Similarity Processing in Multi-Observation Data

Thomas Bernecker

München 2012

Similarity Processing in Multi-Observation Data

Thomas Bernecker

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Thomas Bernecker
aus München

München, den 24.08.2012

Erstgutachter: Prof. Dr. Hans-Peter Kriegel,
Ludwig-Maximilians-Universität München
Zweitgutachter: Prof. Mario Nascimento, University of Alberta
Tag der mündlichen Prüfung: 21.12.2012

Abstract

Many real-world application domains such as sensor-monitoring systems for environmental research or medical diagnostic systems are dealing with data that is represented by multiple observations. In contrast to *single-observation data*, where each object is assigned to exactly one occurrence, *multi-observation data* is based on several occurrences that are subject to two key properties: *temporal variability* and *uncertainty*. When defining similarity between data objects, these properties play a significant role. In general, methods designed for single-observation data hardly apply for multi-observation data, as they are either not supported by the data models or do not provide sufficiently efficient or effective solutions. Prominent directions incorporating the key properties are the fields of time series, where data is created by temporally successive observations, and uncertain data, where observations are mutually exclusive. This thesis provides research contributions for similarity processing – similarity search and data mining – on time series and uncertain data.

The first part of this thesis focuses on similarity processing in time series databases. A variety of similarity measures have recently been proposed that support similarity processing w.r.t. various aspects. In particular, this part deals with time series that consist of periodic occurrences of patterns. Examining an application scenario from the medical domain, a solution for activity recognition is presented. Finally, the extraction of feature vectors allows the application of spatial index structures, which support the acceleration of search and mining tasks resulting in a significant efficiency gain. As feature vectors are potentially of high dimensionality, this part introduces indexing approaches for the high-dimensional space for the full-dimensional case as well as for arbitrary subspaces.

The second part of this thesis focuses on similarity processing in probabilistic databases. The presence of uncertainty is inherent in many applications dealing with data collected by sensing devices. Often, the collected information is noisy or incomplete due to measurement or transmission errors. Furthermore, data may be rendered uncertain due to privacy-preserving issues with the presence of confidential information. This creates a number of challenges in terms of effectively and efficiently querying and mining uncertain data. Existing work in this field either neglects the presence of dependencies or provides only approximate results while applying methods designed for certain data. Other approaches dealing with uncertain data are not able to provide efficient solutions. This part presents query processing approaches that outperform existing solutions of probabilistic similarity ranking. This part finally leads to the application of the introduced techniques to data mining tasks, such as the prominent problem of probabilistic frequent itemset mining.

Zusammenfassung

Viele Anwendungsgebiete, wie beispielsweise die Umweltforschung oder die medizinische Diagnostik, nutzen Systeme der Sensorüberwachung. Solche Systeme müssen oftmals in der Lage sein, mit Daten umzugehen, welche durch mehrere Beobachtungen repräsentiert werden. Im Gegensatz zu Daten mit nur einer Beobachtung (*Single-Observation Data*) basieren Daten aus mehreren Beobachtungen (*Multi-Observation Data*) auf einer Vielzahl von Beobachtungen, welche zwei Schlüsseleigenschaften unterliegen: *Zeitliche Veränderlichkeit* und *Datenunsicherheit*. Im Bereich der Ähnlichkeitssuche und im Data Mining spielen diese Eigenschaften eine wichtige Rolle. Gängige Lösungen in diesen Bereichen, die für *Single-Observation Data* entwickelt wurden, sind in der Regel für den Umgang mit mehreren Beobachtungen pro Objekt nicht anwendbar. Der Grund dafür liegt darin, dass diese Ansätze entweder nicht mit den Datenmodellen vereinbar sind oder keine Lösungen anbieten, die den aktuellen Ansprüchen an Lösungsqualität oder Effizienz genügen. Bekannte Forschungsrichtungen, die sich mit *Multi-Observation Data* und deren Schlüsseleigenschaften beschäftigen, sind die Analyse von Zeitreihen und die Ähnlichkeitssuche in probabilistischen Datenbanken. Während erstere Richtung eine zeitliche Ordnung der Beobachtungen eines Objekts voraussetzt, basieren unsichere Datenobjekte auf Beobachtungen, die sich gegenseitig bedingen oder ausschließen. Diese Dissertation umfasst aktuelle Forschungsbeiträge aus den beiden genannten Bereichen, wobei Methoden zur Ähnlichkeitssuche und zur Anwendung im Data Mining vorgestellt werden.

Der erste Teil dieser Arbeit beschäftigt sich mit Ähnlichkeitssuche und Data Mining in Zeitreihendatenbanken. Insbesondere werden Zeitreihen betrachtet, welche aus periodisch auftretenden Mustern bestehen. Im Kontext eines medizinischen Anwendungsszenarios wird ein Ansatz zur Aktivitätserkennung vorgestellt. Dieser erlaubt mittels Merkmalsextraktion eine effiziente Speicherung und Analyse mit Hilfe von räumlichen Indexstrukturen. Für den Fall hochdimensionaler Merkmalsvektoren stellt dieser Teil zwei Indexierungsmethoden zur Beschleunigung von Ähnlichkeitsanfragen vor. Die erste Methode berücksichtigt alle Attribute der Merkmalsvektoren, während die zweite Methode eine Projektion der Anfrage auf eine benutzerdefinierten Unterraum des Vektorraums erlaubt.

Im zweiten Teil dieser Arbeit wird die Ähnlichkeitssuche im Kontext probabilistischer Datenbanken behandelt. Daten aus Sensormessungen besitzen häufig Eigenschaften, die einer gewissen Unsicherheit unterliegen. Aufgrund von Mess- oder Übertragungsfehlern sind gemessene Werte oftmals unvollständig oder mit Rauschen behaftet. In diversen Szenarien, wie beispielsweise mit persönlichen oder medizinisch vertraulichen Daten, können

Daten auch nachträglich von Hand verrauscht werden, so dass eine genaue Rekonstruktion der ursprünglichen Informationen nicht möglich ist. Diese Gegebenheiten stellen Anfragetechniken und Methoden des Data Mining vor einige Herausforderungen. In bestehenden Forschungsarbeiten aus dem Bereich der unsicheren Datenbanken werden diverse Probleme oftmals nicht beachtet. Entweder wird die Präsenz von Abhängigkeiten ignoriert, oder es werden lediglich approximative Lösungen angeboten, welche die Anwendung von Methoden für sichere Daten erlaubt. Andere Ansätze berechnen genaue Lösungen, liefern die Antworten aber nicht in annehmbarer Laufzeit zurück. Dieser Teil der Arbeit präsentiert effiziente Methoden zur Beantwortung von Ähnlichkeitsanfragen, welche die Ergebnisse absteigend nach ihrer Relevanz, also eine Rangliste der Ergebnisse, zurückliefern. Die angewandten Techniken werden schließlich auf Problemstellungen im probabilistischen Data Mining übertragen, um beispielsweise das Problem des *Frequent Itemset Mining* unter Berücksichtigung des vollen Gehalts an Unsicherheitsinformation zu lösen.

Contents

Part I

Preliminaries

Chapter 1

Introduction

1.1 Preliminaries

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 a broad range of applications including molecular biology [19], medical imaging [129] and multimedia databases [185] as well as data retrieval and decision support systems. At the same time, improvements in our ability to capture and store data has lead to massive datasets with complex structured data, which require special methodologies for efficient and effective data exploration tasks.

The exploration of data and the goal of obtaining knowledge that is implicitly present is part of the field of *Knowledge Discovery in Databases (KDD)*. KDD is the process of extracting new, valid and potentially useful information from data, which can be further processed by diverse applications [94]. The general steps of the KDD process are illustrated in Figure 1.1.

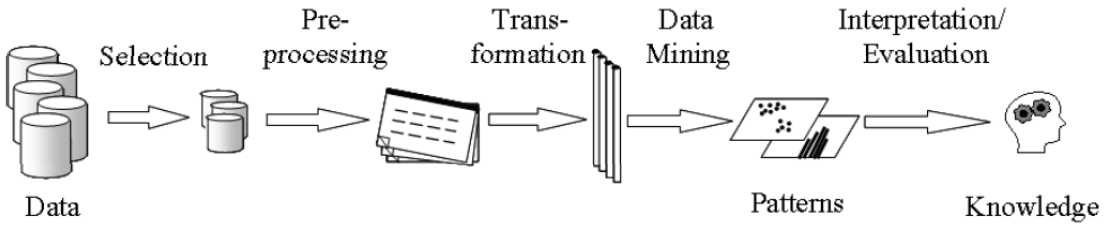


Figure 1.1: Visualization of the KDD process [91].

Following the process description of Ester and Sander [91], the first steps are selection of relevant data from the database, and preprocessing it in order to fill gaps or to combine data derived from different sources. Furthermore, a transformation is performed, which leads to a suitable representation of the data for the targeted application. The actual data mining step uses algorithms that extract patterns from the data, which are finally evaluated by the user.

