time. This assumes that $P_{i,\mathcal{S}\setminus\{X\},x}$ have to be stored for all $0 \le i \le \min\{k,|\mathcal{S}\setminus\{Y\}|\}$, requiring at most $O(k)$ space.

In Case 3, we first have to compute and store the probabilities $P_{i,\mathcal{S}\setminus\{X,Y\},x}$ for all $0 \le i \le \min\{k,|\mathcal{S}\setminus\{X,Y\})|\}$ using the recursive function in (5). This can be done in $O(\min\{k,|\mathcal{S}\setminus\{X,Y\})|\})$ time and space. Next, the computed probabilities can be used to compute $P_{i,\mathcal{S}\setminus\{Y\},y}$ for all $0 \le i \le \min\{k,|\mathcal{S}\setminus\{Y\})|\}$ according to (4), which again takes at most $O(\min\{k,|\mathcal{S}\setminus\{X,Y\})|\})$ time and space. □

After giving the runtime evaluation of the processing of one single object instance, we are now able to extend the cost model for the whole query process. According to Lemma 2, we can assume that each object instance can be processed in constant time if we assume that k is constant. If we assume that the total number of object instances in our database is linear to the number of database objects, we would get a runtime complexity which is linear in the number of database objects, more exactly, $O(kn)$, where n is the size of the database and k the specified depth of the ranking. Up to now, our model assumes that the preprocessing step and the postprocessing step of our framework require at most linear runtime. Since the postprocessing step only includes an aggregation of the results generated in Step 2, the linear runtime complexity of Step 3 is guaranteed. Now, we want to examine the runtime of the object instance ranking in Step 1. Similar to the assumptions that hold for our competitors [23], [26], [5], we can also assume that the object instances are already sorted, which would involve linear runtime cost also for Step 1. However, for the general case where we have to initialize a distance browsing first, the runtime complexity of Step 1 would increase to O($n$log$n$). As a consequence, the total runtime cost of our approach (including distance browsing) sums up to O($n$log$n$+kn). An overview of the computation cost is given in Table 2.[5]

Regarding the space complexity of our approach, we have to store, for each object in the database, a vector of length k for the probabilistic ranking of size $O(kn)$. In addition, we have to store the AOL of at most size $O(n)$, yielding a total space complexity of $O(kn+n)=O(kn)$.

5. Note that the approach proposed in [23] uses a more general correlation model than the x-relational model. It allows more types of correlations between tuples, thus making the given problem harder.

```
Probabilistic Ranking(𝒟,q)

Input: Database 𝒟, Query Vector q
1   AOL = ∅
2   result = Matrix of zeros // size: |instances|*k
3   P_i(x) = [0,...,0] // Length k
4   P_i(y) = [0,...,0] // Length k
5
6   y = 𝒟.next
7   updateAOL(y)
8   P_i(x)[0]=1
9   Add P_i(x) to the first line of result.
10  FOR (𝒟 is not empty AND ∃p ∈ P_i(x): p > 0)
11    x = y
12    y = 𝒟.next
13    updateAOL(y)
14
15    CASE 1: (c.f. Figure 3(a))
16    IF (Y = X)
17      P_i(y) = P_i(x)
18    END-IF
19
20    CASE 2: (c.f. Figure 3(b))
21    ELS-IF (Y ∉ AOL)
22      P(X)=AOL.getProb(X)
23      P_i(y) = dynamicRound(P_i(x),P_y(X))])
24    END-IF
25
26    CASE 3: (c.f. Figure 3(c))
27    ELSE          // (Y != X)
28      P(X)=AOL.getProb(X)
29      P(Y)=AOL.getProb(Y)
30      adjustedProbs = adjustProbs(P_i(x),P_y(Y))
31      p-rank_y = dynamicRound(adjustedProbs,P_y(X))
32    END-IF
33
34    Add P_i(y) to the next line of result.
35    P_i(x) = P_i(y)
36  END-FOR
37  return result
38  END Probabilistic Ranking.
Output: Probabilistic Ranking (c.f. Definition )
```

Fig. 4. Pseudocode of our ranking algorithm.

Note that Yi et al. [26] compute a different ranking (cf., Section 5 for details) with a space complexity of $O(n)$. To compute a probabilistic ranking according to our definition, they [26] require $O(kn)$ space as well.

4 Probabilistic Ranking Algorithm

The pseudocode of the algorithm for the probabilistic ranking is illustrated in Fig. 4, providing the implementation details of the previously discussed steps. Our algorithm requires a query object q and a distance browsing operator D (cf., [12]), which allows us to iteratively access the object instances sorted in ascending order of their similarity distance to a query object.