Well-known data mining tasks are

- the field of clustering, where objects with similar characteristics are grouped, such that the similarity of objects within a cluster is maximized, while the similarity between different clusters is minimized;
- outlier detection, where the objective is to find objects that are not assigned to a cluster;
- classification, where objects are assigned to most appropriate class labels based on the learning effects obtained with previously assigned objects;
- rule mining, where, given a database of transactions, correlations and dependencies are examined by retrieving association rules.

These data mining tasks are strongly connected to applications which take advantage of their output, i.e., from the gained patterns contained in the data. Applications that will be part of this thesis are the following.

Example 1.1 *Prevention of diseases is an important part of medical research. In order to supervise the presence of physical health, methods of medical monitoring provide reliable evidence. In some cases, patients are required to fulfill a particular quota of physical activity, which can be captured via sensing devices. Finally, recognition of activity requires applying classification.*

Example 1.2 *Rule mining is commonly applied to market-basket databases for the analysis of consumer purchasing behavior. Such databases consist of a set of transactions, each containing the items a customer purchased. The most important and computationally intensive step in the mining process is the extraction of frequent itemsets – sets of items that occur in a specified minimum number of transactions.*

Many data mining tasks are based on the similarity of objects. This may, for example, be the case in activity recognition, where a clustering method or a similarity-based classification technique requires determining similarity between objects. This step, the *similarity query*, is not only useful to support the KDD process, but is also important in the context of content-based multimedia retrieval or proximity search. For example, starting from 2001, the popular search engine Google has provided the possibility to retrieve similar images to a selected reference image[1]. Regarding proximity search in geospatial applications, location-based services provide a list of relevant points of interest specified by the user, based on similarity queries w.r.t. the user's current location.

An overview of the basics needed for similarity processing, i.e., for the determination of similarity between objects in order to answer similarity queries and to solve data mining tasks that are based on the similarity among objects, will be given in the following section. This also contains a summary of most commonly used similarity query types.

[1] *Google images:* `http://images.google.com/`

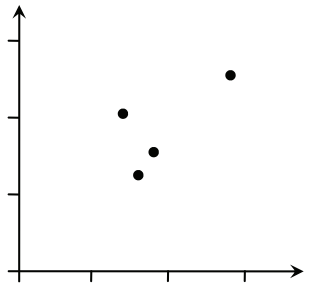


Figure 1.2: Vector objects with their spatial representation, $d = 2$.

1.2 Similarity Processing in Databases

1.2.1 Similarity of Data Objects

The definition of similarity between data objects requires an appropriate object representation. The most prevalent model is to represent objects in a d-dimensional vector space $\mathbb{R}^d$, $d \in \mathbb{N}$, also called *feature space*. An object then corresponds to a d-dimensional feature vector, illustrated as a single point, as depicted in Figure 1.2. The similarity between two d-dimensional objects x and y is commonly reflected by a distance function $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_0^+$, which is one of the L_p-norms ($p \in [1, \infty)$), formally:

$$dist_p = \sqrt[p]{\sum_{i=1}^{d} |x_i - y_i|^p}, \tag{1.1}$$

where x_i (y_i) denotes the value of x (y) in dimension i. In the following, the notation *dist* will denote the currently used L_p-norm, where the most prominent example, namely the Euclidean distance, will be used in the most cases ($p = 2$). An important property of the L_p-norm is that it is a metric, which implies that the triangle inequality is fulfilled. This property can be exploited in order to accelerate the performance of similarity queries.

1.2.2 Similarity Queries: A Short Overview

Basically, in a similarity query, the distance between a query object $q \in \mathcal{D}$ and each database object $x \in \mathcal{D}$ is computed in order to return all objects that satisfy the corresponding query predicate. This work deals with the most prominent query types, which are described in the following.

- An *ε-range query* retrieves the set $RQ(\varepsilon, q)$ that contains all objects from $x \in \mathcal{D}$ for which the following condition holds:

  $$\forall x \in RQ(\varepsilon, q) : dist(x, q) \leq \varepsilon.$$

  ε-range queries are, for example, used with density-based clustering methods, such as *DBSCAN* [90] and *OPTICS* [14], where objects are examined whether they build dense regions and, therefore, generate a clustered structure of the data.

- A *nearest neighbor (NN) query* retrieves the object $x \in \mathcal{D}$ for which the following condition holds:

$$x \in NN(q) \Leftrightarrow \forall y \in \mathcal{D} \setminus \{x\} : dist(x, q) \leq dist(y, q).$$

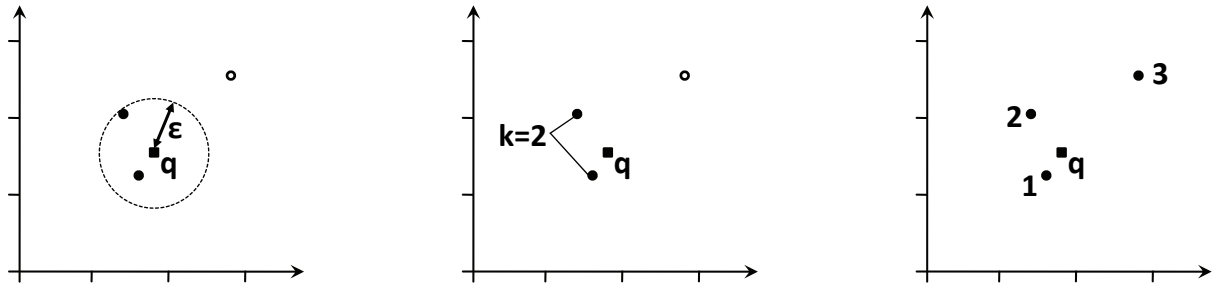
- The NN query can be generalized to the *k-nearest neighbor (k-NN) query*, which retrieves the set $NN(k, q)$ that contains k objects from $x \in \mathcal{D}$ for which the following condition holds:

$$\forall x \in NN(k, q), \forall y \in \mathcal{D} \setminus NN(k, q) : dist(x, q) \leq dist(y, q).$$

  k-NN queries are more user-friendly and more flexible than ε-range queries. Choosing the number k of results that shall be returned by a query is usually much more intuitive than selecting some query radius ε. In addition, many applications like data mining algorithms that further process the results of similarity queries require to control the cardinality of query results [137]. k-NN queries can easily be translated into ε-range queries yielding the same result set, setting the ε parameter to the distance of the query point to its kth nearest neighbor (the *k-NN distance*). One direct use of k-NN queries in data mining is in similarity-based classification tasks, e.g., in the k-NN classification, where k-NN queries are used to classify data items of unknown labels to class labels corresponding to the most similar labeled item.

- A variant of the NN query is the *reverse nearest neighbor (RNN) query*. Given a set of objects and a query object q, an RNN query returns all objects which have q as their nearest neighbor. Analogously to the NN query, the RNN query can be generalized to the Rk-NN query. The works of [35, 36] further generalizes the RNN query for arbitrary query predicates as well as multiple query objects by defining *inverse queries*. Given a subset of database objects $Q \subset \mathcal{D}$ and a query predicate, an inverse query returns all objects that contain Q in their result. Among others, solutions are proposed for inverse ε-range queries, and inverse k-NN queries. Reverse and inverse queries will not be explained in detail, as this is out of scope of this thesis.

- Finally, a *ranking query* iteratively retrieves objects $x \in \mathcal{D}$ in ascending order w.r.t. their distance to a query object. Similarity ranking is one of the most important operations in feature databases, e.g., for search engines, where ranking is used to report the most relevant object first. The iterative computation of answers is very suitable for retrieving results the user could have in mind. This is a big advantage of ranking queries over ε-range and k-NN queries, in particular if the user does not know how to specify the query parameters ε and k. Nevertheless, the parameter k can be used to limit the size of the ranking result (also denoted as *ranking depth*), similarly to the k-NN predicate, but retaining the ordering of results. For example, a ranking query returns the contents of a spatial object set specified by a user (e.g., the k nearest restaurants) in ascending order of their distance to a reference location. In another example in a database of images, a ranking query retrieves feature vectors of



(a) ε-range query. (b) k-NN query. (c) Ranking query.

Figure 1.3: Visualization of the most prominent query types, $d = 2$.

images in ascending order of their distance (i.e., dissimilarity) to a query image and returns the k most similar images. The restriction of the output to a ranking depth allows an early pruning of true drops in the context of multi-step query processing in order to accelerate similarity search algorithms.

- Contrary to the common ranking query, a *probabilistic inverse ranking query* [152] determines the rank for a given query object according to a given, user-defined score function f_{score}, and, thus, rates the significance of the query object among peers.

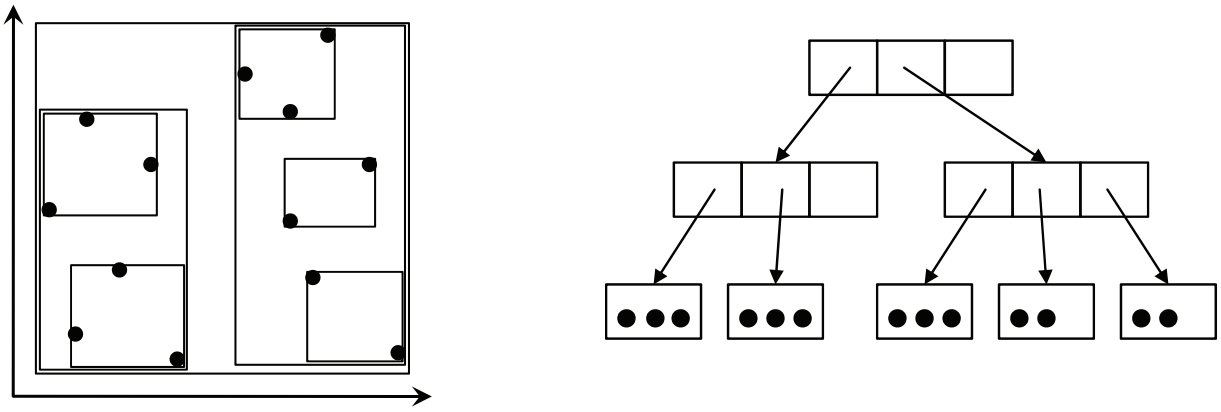
In the general case of relational data, query results are often determined w.r.t. a score function, where the distance to a query object is a special case (i.e., a high score value is reflected by a low spatial distance value). A popular example is the *top-k* query [92], where the objective is to retrieve the k objects with the highest combined (e.g., average) scores, out of a given set of objects that are ranked according to m different ranking or score functions (e.g., different rankings for m different attributes).

Examples for the query types ε-range, k-NN and ranking are visualized in Figure 1.3.

1.2.3 Efficient Processing of Similarity Queries

The acceleration of similarity queries via index structures in an important part in the context of similarity search. A straightforward solution performs a sequential scan of all objects, i.e., computes the distances from the query object to all database objects. Based on these computations, objects that satisfy the query predicate are returned. This solution is, however, very inefficient, yielding a runtime complexity which is linear in the size of the database. The goal of efficient processing techniques is to reduce cost required for distance computations (CPU cost) and read operations on the database (I/O cost).

Using an index structure, the number of objects that have to be accessed can be significantly reduced [52]. Common approaches comprise data-organizing indexes like tree structures (e.g., the *B-tree* [22]) or space-organizing structures like hashing [144, 161]. Popular and commonly used index structures for multidimensional spaces are the variants of the *R-tree* [101], as they showed to perform superior to other structures. The most prominent example here is the *R*-tree* [23], which will also be used in this work.



(a) R-tree MBRs.

(b) R-tree structure.

Figure 1.4: Visualization of an R-tree structure, $d = 2$.

Tree-based structures for multidimensional spaces group objects of spatial proximity and bound each group by a minimum bounding rectangle (MBR), which yields lower and upper approximations of the distance of these objects to a query object. MBRs are further recursively grouped and bounded, yielding a hierarchy of MBRs, where the hierarchically highest MBR represents the root of the tree, comprising the whole data space (cf. Figure 1.4). For efficiently answering similarity queries, the tree is traversed; search paths can then early be discarded ("pruned") based on the distance bounds of the MBRs. Thus, both CPU and I/O cost can be saved, since not all database objects have to be considered. For example, the best-first search algorithm [107] exploits the structure of the R-tree.

With increasing dimensionality, however, index structures like the R-tree degrade rapidly due to the curse of dimensionality [24]. This phenomenon relativizes the term of similarity between spatial objects; distances are no more significant when the dimensionality of the vector space increases. This effect forces index structures to consider more objects and to perform a much higher number of distance computations. Thus, depending on the distribution of the data, the sequential scan often outperforms common index structures already with a dimensionality of about $d = 10$. A solution is provided by commonly applied methods enhancing the sequential scan, for example the *VA-file* [207]. These structures follow a process of multistep query processing (cf. Figure 1.5), which consists of a filter step (or several successive filter steps) and a refinement step. In the filter step, distance approximations of objects are used in order to categorize the objects. True hits already satisfy the query predicate based on their distance approximations and, thus, can be added to the result. True drops do not satisfy the query predicate based on the approximated distances and can therefore be discarded from further processing. Candidates may satisfy the query predicate based on their approximations and have to be further processed. Multiple filter steps can be performed, successively reducing the candidate set, before finally refining all retrieved candidates, which is, in general, more expensive than examining objects based on their distance approximations.

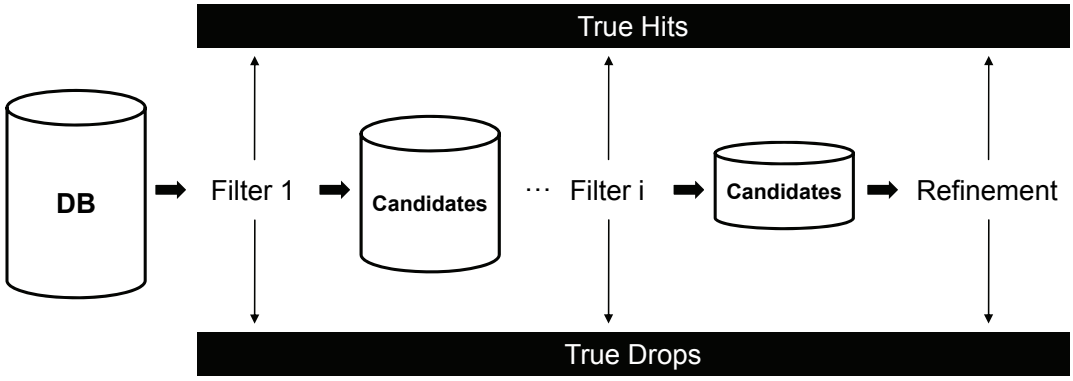


Figure 1.5: Multistep query processing.

1.2.4 From Single to Multiple Observations

Similarity relationships can clearly be determined via distance functions if the objects are created by single occurrences. However, tackling the problem of solving the above similarity queries for objects that consist of multiple occurrences, where these occurrences are subject to specific key properties, poses diverse challenges. The following section will introduce the terminology of *multi-observation data*, where objects are represented by more than one occurrence, in contrast to *single-observation data*, which denotes data obtained from a single occurrence.

1.3 A Definition of Multi-Observation Data

Many real-world application domains such as sensor-monitoring systems for object tracking, environmental research or medical diagnostics are dealing with data objects that are observed repeatedly, which creates multiple observations for one object. These observations are subject to two key properties that do not occur in the single-observation case:

- Key Property of *Temporal Variability*: Considering an object X evolving in time, multiple observations x_i $(1 \leq i \leq n)$ of X occur in a temporal *sequence*, which incorporates the key property of temporal variability. Then, a multi-observation object represents characteristics of measurements that are captured over time, such that x_i is the observation of X at time t_i.

- Key Property of *Uncertainty*: An object X may be represented by several possible states at the same time. Then, X consists of a finite *set* of observations x_j $(1 \leq j \leq m)$, where exactly one observation corresponds to the real occurrence of X. Incorporating possible states, each observation x_j is associated with a probability (or confidence) value, indicating the likelihood of being the real occurrence of X. In common data models, observations correspond to alternative occurrences, which creates an existential dependency among the observations of an object.

Incorporating these two key properties, a d-dimensional object in the context of multi-observation data, in the following called *multi-observation object*, can be defined as follows.

Definition 1.1 (Multi-Observation Object) *A d-dimensional object X is called* multi-observation object*, if at least one of the above properties is fulfilled. It consists of multiple observations $x_{i,j} \in \mathbb{R}^d$ ($1 \leq i \leq n$, $1 \leq j \leq m$) evolving in time, represented by m different states at each of n points in time.*

Definition 1.1 considers a discrete object representation with a finite number of observations. This discrete representation will be assumed in this work. The special case of an object having only one observation ($n = m = 1$) will be called *single-observation object*.

Multi-observation data as defined above is not far from the definition of *multi-instance data*. According to [142], an object in the context of multi-instance data is represented by a set of instances in a feature space. However, the essential difference is that, for the instances of a such an object, no special assumptions are made about specific properties or characteristics in contrast to multi-observation objects.

The task of similarity processing in multi-observation data poses diverse challenges. While both key properties, temporal variability and uncertainty, are coexisting in the general case, this thesis will distinguish between two different contexts for multi-observation data, each incorporating one key property of multi-observation data:

- *Part II* will focus on the key property of temporal variability while neglecting the uncertainty property. An object X is then described by n (temporally ordered) observations $x_1, \ldots, x_n$ and $m = 1$. The presence of temporal changes of an object with observations taken over time leads to the context of *time series*. A short introduction to this part will be provided in Section 1.4.

- *Part III* will deal with the key property of uncertainty while neglecting the property of temporal variability. In this case, an object X is described by m (mutually exclusive) observations $x_1, \ldots, x_m$ and $n = 1$. This part provides contributions in the context of *probabilistic (uncertain) databases* and will briefly be introduced in Section 1.5.