First, we initialize the *Active Object List* (AOL), a data structure that contains one tuple $(X, P(X))$ for each object X that

- has previously been found in D, i.e., at least one instance of X has been processed and
- has not yet been completely processed, i.e., at least one instance of X has yet to be found,

associated with the sum $P(X)$ of probabilities of all its instances that have been found. The AOL offers two functionalities as follows:

- updateAOL(instance x): Adds the probability of x ($P(X = x)$) to $P(X)$, where X is the object that x belongs to.
- getProb(object X): Returns $P(X)$.

Note that it is mandatory that the position of a tuple $(X, P(X))$ can be found in constant time, in order to sustain the constant time complexity of an iteration. This can be

- approached by means of hashing or
- reached by giving each object X the information about the location of its corresponding instances ($P(X)$) at an additional space cost of $O(n)$.

We also keep the *result*, a matrix that contains, for each object instance x that has been found and each ranking position i, the probability $P_i(x)$ that x is located at ranking position i. Note that this result is instance-based. In order to get an object-based rank probability, we can aggregate instances belonging to the same object, using (1). Additionally, we initialize two arrays *p-rank*_x and *p-rank*_y, each of length k, which contain, at any iteration of the algorithm, the probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$ and $P_{i,\mathcal{S}\setminus\{Y\},y}$, respectively, for all $0 \leq i \leq k$. $x \in X$ is the instance found in the previous iteration and $y \in Y$ is the instance found in the current iteration (see Fig. 3).

In line 6, the algorithm starts by fetching the first object instance, which is the closest to the query q in the database. A tuple containing the corresponding object as well as the probability of this instance is added to the AOL.

Then, the first position of *p-rank*_x is set to 1 while all other $k-1$ positions remain at 0 because

$$P_{1,\mathcal{S}\setminus\{y\},y} = P_{1,\emptyset,y} = 1$$

and

$$P_{i,\mathcal{S}\setminus\{y\},y} = P_{i,\emptyset,y} = 0$$

for $i > 1$ by definition (see (2)). This simply reflects the fact that the first instance is always on rank 1. Note that *p-rank*_y is implicitly assigned to *p-rank*_x here.

Then, the first iteration of the main algorithm begins by fetching the next object instance from D. Now, we have to distinguish the three cases explained in Section 3.

In the first case (line 16), both the previous and current instances refer to the same object. As explained in Section 3, we have nothing to do in this case, since $P_{i,\mathcal{S}\setminus\{X\},x} = P_{i,\mathcal{S}\setminus\{Y\},y}$ for all $0 \leq i \leq k-1$.

In the second case (line 21), the current instance refers to an object that has not been seen yet. As explained in Section 3, we only have to apply an additional iteration of the DP algorithm (cf. (2)). This *dynamicRound* algorithm is shown in Fig. 5 and is used here to incorporate the probability that X is closer to y into *p-rank*_y in a single iteration of the dynamic algorithm.

In the third case (line 27), the current instance relates to an object that has already been seen. Thus, the probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$ depend on Y. As explained in Section 3, we first

```
dynamicRound(oldRanking,P_y(X))
Input: oldRanking: Intermediate result without object X
       P_y(X): Prob. that object X is closer to q than instance y.
1   newRanking = [0,...,0] // Length k
2   newRanking[0] =
3       oldRanking[0]*(1-P_y(X))
4   FOR i = 1,...,k-1
5     newRanking[i] =
6         oldRanking[i-1]*P_y(X)
7         +oldRanking[i]*(1-P_y(X))
8   END-FOR
9    return newRanking
10  END dynamicRound.
Output: Result including object X
```

Fig. 5. Pseudocode of a dynamic iteration at instance y.

```
adjustProbs(oldRanking,P_y(Y))
Input: oldRanking: Intermediate result including object Y
       P_y(X): Prob. that another instance of object Y is closer
to q than instance y.

1   adjustedRanking = [0,...,0] // Length k
2   adjustedProbs[0] =
3   oldRanking[0] / P_y(Y)
4   FOR i = 1,...,k-1
5     adjustedProbs[i] = (oldRanking[i]-oldRanking[i-1]*P_y(Y)) / (1-P_y(Y))
6   END-FOR
7   return adjustedProbs
8   END adjustProbs.
Output: Intermediate result at instace y excluding object Y
```

Fig. 6. Pseudocode of the algorithm that excludes one object Y from the current result at instance $y \in Y$.

have to filter out the influence of Y on $P_{i,\mathcal{S}\setminus\{X\},x}$ and compute $P_{i,\mathcal{S}\setminus\{X,Y\},x}$. This is performed by the *adjustProbs* algorithm in Fig. 6 utilizing the technique explained in Section 3. Using $P_{i,\mathcal{S}\setminus\{X,Y\},x}$, the algorithm then computes $P_{i,\mathcal{S}\setminus\{Y\},y}$ using a single iteration of the dynamic algorithm like in case 2.