1.4 Temporal Variability: Time Series

1.4.1 Motivation

In a large range of application domains, the analysis of meteorological trends, of medical behavior of living organisms, or of recorded physical activity is built on temporally dependent observations. The presence of a temporal ordering of the observations of a multi-observation object incorporates the key property of temporal variability and leads to the data model of *time series*. In particular in environmental, biological or medical applications, we are faced with time series data that features the occurrence of temporal patterns composed of regularly repeating sequences of events, where cyclic activities play a

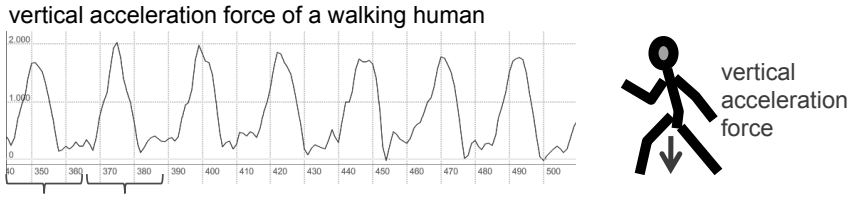


Figure 1.6: Evolution of periodic patterns in medical and biological applications [2].

key role. An example of a periodic time series is depicted in Figure 1.6, showing the motion activity of a human, in particular the vertical acceleration force that repetitively occurs during a human motion like walking or running. Though consecutive motion patterns show similar characteristics, they are not equal. It is possible to observe changes in the shape of consecutive periodic patterns that are of significant importance.

In the medical domain, physical activity becomes more and more important in the modern society. Nowadays, cardiovascular diseases cover a significant part of annually occurring affections, which is due to the reduced amount of activity in the daily life [21]. The automation of working processes as well as the availability of comfortable travel options may cause overweight [211], which may result in lifestyle diseases, such as diabetes mellitus [163]. Warburton et al. [204] showed that prevention and therapy of such diseases as well as the rehabilitation after affections or injuries can be supported by continuous and balanced physical activity. For this purpose, patients are required to fulfill a regular quota of activity which follows a particular training schedule that is integrated into the daily life, but which cannot be supervised. In order to obtain reliable evidence about the achieved physical activity within a particular time period, accelerometers can act as tools that provide accurate results, since filled questionnaires tend to be strongly subjective [12, 206]. This statement is obvious, as, according to [97], the patients tend to overestimate their own abilities, which leads to results that are likely to be biased. Furthermore, the evaluation of results is very complex and time-consuming. In order to improve the quality, i.e., the accuracy and the objectivity of these results, accelerometers serve as suitable devices for medical monitoring. The recordings of sensor observations allow the detection of any type of human motions that are composed of cyclic patterns. Cycling, for example, is a typical activity where cyclic movements repeatedly occur via pedaling; but periodic patterns can also be detected from other activities, such as walking, running, swimming and even working. In this context, the analysis of time series leads to the field of activity recognition.

1.4.2 Challenges

In the single-observation case, temporal characteristics do not occur, since an object is created by only one observation. Assuming a dimensionality of $d = 1$ for the purpose of simple illustration, distances between objects can be mapped to the simple differences of the values (cf. Figure 1.7, left depiction). In the illustrated example, $dist(A, B) < dist(A, C)$ holds.

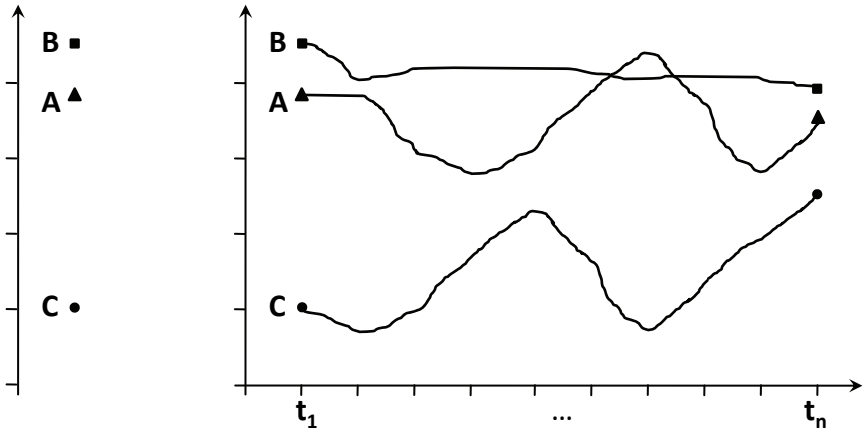


Figure 1.7: Single- and multi-observation objects w.r.t. temporal variability ($d = 1$).

In the multi-observation case, an object corresponds to a time series. In the right depiction of Figure 1.7, the objects A, B and C are extended to three one-dimensional time series of length n. In addition to the domain describing the value (the amplitude) of an observation, a temporal domain is needed, which creates the sequence of values.

While the similarity relationships can be observed clearly in the single-observation case, getting the intuition in the multi-observation case is more complicated. A visual exploration yields the impression that the amplitude values of the observations of time series A are closer to the amplitudes of time series B than to the amplitudes of C, i.e., here again, $dist(A, B) < dist(A, C)$ seems to hold if the Euclidean distance is simply translated to the multi-observation case. According to Equation (1.1), in the general case, the Euclidean distance between two d-dimensional time series X and Y of length n is computed as

$$dist = \sqrt{\sum_{i=1}^{n} \left(\sum_{j=1}^{d} (x_{i,j} - y_{i,j})^2 \right)},$$

where $x_{i,j}$ ($y_{i,j}$) denotes the value of the ith observation of X (Y) in dimension j. However, for some scenarios, this relationship may not be satisfying. B may be closer to A regarding the single amplitude values, but incorporating the temporal ordering, C may be closer to A, as it contains the same, but shifted pattern evolution as A, whereas the evolution of B shows different characteristics. Even if the amplitudes are normalized to an equal range, e.g., $[0, 1]$, we still cannot be sure whether the result corresponds to the desired semantics.

Here, the question arises where exactly to put emphasis when computing similarity among time series. Important characteristics of time series are defined by temporal patterns of observations, which show periodic occurrences in many scenarios. Regarding these periodic patterns, the general challenges are how they can be determined and how appropriate similarity measures can be applied in order to take these patterns into account. Examining the medical application scenario of activity recognition, a method of analyzing cyclic activities will be presented in Part II.

The complex data structure of potentially long time series in conjunction with the temporal ordering as well as the presence of noise and missing values due to erroneous sensor recordings and hardware limitations pose further challenges. A combination of feature extraction, a sufficiently good representation of the time series by feature vectors and the possibility to use suitable indexes for enhancing similarity queries and data mining tasks in potentially high-dimensional feature spaces is required. These requirements will also be addressed in Part II.

1.5 Uncertainty: Probabilistic Databases

1.5.1 Motivation

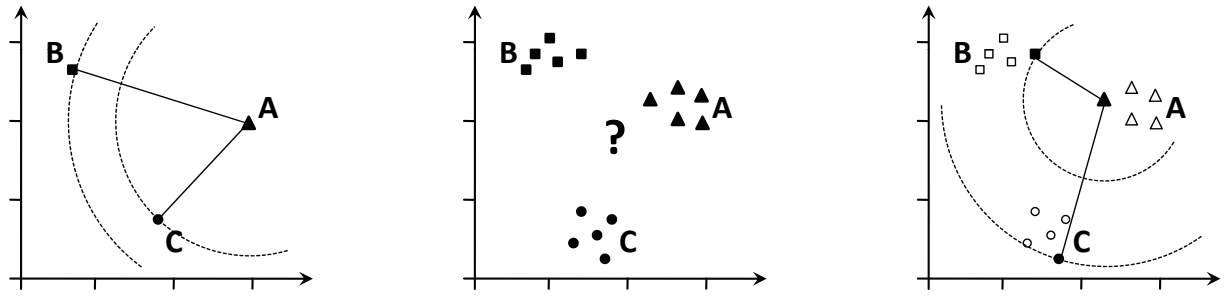
Following the key property of uncertainty, observations of an object are given as a set of occurrences of this object that are available at the same time. The question of interest in this case is the following: "Which observation is most likely to represent object X?" Depending on the data model, the existence of an observation affects the existence of the others that may represent the same object.

The potential of processing probabilistic (uncertain) data has achieved increasing interest in diverse application fields, such as traffic analysis [143] and location-based services [209]. By now, modeling, querying and mining probabilistic databases has been established as an important branch of research within the database community.

Uncertainty is inherent in many applications dealing with data collected by sensing devices. Recording data involves uncertainty by nature either caused by imprecise sensors or by discretization which is necessary to record the data. For example, vectors of values collected in sensor networks (e.g., temperature, humidity, etc.) are usually inaccurate, due to errors in the sensing devices or time delays in the transmission. In the spatial domain, positions of moving individuals concurrently tracked by multiple GPS devices are usually imprecise or inconsistent, as the locations of objects usually change continuously. Uncertainty also obviously occurs in prediction tasks, e.g., weather forecasting, stock market prediction and traffic jam prediction. Here, the consideration of alternative prediction results may 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 all possible locations of all vehicles at that time are incorporated. Furthermore, personal identification and recognition systems based on video images or scanned image data images may also have errors due to low resolution or noise. Finally, data may be rendered uncertain due to privacy-preserving issues, where uncertainty is required in order to distort exact information on objects or individuals.

1.5.2 Challenges

The challenges for similarity processing in probabilistic databases are motivated by Figure 1.8, where three objects A, B and C are depicted in a two-dimensional vector space



(a) Single-observation (certain) case.

(b) Multi-observation (uncertain) case.

(c) Possible world.

Figure 1.8: Single- and multi-observation objects w.r.t. uncertainty ($d = 2$).

($d = 2$). Here, the dimensions are assumed to be of equal range (which can be generalized to different ranges or weights for the context of relational attributes). Again assuming that the Euclidean distance is used, it can be observed from the example in Figure 1.8(a) that $dist(A, C) < dist(A, B)$ holds in the single-observation (certain) case.

In the example of the multi-observation case, each object consists of a set of $m = 5$ observations. The question is now to define an appropriate distance measure between the objects, as the relationship $dist(A, C) < dist(A, B)$ of the single-observation case may not be valid anymore (cf. Figure 1.8(b)). Measures reflecting the distance of point sets (e.g., the Sum of Minimum Distances [86] as used with multi-instance objects) are not appropriate, as they neglect the fact that each observation is associated with a confidence value, which also has to be incorporated when determining the distances between objects. Other possible solutions, e.g., the single-link distance [190] from the field of hierarchical clustering, only yield one representative (in this case a lower bound) of the distances.

Incorporating the confidences of the observations, there are two straightforward solutions of determining the distances, which, however, bear significant disadvantages. On the one hand, considering all *possible worlds* (cf. Chapter 9), i.e., computing the pairwise, probability-weighted Euclidean distances between all combinations of observations of two objects, causes exponential runtime and is therefore not applicable. In the above example, Figure 1.8(c) depicts one possible world, which also relativizes the previously observed relationship; here, the relationship $dist(A, C) > dist(A, B)$. The second solution is to represent each uncertain object by the mean vector of its observations and then simply apply the Euclidean distance to these (single-observation) objects. However, this aggregated representation causes a significant information loss w.r.t. the real distribution and the confidence of the observations within the objects, which may lead to incorrect or inaccurate results.

Part III will address the need for effective and efficient approaches for similarity processing in uncertain databases, in particular with solutions for similarity ranking queries in spatially uncertain data and with extending the used techniques for data mining tasks, such as the probabilistic variant of the prominent problem of frequent itemset mining.

Chapter 2

Outline

The body of this thesis is organized as follows:

Part II will deal with the key property of temporal variability of multi-observation data by focusing on similarity processing in time series databases. Here, similarity based on the extraction of periodic patterns from time series will play a key role. After giving a motivation for the analysis of time series and the need of acceleration techniques in Chapter 3, Chapter 4 will provide an overview of related work. Here, most important time series analysis methods as well as indexing techniques for high-dimensional feature spaces that support efficient processing will be summarized.

Chapter 5 will present the generic data mining framework *Knowing* which is designed for time series analysis. The central application scenario for this framework is the process of activity recognition. Chapter 6 [39, 41] will present an activity recognition approach for three-dimensional time series from accelerometers. The process chain of common solutions will be augmented by additional steps in order to achieve superior results to those of competitors. The experimental evaluation of the presented approach was supported by the *Knowing* framework.

An important step of the activity recognition process is the extraction of feature vectors from time series. This allows the acceleration of similarity queries, which are a potential part of the classification step within the activity recognition process, by the use of index structures for the potentially high-dimensional feature space. Chapter 7 [40] will address this problem for the full-dimensional space by providing a technique which enhances the sequential scan and which is based on a modified physical database design. Chapter 8 [32, 33] will address the case where only a subset of attributes chosen at query time is relevant. Two index-based solutions will be presented which address similarity processing for arbitrary subspaces. These solutions can in particular be applied in the context of querying and analyzing time series that are represented by feature vectors, if the user is aware of selecting appropriate subspace dimensions.

Part III will address the key property of uncertainty of multi-observation data by dealing with similarity processing in the context of probabilistic databases. The main focus here will be set on the acceleration of probabilistic similarity ranking of spatially uncertain objects. The techniques for efficient processing will then be applied to probabilistic mining

applications. Preliminary definitions of the used data models and the motivations to the problems that are to solve will first be given in Chapter 9. An overview of related work will follow in Chapter 10.

Chapter 11 [45, 49] will introduce a framework that supports iterative probabilistic similarity ranking. A ranking algorithm based on a divide-and-conquer method will be presented that exploits the full probabilistic information given by inexact object representations in a more efficient way. A second approach will apply an existing solution for relational data, which is based on a dynamic-programming technique, to spatially uncertain data. Chapter 12 [43] will introduce an incremental probabilistic ranking approach that enhances the dynamic-programming algorithm. This will reduce the computational cost of the former solutions from exponential and quadratic runtime to linear complexity.

Chapter 13 [44] will focus on the probabilistic inverse ranking query, which represents the contrary problem of the "common" probabilistic ranking. Therefore, the dynamic-programming technique proposed in Chapter 12 will be extended to uncertain stream data, i.e., to data that changes with elapsing time. The solution will provide result updates requiring constant time.

Chapter 14 [48] will propose an approach for the detection of potentially interesting objects (hot items) of an uncertain database in a probabilistic way. A hot item is defined by a sufficiently large population of similar objects in the database and is an essential step for several density-based data mining techniques. This approach will be based on a further extension of the dynamic-programming technique used in the previous chapters.

The final chapters of Part III will go beyond the definition of multi-observation data, but remain in the area of uncertainty. Chapters 15 [46] and 16 [47] will tackle the problem of probabilistic frequent itemset mining. Chapters 15 will introduce a framework which efficiently computes the frequentness of probabilistic itemsets, again extending the dynamic-programming technique used in the previous chapters of Part III. Chapter 16 will utilize a similar, but more intuitive technique. Furthermore, an approach will be presented to mine all probabilistic frequent itemsets in uncertain transaction databases without candidate generation, thus providing a solution which is more efficient in terms of computation time and memory requirements.

Finally, Part IV will conclude this thesis. The contributions of this work for current research will first be summarized in Chapter 17. The last chapter (Chapter 18) will examine possible future directions for each of the contributions included in this thesis for the context of the research areas of time series, indexing of high-dimensional feature spaces and probabilistic databases, respectively.

Part II

Key Property of *Temporal Variability*: Time Series

Chapter 3

Introduction

3.1 Preliminaries

3.1.1 A Definition of Time Series

The data model of *time series* incorporates the key property of temporal variability of multi-observation data (cf. Chapter 1). The general time series model used in this part is defined as follows, picking up a slight modification of the definition given in [16].

Definition 3.1 (Time Series) *A time series $X = (x_1, \ldots, x_n)$ is an ordered sequence of values $x_i \in \mathbb{R}$ $(1 \leq i \leq n)$ w.r.t. a temporal domain, where $t_i < t_{i+1}$ and $f(t_i) = x_i$. Hereby, $f : \mathbb{N} \to \mathbb{R}$ is a function mapping time stamps to amplitude values.*

In the existing literature, the temporal domain which comprises the time stamps, in most cases, assumed to be discrete, i.e., X contains a finite number of values; in this work, a discrete time domain will be assumed as well. Hence, in this part, the points t_i are called *time stamps*. The (amplitude) values of a time series will be referred to as *observations*.

3.1.2 Similarity-Based Time Series Analysis

When querying time series within analysis and mining tasks, most methods focus on time series retrieval w.r.t. the best whole or subsequence matching with a query time series. Popular distance measures for time series comprise, for example, the Euclidean distance or the *Dynamic Time Warping* (DTW) approach, which has first been used for speech recognition [182] and proposed for the utilization in time series similarity and data mining in [29]. These measures, however, bear significant drawbacks. The Euclidean distance does not consider the temporal dependency of observations, and, thus, does not reflect particular characteristics of time series (cf. Chapter 1). DTW addresses this shortcoming by allowing shifting and scaling in the time domain, but rapidly degenerates due to its high computational cost with a high number of observations. Commonly applied solutions are provided via dimensionality reduction methods and the extraction of significant features

that allow the usage of L_p-norms in the feature space and traditional indexing techniques like the R^*-tree [23] for the feature space.

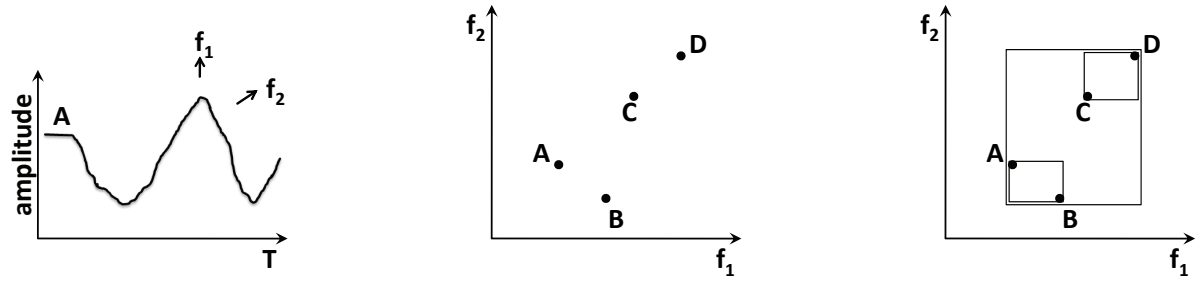
This part will focus on similarity processing on time series with a special focus on cyclic activities, in particular on the evolution of periodic patterns that repeatedly occur in specified periods over time. The motivation is given by the application scenario of activity recognition.