At line 35, the computed ranking, for instance, y is added to the result. If the application (i.e., the ranking method) requires objects to be ranked instead of instances, then *p-rank*_y is used to incrementally update the probabilities of Y for each rank.

The algorithm continues fetching object instances from the distance browsing operator D and repeats this case analysis until either no more samples are left in $\mathcal{D}$ or until an object instance is found, for the probability zero for each of the first k positions. In the latter case, there exist k objects that are closer to k with a probability of one and the computation can be stopped because the same k objects must be closer to all further object instances in the database that have not yet been found.

5 Probabilistic Ranking Approaches

The method proposed in Section 3 efficiently computes for each uncertain object instance x_j and each ranking position i $(0 \leq i \leq k-1)$ the probability that x_j has the ith rank. However, most applications require an unique object ranking, i.e., each object (or object instance) is uniquely assigned to exactly one rank. Various top-k query approaches have been proposed generating deterministic rankings from probabilistic data, which we call probabilistic ranking queries. The question at issue is how our framework can be exploited in order to significantly accelerate probabilistic ranking queries. In the remainder, we show that our framework is able to support and significantly boost the performance of the state-of-the-art probabilistic ranking queries. Specifically, we demonstrate this by applying the state-of-the-art ranking approaches including U-k Ranks, PT-k, and *Global top-k*.

Note that the following ranking approaches are based on the x-relation model [3], [1]. As mentioned before, the x-relation model conceptionally corresponds to our uncertainty model, where the object instances correspond to the tuples and the uncertain vector objects correspond to the x-tuples. In the following, we use the terms *object instance* and *object*.

5.1 Expected Score and Expected Ranks

The *Expected Score* and *Expected Ranks* [10] compute for each object instance its expected score (rank) and rank the instances by this expected score (rank). *Expected Ranks* runs in $O(n \cdot log(n))$-time, thus outperforming exact approaches that do not use any estimation. The main drawback of this approach is that by using the expected value estimator, information is lost about the distribution of the objects. In the following, we will show how our framework can be used to accelerate the remaining state-of-the-art approaches, including U-kRanks, PT-k, and *Global top-k*, to $O(n \cdot logn + kn)$ runtime.

5.2 U-kRanks

The U-kRanks [23] approach reports the most likely object instance at each rank i, i.e., the instance that is most likely to be ranked ith over all possible worlds. This is essentially the same definition as proposed in $PRank$ in [19] in the context of distributions over spatial data. The approach proposed in [23] has exponential runtime. The runtime has been reduced to $O(n^2k)$ time in [27]. Using our framework, the problem of U-kRanks can be solved in $O(n \cdot log(n) + nk)$ time using the same space complexity as follows:

Use the framework to create the probabilistic ranking in $O(n \cdot log(n) + nk)$ as explained in the previous section. Then, for each rank i, find the object instance $argmax_j(p\_rank_q(X_j, i))$ that has the highest probability of appearing at rank i in $O(nk)$. This is performed by (cf., Fig. 7) finding for each rank i the object instance that has the highest probability to be assigned to rank i. Obviously, a problem of this problem definition is that a single object instance o_j may appear at more than one ranking position, or at no ranking position at all. For example, in Fig. 7, object instance A is ranked on both ranks 1 and 2, while object instance B is ranked nowhere. The total runtime for U-kRanks has thus been reduced from $O(n^2)$ to $O(nlog(n) + kn)$, that is, $O(n * log(n))$ if k is assumed to be constant.

5.3 PT-k

The *probabilistic threshold top-k* query (PT-k) [13] problem fixes the problem of the previous definition by aggregating the probabilities of an object instance x_j appearing at rank k or better. Given a user-specified probability threshold p, PT-k

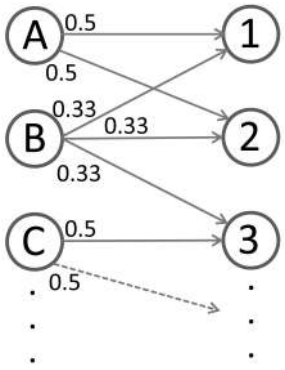

Fig. 7. Small example extract of a probabilistic ranking as produced by our framework.

returns all instances, which have a probability of at least p of being at rank k or better. Note that in this definition, the number of results is not limited to k and depends on the threshold parameter p. The model of PT-k consists of a set of instances and a set of generation rules that define mutually exclusiveness of instances. Each object instance occurs in one and only one generation rule. This model conceptionally corresponds to the x-relation model (with disjoint x-tupels). PT-k computes all result instances in $O(nk)$ time while also assuming that the instances are already presorted, thus having a total runtime of $O(nlog(n) + kn)$. The framework can be used to solve the PT-k problem in the following way.