3.1.3 From Time Series Analysis to Activity Recognition

The field of activity recognition is an important application domain where the analysis of time series supports the detection and prevention of diseases. In this context, a dedicated processing chain is performed, including time series segmentation (which detects the periodic patterns), dimensionality reduction and the final classification. A short summary of these steps with related work will be provided in Chapter 4.

The process of activity recognition can be supported by the time series analysis framework *Knowing* [41], which will be presented in Chapter 5. *Knowing* is based on a modular structure that supports the extraction of knowledge from data, which is, in the general KDD process, not restricted to the analysis itself, but accompanied by pre- and postprocessing steps. Handling data coming directly from the source, e.g., a sensor, often requires preconditioning like parsing and removing irrelevant information before data mining algorithms can be applied to analyze the data. Standalone data mining frameworks do not provide such components since they require a specified input data format. Furthermore, they are often restricted to the available algorithms or a rapid integration of new algorithms for the purpose of quick testing is not possible. *Knowing* addresses this shortcoming and is easily extendible with additional algorithms by using an OSGi compliant architecture. In the context of activity recognition, *Knowing* serves as a medical monitoring system recording physical activity. *Knowing* was originally designed for time series analysis in the context of medical monitoring. However, the need for an extensive data mining functionality lead to a decoupling of the basic structures, resulting in a powerful data mining framework.

Chapter 6 will propose an activity recognition approach which utilizes matching-based similarity processing on time series derived from three-dimensional accelerometers. Here, the structure of patterns is strongly dependent of the characteristics of an activity. State-of-the-art activity recognition systems already provide good results, but the accuracy of recognition algorithms often depends on the position of the sensors and the quality of the data. The provided solution [39], that emerged from publications in the medical sector [197, 198], proposes an activity recognition system designed for accelerometers positioned at the ankle. In general, this position achieves superior recordings to other body positions [196]. Here, the detection of the periodic patterns is a basic task. A periodic activity appears as a time series containing similar, consecutive periodic segments; however, a pattern sequence may contain (nonperiodic) gaps due to measurement errors or intended movement breaks because of diverse influences like red traffic lights that interrupt continuous cycling or walking. An optimal classification result should include both the periodic parts and the nonperiodic segments in case the latter are surrounded by the same activity. However,



(a) Time-domain space. (b) Feature space ($d = 2$). (c) Feature space index (Example: R^*-tree, $d = 2$).

Figure 3.1: Transformation of a time series into the feature space.

the characteristics of periodic and nonperiodic segments are not necessarily the same. An adequate representation of the characteristics of the segments will be provided by the transformation to the feature space. This implies the possibility to apply common spatial indexing techniques for efficient processing. An extensive evaluation on real-world datasets will show that the provided solution outperforms prior work, while focusing on the effectiveness of activity recognition.

3.1.4 Accelerating the Process via Indexing

The activity recognition process of Chapter 6 is performed based on the extraction of relevant characteristics from time series. A time series of length n is then represented by a single point in the d-dimensional feature space, which reduces the complexity of time series, as commonly $d \ll n$ holds. In the context of similarity processing, this allows the application of spatial index structures, which accelerate similarity queries and, therewith, data mining algorithms that further process the results of similarity queries (such as similarity-based classification), resulting in a significant efficiency gain.

An example for feature transformation is illustrated in Figure 3.1. Here, two characteristic features f_1 and f_2 are extracted from a time series A (cf. Figure 3.1(a)), which then represent A as two-dimensional points in the feature space, where distances between objects are commonly determined by L_p-norms (cf. Figure 3.1(b)). Efficient query processing is then performed using a spatial index – in this example by an R^*-tree (cf. Figure 3.1(c)).

It will be shown in Chapter 6 that the derived feature vectors from the time series tend to be high-dimensional. To address possibilities to boost query processing in high-dimensional feature spaces, this part will present two indexing variants for both the full-dimensional case and for arbitrary subspaces. Both solutions will focus on k-nearest neighbor (k-NN) queries, as these can directly be used in activity classification tasks.

3.2 Indexing in High-Dimensional Feature Spaces

3.3 Full-Dimensional Indexing

Similarity processing in high-dimensional data is inherently difficult, due to the curse of dimensionality [24]. This phenomenon relativizes the term of similarity between spatial objects; distances are no more significant when the dimensionality of the feature space increases. Then, for example, nearest neighbor search is no more meaningful if the nearest neighbor of an arbitrary query object is not sufficiently different from its farthest neighbor [51].

Common index structures for feature spaces degenerate due to this well-known problem; it has been stated that, depending on the data distribution, the sequential scan performs superior to index structures. Addressing the drawbacks of traditional index structures in high-dimensional spaces, Chapter 7 [40] will elaborate on the vertical decomposition technique employed in [85], which provides a method for enhancing similarity processing high-dimensional data based on the sequential scan. While the abundance of data storage and retrieval systems is based upon horizontally decomposed data, vertical decompositions exhibit intriguing advantages, but also contain serious restrictions. Some of these restrictions will be overcome in Chapter 7.

3.4 Indexing Approaches for Subspace Queries

There are many scenarios for applications where the similarity of objects is defined for a subset of attributes. Moreover, users should be able to define an interesting subspace for each query independently. While much research has been done in efficient support of similarity queries regarding the full-dimensional space or single dimensions only, scarcely any support of similarity search in subspaces has been provided so far, e.g., [136, 156]. These approaches, however, are variations of the sequential scan and, thus, lacking conditions for efficient processing. Overcoming these drawbacks, two index-based solutions introduced in [32, 33] will be presented in Chapter 8. They facilitate efficient similarity processing for user-defined, arbitrary subspaces in large and potentially high-dimensional databases, if the user is aware of a meaningful feature combination.

Regarding the relevance of features in subspace search for activity recognition (cf. Chapter 6), the user may examine arbitrary feature combinations in order to classify new activities. In other scenarios like image retrieval, it could be of interest for any user to search, e.g., in a database of images represented by texture-, color-, and shape-descriptions, for objects that are similar to a particular image where the similarity is related to the shape of the motifs only but not to their color or even the color of the background. Also, for different queries, different regions of interest in a picture may be relevant. Furthermore, An online-store might like to propose similar objects to a customer where similarity can be based on different subsets of features. While in such scenarios, meaningful subspaces can be suggested beforehand [105, 130], in other scenarios, possibly any subspace could

be interesting. For example, for different queries, different regions of interest in a picture may be relevant. Since there are 2^d possible subspaces of a d-dimensional data set, it is practically impossible to provide data structures for each of these possible subspaces in order to facilitate efficient similarity search. Another application where efficient support of subspace similarity queries is required are subspace clustering algorithms [137] that rely on searching for clusters in a potentially large number of subspaces. If efficient support of subspace range queries or subspace nearest neighbor queries were available, virtually all subspace cluster approaches could be accelerated considerably.

Chapter 4

Related Work

4.1 Similarity of Time Series

4.1.1 Similarity Measures

Matching-based analysis comprises methods that, given a query time series, return the time series from the database that yield the best matching(s) to the query. The two main foci here are full matching w.r.t. the complete time series and partial matching w.r.t. subsequences. Overall, there are abundant approaches performing matching-based analysis of time series. Typical measures are the L_p-norms, where the Euclidean distance is most popular. However, its ability to reflect the temporal ordering of observations is poor (cf. Chapter 1). Searching patterns can be supported by the *edit distance* measures, comprising *Dynamic Time Warping* (*DTW*) [29], that supports asynchronous matching, and other variants of the edit distance, such as the *Longest Common Subsequence* (*LCSS*) [200], the *Edit Distance on Real sequence* (*EDR*) [71] and the *Edit distance with Real Penalty* (*ERP*) [70]. Since the edit distance measures support only scaling and shifting in the time domain, the works [72, 73] introduce the *Spatial Assembling Distance* (*SpADe*) model, which additionally supports scaling and shifting in the amplitude domain and, thus, is also applicable for pattern detection in streaming time series.

Matching-based approaches based on warping techniques often suffer from their unsatisfying time complexity. Thus, a number of dimensionality reduction techniques are commonly applied on time series. Well-known examples among them are the *Discrete Wavelet Transform* (*DWT*) [66], the *Discrete Fourier Transform* (*DFT*) [9], the *Piecewise Aggregate Approximation* (*PAA*) [120, 213], the *Singular Value Decomposition* (*SVD*) [128], the *Adaptive Piecewise Constant Approximation* (*APCA*) [121], *Chebyshev Polynomials* [64], the *Piecewise Linear Representation* (*PLR*) [122], the *Piecewise Linear Approximation* (*PLA*) [167], or the *Symbolic Aggregate Approximation* (*SAX*) [159, 160]. In [93], the authors propose the *GEMINI* framework, that allows to incorporate any dimensionality reduction method into efficient indexing, as long as the distance function on the reduced feature space satisfies the lower bounding property. Extracted features of different types and expressiveness are combined to feature vectors.

Existing work that utilizes dimensionality reduction methods in the context of activity recognition will be reviewed in Subsection 4.1.2.

In some application scenarios, the exact observations are less important than the fact whether a particular amplitude threshold is exceeded, such that an observation can be regarded to represent a significant event. This leads from *matching-based* to *threshold-based* time series analysis. Threshold-based analysis on time series is performed by the detection of similar events or regions of significance which exceed a particular amplitude level, and finally by the consideration of a representation or a similarity measure that focuses on these events.

The authors of [176] propose a bit sequence representation of time series. For each observation, a bit is set if the corresponding amplitude value exceeds a particular threshold value. Thus, sequence patterns are defined on the threshold-exceeding amplitudes. Similarity can then efficiently be computed based on those bits. However, this solution does not provide a possibility to specify a particular threshold value at query time.

This restriction has been addressed in [2, 3], which support similarity search methods based on the consideration of significant events that can be recognized with amplitude values that exceeding a particular threshold. Given a threshold value τ, this approach reduces time series to a sequence of intervals corresponding to time periods where the amplitude value of a time series exceeds τ. Based on this threshold representation, the features proposed in [4] can be calculated over the whole amplitude spectrum for different values of τ. Thus, time-domain properties can be captured over the whole available amplitude range. The authors of [2] introduce the *dual-domain* time series representation, where the existence of periodic patterns is captured from multiple time domains. Threshold-based techniques allow to materialize these patterns as spatial objects. There, it is shown that the extraction of simple features can achieve a good quality of similarity query results. An implementation is provided in [42].

4.1.2 Applicative Time Series Analysis: Activity Recognition

The general steps of the activity recognition process is related to the general KDD process (cf. Chapter 1). Contributions as well as applicative publications that use particular techniques will be summarized in the following part. A more detailed survey of this processing chain is given in [17].

Data Preprocessing

Recorded time series data from accelerometers often contains noise of high frequency, which in many cases distorts the actual signal. Thus, sliding-window-based average [127] or median filters [118] are applied in order to remove outliers. Furthermore, removing the effect of the gravitational force is supposed to distinguish activity from non-activity phases. This is in general obtained by applying a low-pass filter, as shown in [13, 118].

Segmentation

In order to separate periodic parts from nonperiodic parts, time series are divided into subsequent segments. In the literature, there exist different techniques for segmenting time series. Sliding-window-based methods [131, 174, 201, 202] are suitable for online processing and provide pattern matching algorithms starting with a reference sample that is extended until the matching distance exceeds a particular threshold. Top-down approaches in the context of time series processing [148, 153, 188] recursively split a time series into two subsequences w.r.t. an approximation error threshold. Complementary approaches [123, 124, 125] work in a bottom-up manner, starting with $\frac{n}{2}$ segments of size 2 (where n denotes the number of observations) and combining adjacent subsequences until an upper cost bound is reached.

In [122], time series segmentation is used to obtain a piecewise linear representation of time series, such as *PLA* [167] *PLR* [122]. The authors propose the *SWAB* framework, which combines the advantages of sliding-window, which is most efficient, and bottom-up, which provides best segmentation quality. Nevertheless, sliding-window-based methods are still most frequently used in the area of activity recognition [17], as it is suitable for online processing. Selected coefficients obtained by dimensionality reduction methods and further characteristics are then obtained by feature extraction. Thus, the segmentation method used in Chapter 6 is based on a sliding-window algorithm. The obtained segments will turn out to provide a good separation of the activity recordings.

Feature Extraction

Periodic and nonperiodic segments of time series are commonly described by a combination of features of different types.

- Time-domain features, such as mean, variance and standard deviation [146, 171, 205] or the *Root Mean Square* (*RMS*) [100, 164] are directly derived from the time series. Further prominent examples of this feature type are the average time between the peaks [146] and the number and average value of the peaks [205].

- Many publications apply well-known dimensionality reduction techniques by transforming the time series into the frequency domain (see also Subsection 4.1.1). Frequency-domain features can be derived by the DFT (or *FFT, Fast Fourier Transform [62]*) and are used in [20, 193]. Features like aggregated FFT coefficients or the entropy of the frequency domain [20], that distinguish activities where similar energy values are detected (e.g., running and cycling), or single FFT coefficients [133] are also used in existing literature.

- A combination of domains w.r.t. time and frequency is given by wavelet features, derived from the DWT and is used in the context of gait classification [169].

- Heuristic features cannot be directly derived from the time series, but require mathematical and statistical methods to be extracted from the three dimensions of ac-

celerometrical time series data simultaneously. A prominent example here is the *Signal Magnitude Area* (*SMA*), which is defined by the sum of the absolute values of all axes within the current time window and that is used in several works [13, 58, 118, 127, 212]. A further feature of this class is given by the *Inter-Axis Correlation*, which is a suitable measure to distinguish between movements measured at different body parts [20]. However, the authors of [171] could prove that this feature performs inferior to the simple features like mean and standard deviation. Further heuristic features will be presented in Chapter 6.

The adequate combination of features is an important task, since the classification of the time series highly depends on a good representation.

Feature Vector Dimensionality Reduction

In order to reduce the computational effort of the classification process, dimensionality reduction is typically applied in order to remove redundant information; this decreases the size of the feature vectors. In the context of accelerometer data, the literature distinguishes between methods of *feature selection* and *feature transformation*, which can also be used in combination.

- Feature selection methods include, for example, methods based on *Support Vector Machines* (*SVMs* [132], e.g., applied in [203]), or the *forward-backward search* technique [218] (e.g., used in [171] and also in Chapter 6).

- Feature transformation methods further support the separation of different classes. Commonly applied techniques here are the *Principal Component Analysis* (*PCA* [170], e.g., used in [212, 216]), the *Independent Component Analysis* (*ICA* [81], e.g., used in [166]) or the *Linear Discriminant Analysis* (*LDA* [95], e.g., used in [100, 127] and also in Chapter 6).

Classification

The effectiveness and also the efficiency varies with the selection of the classifier. Using similarity-based classifiers, similarity queries performed within the classification process can be further accelerated via indexing techniques. The latter issue will be addressed in Chapters 7 and 8 for the case of k-nearest neighbor (k-NN) queries, which are performed in the context of k-NN classification.

Several publications in the context of activity recognition apply supervised classification methods based on pattern recognition and training phases, e.g., decision trees [20, 106, 118], *Hidden Markov Models* [205], *Gaussian Mixture Models* [13], k-NN classifiers [106, 171], *Naïve Bayes* classifiers [106], *Support Vector Machines* (*SVMs*) [132], or *Neural Networks* [127, 146]. Chapter 6 will propose an additional step that improves the classification result.

4.2 Indexing in High-Dimensional Feature Spaces

4.2.1 Full-Dimensional Indexing

The contributions of research on full-dimensional index structures are abundant [183]. Established index structures, such as [23, 28, 101, 119], are designed and optimized for the complete data space where all attributes are relevant for data partitioning and clustering or for simply satisfying a query predicate. With increasing dimensionality, however, index structures degrade rapidly due to the curse of dimensionality [24].

A solution is provided by commonly applied methods enhancing the sequential scan, for example the *VA-file* [207]. Other approaches use a hybrid structure, which is tree-based, but requires to scan successive blocks or node elements [27, 28].

A third solution tackling the problem of indexing high-dimensional data called *BOND* is given in [85], which is also a search strategy enhancing the sequential scan. Contrary to the aforementioned techniques, *BOND* exploits modifications w.r.t. the physical database design. The basic idea is to use a columnstore architecture (as known from NoSQL database systems), sort the columns according to their potential impact on distances and prune columns if their impact becomes too small to change the query result. However, *BOND* depends on particular assumptions that restrict the applicability of the approach. Chapter 7 [40] will introduce a solution that overcomes most of these restrictions.

4.2.2 Indexing Approaches for Subspace Queries

The first approach addressing the problem of subspace similarity search explicitly has been proposed by the *Partial VA-file* in [136]. There, the authors propose an adaptation of the VA-file [207] to the problem of subspace similarity search. The basic idea of this approach is to split the original VA-file into one partial VA-file for each dimension, containing the approximation of the original full-dimensional VA-file in that dimension. Based on the information of the partial VA-files, upper and lower bounds of the true distance between data objects and the query are derived. Subspace similarity queries are processed by scanning only the relevant files in the order of relevance, i.e., the files are ranked by the selectivity of the query in the corresponding dimension. This processing is similar to [85], which will be reviewed in the full-dimensional case in Chapter 7, and which implicitly addresses the subspace problem by its physical design via weighted search. A third approach to the problem is proposed in [156], although only ε-similarity range queries are supported. The idea of this multipivot-based method is to derive lower and upper bounds for distances based on the average minimum and maximum impact of a possible range of the subspace dimensions; these bounds are computed in a preprocessing step for a couple of pivot points.

All these approaches are variations of the sequential scan. Contrary, Chapter 8 will present two index-based solutions that accelerate similarity search in arbitrary subspaces.

Chapter 5

Knowing: A Generic Time Series Analysis Framework

5.1 Motivation

Supporting the data mining process by tools was and still is a very important step in the history of data mining. With the support of several tools like *ELKI* [1], *MOA* [56], *WEKA* [102], *RapidMiner* [165] or *R* [175], scientists are nowadays able to apply a diversity of well-known and established algorithms on their data for quick comparison and evaluation. Although all frameworks perform data mining in their core, they all have different target groups.