We create the probabilistic ranking in $O(nk)$ as explained in the previous section. For each object instance x, we compute the probability that x appears at position k or better (in $O(nk)$). Formally, we return all instances $x \in \mathcal{D}$ for which:

$$\left\{ x \in \mathcal{D} \middle| \sum_{i=1}^{k} P_i(x) > p \right\}.$$

As seen in Fig. 7, this probability can simply be computed by aggregating all probabilities of an object instance to be ranked at k or better. For example, for $k = 2$ and $p = 0.5$, we get A and B as results. Note that for $p = 0.1$, further object instances may be in the result because there must be further object instances (from object instances that are left out here for simplicity) with a probability greater than 0 to rank 1 and rank 2, since the probability of their respective edges does not sum up to 1.0 yet.

Note that our framework is only able to match, not to beat the runtime of PT-k. However, using our approach, we can additionally return the ranking order, instead of just the top-k set.

5.4 Global top-k

Global top-k [29] is very similar to PT-k and ranks the object instances by their top-k probability, and then takes the top-k of these. This approach has a runtime of $O(n^2k)$. The advantage here is that, unlike in PT-k, the number of results is fixed, and there is no user-specified threshold parameter. Here, we can exploit the ranking order information that we acquired in the PT-k using our framework to solve *Global top-k* in $O(n \cdot log(n) + kn)$ time.

We use the framework to create the probabilistic ranking in $O(n \cdot log(n) + kn)$ as explained in the previous section. For each object instance x, we compute the probability that x appears at position k or better (in $O(nk)$) like in PT-k. Then, we find the k object instances with the highest probability in $O(k \cdot log(k))$.

6 Experimental Evaluation

We have performed extensive experiments to evaluate the performance of our proposed probabilistic ranking approach proposed in Section 3 w.r.t. the database size ($|\mathcal{D}|$) measured in the number of uncertain vector objects, ranking depth (k), and degree of uncertainty (UD) as defined below. In the following, the ranking framework is briefly denoted by **PSR**.

6.1 Data Sets and Experimental Setup

The probabilistic ranking was applied to a scientific real-world data set *SCI* and several artificial data sets *ART_X* of varying size and degree of uncertainty. All data sets are based on the discrete uncertainty model, i.e., each object is represented by a collection of vector samples.

The *SCI* data set is a set of 1,600 objects, where each object consists of 48 10-dimensional instances. Each instance corresponds to a set of environmental sensor measurements of one single day (one per 30 minutes) that consists of 10 dimensions (attributes): Temperature, humidity, speed and direction of wind w.r.t. degree and sector, as well as concentrations of CO, SO_2, NO, NO_2, and O_3. These attributes are normalized within the interval [0, 1] to give each attribute the same weight.

The *ART_1* data set consists of 1,000,000 objects, each consisting of 20 object instances for the scalability experiments. For the evaluation of the performance w.r.t. the ranking depth and the degree of uncertainty, we applied a collection *ART_2* of data sets each composing 10,000 objects. Each object is represented by a set of 20 three-dimensional instances. The *ART_2* data sets differ in the degree of uncertainty (UD) the corresponding objects have. The UD reflects the following distribution of object instances: each uncertain vector object is assumed to be located within a three-dimensional hyperrectangle. The object instances are uniformly distributed within the corresponding rectangle. In the following, we will refer to the side length of the rectangles as UD. The rectangles are uniformly distributed within a $10 \times 10 \times 10$ vector space. The *ART_3* data sets are very similar to *ART_3* data sets, except that the instances of object (again 10,000 objects uniformly distributed in the vector space with 20 instances each) follow a three-dimensional normal distribution. The data sets of *ART_3* vary in the degree of uncertain as well. For this data set, the degree of uncertain simply denotes the standard deviation of the normal distribution of the objects.

The degree of uncertainty is interesting in our performance evaluation since it is expected to have a significant influence on the runtime. The reason is that a higher degree of uncertainty obviously leads to a higher overlap between the objects, which influences the size of the AOL (cf., Section 4) during the distance browsing. The higher the object overlap, the more objects are expected to be in the AOL at a time. Since the size of the AOL influences the runtime of the rank probability computation, a higher degree of uncertainty is

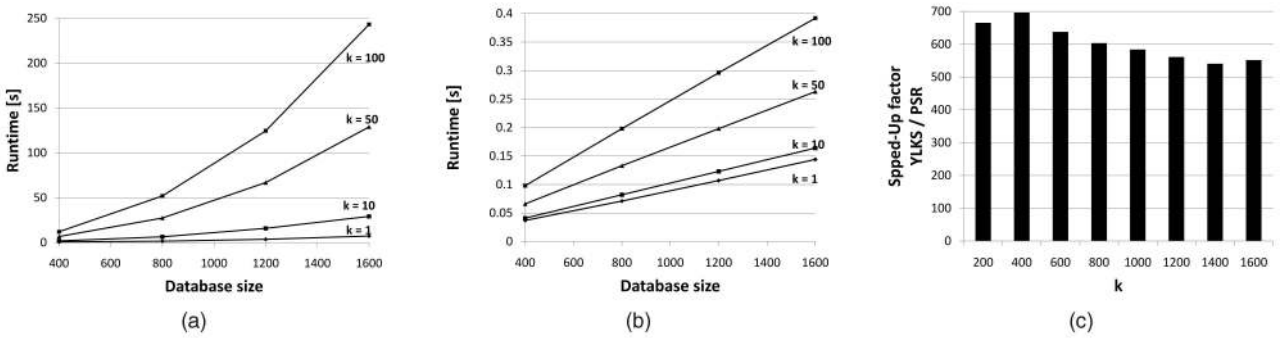

Fig. 8. Scalability evaluated on *SCI* for different k values. (a) **YLKS**. (b) **PSR**. (c) Speedup gain w.r.t. k on *SCI*.

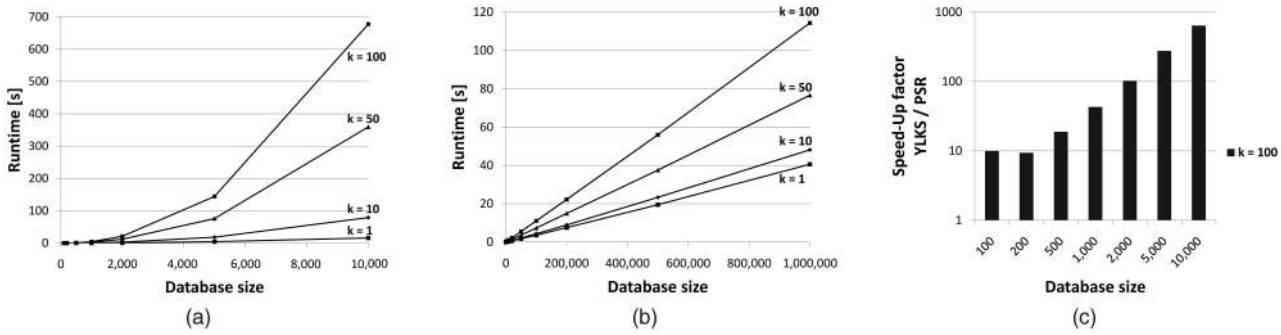

Fig. 9. Scalability evaluated on *ART_1* for different k values. (a) **YLKS**. (b) **PSR**. (c) Speedup factor.

expected to lead to a higher runtime. This is experimentally evaluated in Section 6.4.

6.2 Scalability

In this section, we give an overview of our experiments regarding the scalability of **PSR**. We compare our results to the dynamic programming-based rank probability computation used for the U-kRanks method as proposed by Yi et al. in [26]. This method in the following, denoted by **YLKS**, is the best approach currently known for solving the (instance-based) rank probability problem (cf., Table 2). For a fair comparison, we used the **PSR** framework to compute the same (instance-based) rank probability problem as described in Section 3. Let us note that the cost required to solve the object-based rank probability problem is similar to that required to solve the instance-based rank probability problem. This is because the former problem additionally only requires to build the sum over all instance-based rank probabilities, which can be done on the fly without additional cost. Furthermore, we can neglect the cost required to build a final definite ranking (e.g., the rankings proposed in Section 5) from the rank probabilities because they can be also computed on the fly by simple aggregations of the corresponding (instance-based) rank probabilities.

For the sorting of the distances of the instances to the query point, we used a tuned quicksort adapted from [4]. This algorithm offers $O(n \cdot log(n))$ performance on many data sets that cause other quicksort algorithms to degrade to quadratic runtime.

The results of our first scalability tests on the real data set *SCI* are depicted in Fig. 8. It can be observed in Fig. 8b that the runtime of the probabilistic ranking using the **PSR** framework increases linearly in the database size, whereas **YLKS** has a runtime quadratic in the database size in the same parameter settings (cf., Fig. 8a). We can also see that this effect persists for different settings of k. Note that the effect of the $O(n \cdot log(n))$ sorting of the distances of the instances is insignificant on this relatively small data set. The direct speedup of the rank probability computation using **PSR** in comparison to **YLKS** is depicted in Fig. 8c. It shows for different values of k the speedup factor that is defined as the ratio $\frac{runtime(\mathbf{YLKS})}{runtime(\mathbf{PSR})}$ describing the performance gain of **PSR** versus **YLKS**. It can be observed that, for a constant number of objects in the database ($|DB| = 1{,}600$), the ranking depth k has no impact on the speedup factor. This can be explained by the observation that both approaches scale linearly in k.

Next, we evaluate the scalability of the database size based on the *ART_1* data set. The results of this experiment are depicted in Fig. 9. Fig. 9b shows that we are able to perform ranking queries in a reasonable time of less than 120 seconds, even for very large database containing 1,000,000 and more

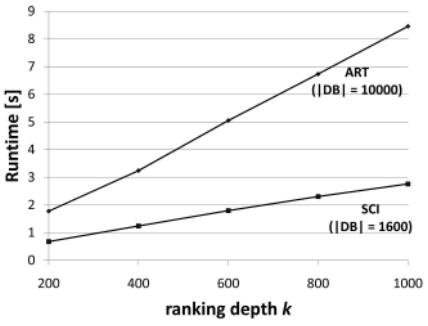

Fig. 10. Runtime using **PSR** on *SCI* and *ART*.

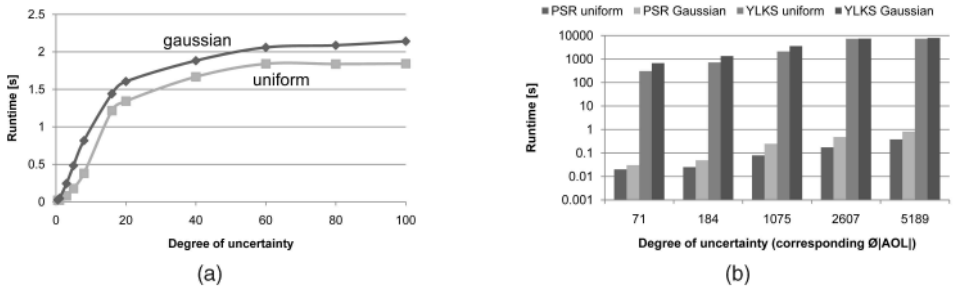

Fig. 11. Runtime w.r.t. the degree of uncertainty. (a) Evaluation of **PSR** by an increasing uncertainty degree. (b) **YLKS** versus **PSR** in a logarithmic scale w.r.t. different $\oslash(|AOL|)$ values.

objects, each having 20 instances (thus having a total of 20,000,000 instances (tuples)). Note that an almost perfect linear scale-up can be seen in Fig. 9 despite of the $O(n \cdot log(n))$ cost for sorting the database. This is due to the very efficient quicksort implementation in [4] that our experiments have shown to require only slightly worse than linear time.

In Fig. 9a, it can be observed that due to the quadratic scaling of the **YLKS** algorithm, it is inapplicable for relatively small databases of size 5,000 or more. The direct speedup of the rank probability computation using **PSR** in comparison to **YLKS** for varying database size is depicted in Fig. 9c. Here, we can see that the speedup of our approach in comparison to **YLKS** increases linearly with the size of the database which is consistent with our runtime analysis in Section 3.

6.3 Ranking Depth k

The influence of the ranking depth k on the runtime performance of our probabilistic ranking method **PSR** is studied in the next experiment. As depicted in Fig. 10, where the experiments were performed using both the *SCI* and the *ART* data set, the influence of an increasing k yields a linear effect on the runtime of **PSR**, but does not depend on the type of the data set. This effect can be explained by taking into consideration that each iteration of Case 2 or Case 3 requires a probability computation for each ranking position $0 \leq i \leq k$.

6.4 Influence of the Degree of Uncertainty

In the next experiment, we varied the uncertainty degree of objects using the *ART_2* and *ART_3* data sets. In the following experiments, the ranking depth is set to a fixed value of $k = 100$. As previously discussed, a varying degree of uncertainty leads to an increase in the overlap between the instances of the objects, and thus, objects will remain in the AOL for a longer time. The influence of the degree of uncertainty depends on the probabilistic ranking algorithm. This statement is underlined by the experiments shown in Fig. 11. It can be seen in Fig. 11a that **PSR** scales superlinearly in the degree of uncertainty at first, until a maximal value is reached. This maximal value is reached when the degree of uncertainty becomes so large that the instances of an object cover the whole vector space. In this case, objects remain on the *AOL* until almost the whole database is processed in most cases due to the increased overlap of object instances. In this case of extremely high uncertainty, almost no spatial pruning can be performed, slowing down the algorithm by several orders of magnitude. It is also worth noting that in our setting, the algorithm performs worse on Gaussian distributed data than on uniformly distributed data. This is explained by the fact that the space covered by a normal distribution with standard deviation x in each dimension is generally larger than a hyperrectangle with a side length of x in each dimension. A comparison of the runtime of **YLKS** and **PSR** w.r.t. the average AOL size is depicted in Fig. 11b for both the uniform and the normal distributed data sets. The degree of uncertain has a similar influence on both **YLKS** and **PSR**.

6.5 Summary

The experiments presented in this section show that the theoretical analysis of our approach given in Section 5 can be confirmed empirically on both artificial and real-world data. The performance studies showed that our framework computing the rank probabilities indeed reduces the quadratic runtime complexity of the state-of-the-art approaches to linear. Note that the cost required to presort the object instances is neglected in our settings. It could be shown that our approach scales very well even for large databases. The speedup gain of our approach w.r.t. the rank depth k has shown to be constant, which proofs that both approaches scale linearly in k. Furthermore, we could observe that our approach is applicable for databases with a high degree of uncertainty (i.e., the degree of variance of the instance distribution).

7 Conclusions

In this paper, we proposed a framework for efficient computation of probabilistic similarity ranking queries in uncertain vector databases. We introduced a novel concept that achieves a log-linear runtime complexity in contrast to the best known existing approach that solves the same problem with quadratic runtime complexity. Our concepts are theoretically and empirically proved to be superior to all existing approaches. In an experimental evaluation, we showed that our approach scales very well, and thus, is applicable even for large databases. As future work, we plan to extend the concepts proposed in this paper to further uncertainty models.

References

[1] P. Agrawal, O. Benjelloun, A. Das Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom, "Trio: A System for Data, Uncertainty, and Lineage," *Proc. Int'l Conf. Very Large Databases (VLDB '06)*, 2006.

[2] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and Simple Relational Processing of Uncertain Data," *Proc. 24th Int'l Conf. Data Eng. (ICDE '08)*, 2008.

[3] O. Benjelloun, A.D. Sarma, A. Halevy, and J. Widom, "ULDBs: Databases with Uncertainty and Lineage," *Proc. Int'l Conf. Very Large Databases (VLDB '06)*, pp. 1249-1264, 2006.

[4] J.L. Bentley and M. Douglas McIlroy, "Engineering a Sort Function," *Software—Practice & Experience,* vol. 23, pp. 1249-1265, 1993.

[5] T. Bernecker, H.-P. Kriegel, and M. Renz, "Proud: Probabilistic Ranking in Uncertain Databases," *Proc. 20th Int'l Conf. Scientific and Statistical Database Management,* July 2008.

[6] C. Böhm, A. Pryakhin, and M. Schubert, "Probabilistic Ranking Queries on Gaussians," *Proc. Int'l Conf. Scientific and Statistical Database Management,* pp. 169-178, 2006.

[7] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Evaluating Probabilistic Queries over Imprecise Data," *Proc. ACM SIGMOD,* pp. 551-562, 2003.

[8] 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),* 2005.

[9] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. Vitter, "Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data," *Proc. 30th Int'l Conf. Very Large Databases (VLDB '04),* pp. 876-887, 2004.

[10] G. Cormode, F. Li, and K. Yi, "Semantics of Ranking Queries for Probabilistic Data and Expected Results," *Proc. 25th Int'l Conf. Data Eng. (ICDE '09),* pp. 305-316, Mar./Apr. 2009.

[11] N. Dalvi and D. Suciu, "Efficient Query Evaluation on Probabilistic Databases," *The VLDB J.,* vol. 16, no. 4, pp. 523-544, 2007.

[12] G.R. Hjaltason and H. Samet, "Ranking in Spatial Databases," *Proc. Fourth Int'l Symp. Advances in Spatial Databases,* M.J. Egenhofer and J.R. Herring, eds., pp. 83-95, 1995.

[13] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach," *Proc. ACM SIGMOD,* pp. 673-686, June 2008.

[14] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz, "Probabilistic Similarity Join on Uncertain Data," *Proc. 11th Int'l Conf. Database Systems for Advanced Applications,* pp. 295-309, 2006.

[15] H.-P. Kriegel, P. Kunath, and M. Renz, "Probabilistic Nearest-Neighbor Query on Uncertain Objects," *Proc. 12th Int'l Conf. Database Systems for Advanced Applications,* 2007.

[16] H.-P. Kriegel, M. Renz, M. Schubert, and A. Züfle, "Statistical Density Prediction in Traffic Networks," *Proc. Eighth SIAM Conf. Data Mining (SDM '08),* 2008.

[17] H.-P. Kriegel, B. Seeger, R. Schneider, and N. Beckmann, "The r*-Tree: An Efficient Access Method for Geographic Information System," *Proc. Int'l Conf. Geographic Information Systems,* 1990.

[18] K. Lange, *Numerical Analysis for Statisticians.* Springer, 1999.

[19] X. Lian and L. Chen, "Probabilistic Ranked Queries in Uncertain Databases," *Proc. 11th Int'l Conf. Extending Database Technology (EDBT '08),* pp. 511-522, Mar. 2008.

[20] C. Re, N. Dalvi, and D. Suciu, "Efficient Top-K Query Evaluation on Probabilistic Databases," *Proc. 23rd Int'l Conf. Data Eng. (ICDE '07),* 2007.

[21] P. Sen and A. Deshpande, "Representing and Querying Correlated Tuples in Probabilistic Databases," *Proc. 23rd Int'l Conf. Data Eng. (ICDE '07),* 2007.

[22] M. Soliman and I. Ilyas, "Ranking with Uncertain Scores," *Proc. 25th Int'l Conf. Data Eng. (ICDE '09),* pp. 317-328, Mar./Apr. 2009.

[23] M. Soliman, I. Ilyas, and K. Chen-Chuan Chang, "Top-k Query Processing in Uncertain Databases," *Proc. 23rd Int'l Conf. Data Eng. (ICDE '07),* pp. 896-905, Apr. 2007.

[24] Y. Tao, R. Cheng, X. Xiao, W. 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.

[25] O. Wolfson, P. Sistla, S. Chamberlain, and Y. Yesha, "Updating and Querying Databases That Track Mobile Units," *Distributed and Parallel Databases,* vol. 7, no. 3, pp. 257-387, 1999.

[26] K. Yi, F. Li, G. Kollios, and D. Srivastava, "Efficient Processing of Top-K Queries in Uncertain Databases," *Proc. 24th Int'l Conf. Data Eng. (ICDE '08),* pp. 1406-1408, Apr. 2008.

[27] K. Yi, F. Li, G. Kollios, and D. Srivastava, "Efficient Processing of Top-K Queries in Uncertain Databases with X-Relations," *IEEE Trans. Knowledge and Data Eng.,* vol. 20, no. 12, pp. 1669-1682, Dec. 2008.

[28] M.L. Yiu, N. Mamoulis, X. Dai, Y. Tao, and M. Vaitis, "Efficient Evaluation of Probabilistic Advanced Spatial Queries on Existentially Uncertain Data," *IEEE Trans. Knowledge and Data Eng.,* vol. 21, no. 1, pp. 108-122, Jan. 2009.

[29] X. Zhang and J. Chomicki, "On the Semantics and Evaluation of Top-K Queries in Probabilistic Databases," *Proc. 24th Int'l Conf. Data Eng. Workshops (ICDE '08),* pp. 556-563, Apr. 2008.

Thomas Bernecker is an academic assistant in the Database Systems and Data Mining Group of Hans-Peter Kriegel at the Ludwig-Maximilians-Universität München, Germany. His research interests include query processing in uncertain databases, spatiotemporal data mining, and similarity search in spatial, temporal, and multimedia databases.

Hans-Peter Kriegel is a full professor of database systems and data mining in the Institute for Informatics at the Ludwig-Maximilians-Universität München, Germany, and has served as the department chair or vice chair over the last years. His research interests are in spatial and multimedia database systems, particularly in query processing, performance issues, similarity search, high-dimensional indexing, as well as knowledge discovery and data mining. He has published more than 300 refereed conference and journal papers. He received the "SIGMOD Best Paper Award" 1997 and the "DASFAA Best Paper Award" 2006 together with the members of his Research Team.

Nikos Mamoulis is an associate professor in the Department of Computer Science at the University of Hong Kong, which he joined in 2001. In the past, he has worked as a research and development engineer at the Computer Technology Institute, Patras, Greece, and as a postdoctoral researcher at the Centrum voor Wiskunde en Informatica (CWI), the Netherlands. During 2008 and 2009, he was on leave to the Max-Planck Institute for Informatics (MPII), Germany. His research focuses on the management and mining of complex data types, including spatial, spatiotemporal, object-relational, multimedia, text, and semistructured data. He has served on the program committees of more than 70 international conferences and workshops on data management and data mining. He was the general chair of the SSDBM 2008, the PC chair of the SSTD 2009, and he organized the SSTDM 2006 and DBRank 2009 workshops. He has served as PC vice chair of the ICDM 2007, ICDM 2008, and CIKM 2009. He was the publicity chair of the ICDE 2009. He is an editorial board member for *Geoinformatica* journal and was a field editor of the *Encyclopedia of Geographic Information Systems.*

Matthias Renz is an assistant professor in the Database Systems and Data Mining Group of Hans-Peter Kriegel at the Ludwig-Maximilians-Universität München, Germany. He finished the PhD thesis on query processing in spatial and temporal data in Winter 2006. His research interests include query processing in uncertain databases, spatiotemporal data mining, and similarity search in spatial, temporal, and multimedia databases.

Andreas Zuefle is an academic assistant in the Database Systems and Data Mining Group of Hans-Peter Kriegel at the Ludwig-Maximilians-Universität München, Germany. His research interests include query processing in uncertain databases, spatiotemporal data mining, and similarity search in spatial, temporal, and multimedia databases.