WEKA and *MOA* provide both algorithms and GUIs. By using these GUIs, the user can analyze datasets, configure and test algorithms and visualize the outcome of the according algorithm for evaluation purposes without needing to do some programming. As the GUI cannot satisfy all complex scenarios, the user still has the possibility to use the according APIs to build more complex scenarios in his or her own code.

RapidMiner integrates *WEKA* and provides powerful analysis functionalities for analysis and reporting which are not covered by the *WEKA* GUI itself. *RapidMiner* provides an improved GUI and also defines an API for user extensions. Both *RapidMiner* and *WEKA* provide some support to external databases.

The aim of *ELKI* is to provide an extensible framework for different algorithms in the fields of clustering, outlier detection and indexing with the main focus on the comparability of algorithm performance. Therefore, single algorithms are not extensively tuned to performance, but tuning is done on the application level for all algorithms and index structures. Like the other frameworks, *ELKI* also provides a GUI, so that programming is not needed for the most basic tasks. *ELKI* also provides an API that supports the integration of user-specified algorithms and index structures.

All the above frameworks provide support for the process of quick testing, evaluating and reporting and define APIs in different depths. Thus, scientists can incorporate new algorithms into the systems. *R* provides a rich toolbox for data analysis. Also, there are

many plug-ins which extend the functionality of *R*.

In cases where the requirements enforce a rapid development from data mining to a representative prototype, these unstandardized plug-in systems can cause a significant delay which is caused by the time needed to incorporate the algorithms. Each implementation of an algorithm is specifically adapted to the according framework without being interchangeable.

With the use of a standardized plug-in system like OSGi[1], Java Plug-in Framework (JPF) or Java Simple Plug-in Framework (JSPF), each implementation of an algorithm does not have to be specifically adapted to the according framework. This chapter will introduce *Knowing* (*Know*ledge Engineer*ing*) [41], a framework that addresses this shortcoming by bridging the gap between the data mining process and rapid prototype development. This is achieved by using a standardized plug-in system based on OSGi, so that algorithms can be packed in OSGi resource bundles. This offers the possibility to either create new algorithms as well as to integrate and exchange existing algorithms from common data mining frameworks. The advantage of these OSGi compliant bundles is that they are not restricted for use in *Knowing*, but can be used in any OSGi compliant architecture.

The data mining tool *Knowing* includes the following contributions:

- a simple, yet powerful graphical user interface (GUI),
- a bundled embedded database as data storage,
- an extensible data mining functionality,
- extension support for algorithms addressing different use cases, and
- a generic visualization of the results of the data mining process.

Details of the architecture of *Knowing* will be given in Section 5.2. The application scenario, described in Section 5.3, presents the medical monitoring system *MedMon* [186], which itself extends *Knowing*. In the developer stage, it is easily possible to switch between the scientific data mining view and the views which will be presented to the end users later on. As *MedMon* is intended to be used by different target groups of the medical area (physicians and patients), it is desired to use a single base system for all views and only deploy different user interface bundles for each target group. This way, the data mining process can seamlessly be integrated into the development process by reducing long-term maintenance to a minimum, as only a single system with different interface bundles has to be kept up to date and synchronized instead of a special data mining tool, a physician tool and a patient tool.

[1] OSGi Alliance: `http://www.osgi.org/`

5.2 Architecture

5.2.1 Modularity

Applying a standardized plug-in system like OSGi, the bundles can be used in any OSGi compliant architecture like the Eclipse Rich Client Platform (RCP)[2] or the NetBeans RCP[3]. Then, the integration of existing algorithms can simply be done by wrapping and packing them into a separate bundle. Such bundles are then registered as independent service providers to the framework. In either case, algorithms are wrapped into Data Processing Units (DPU) which can be integrated and configured via pluggable RCP-based GUI controls. Thus, the user is able to perform an arbitrary amount of steps to pre- and postprocess the data. Furthermore, the possibility is provided to use the DPUs contained in the system in any other OSGi compliant architecture. As dependencies between resource bundles have to be modeled explicitly, it is much easier to extract particular bundles from the system. This loose coupling is not only an advantage in case where algorithms should be ported between completely different systems, but also if the GUI should be changed from a data mining view to a prototype view for the productive system. This can be done by either using the resource bundles containing the DPUs, or by directly extending *Knowing* itself.

In the current implementation, the *Knowing* framework is based on the established and well-known Eclipse RCP system and uses the standardized OSGi architecture[4], which allows the composition of different bundles. This brings the big advantage that data miners and developers can take two different ways towards their individual goal: if they start a brand new RCP-based application, they can use *Knowing* out of the box and create the application directly on top of *Knowing*. The more common case might be that an RCP- or OSGi-based application already exists and should only be extended with data mining functionality. In this case, only the appropriate bundles are taken from *Knowing* and integrated into the application.

The following part describes the architecture of the *Knowing* framework, which consists of a classical three-tier architecture comprising data storage tier, data mining tier and GUI tier, where each tier can be integrated or exchanged using a modular concept.

5.2.2 Data Storage

The data storage tier of *Knowing* provides the functionality and abstraction layers to access, import, convert and persist the source data. The data import is accomplished by an import wizard using service providers, so that importing data is not restricted to a particular format.

Applying the example of the *MedMon* application, a service provider is registered that reads binary data from a three-dimensional accelerometer [198] which is connected via

[2]Eclipse RCP: `http://www.eclipse.org/platform/`
[3]NetBeans RCP: `http://netbeans.org/features/platform/`
[4]Eclipse Equinox: `http://www.eclipse.org/equinox/`

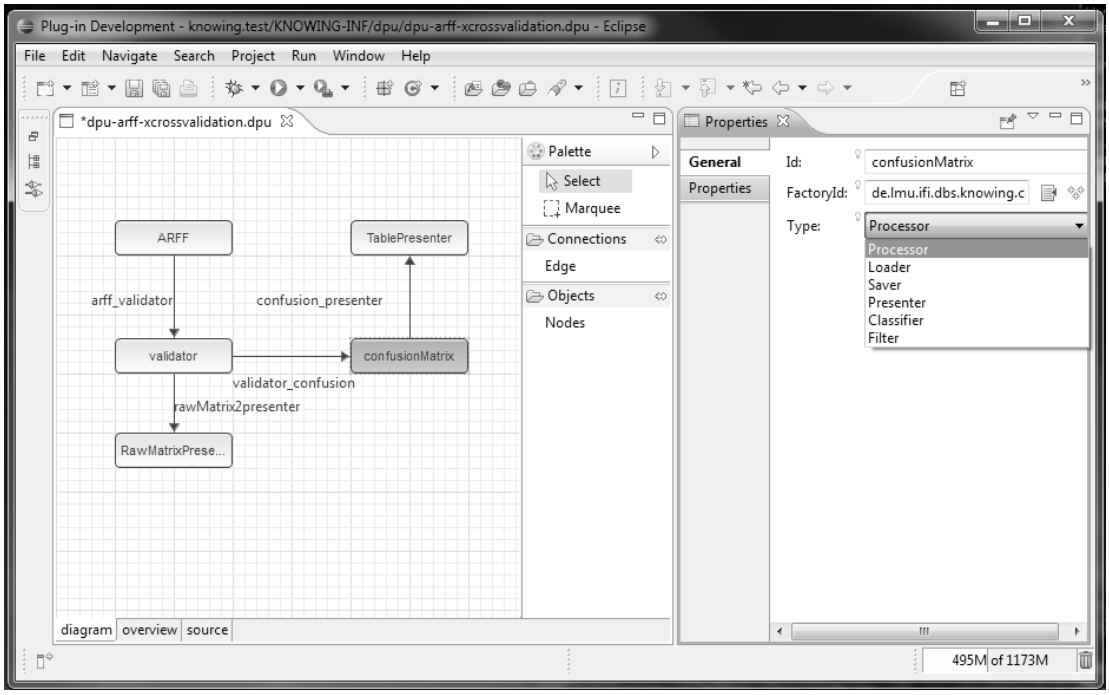


Figure 5.1: The process chain editor of *Knowing* user interface.

USB. The data storage currently defaults to an embedded Apache Derby database[5] which is accessed by the standardized Java Persistence API (JPA & EclipseLink). This has the advantage that the amount of data being read is not limited by the main memory of the used workstation and that the user does not have to set up a separate database server on his or her own. However, by using the JPA, there is the possibility to use more than 20 elaborated and well-known database systems which are supported by this API[6]. An important feature in the data storage tier arises from the possibility to use existing data to support the evaluation of newly recorded data, e.g., to apply particular parts of the data as training sets or reference results.

5.2.3 Data Mining

This tier includes all components needed for data mining and data analysis. OSGi bundles containing implemented algorithms are available fully transparently to the system after the bundle is registered as a service provider.

Algorithms are either implemented directly or wrapped in DPUs. Following the design of *WEKA*, DPUs represent filters, algorithms or classifiers. One or more DPUs can be bundled into an OSGi resource bundle which is registered into the program and, thus, made available in the framework. Bundling algorithms enforces a pluggable and modular

[5]Apache Derby: `http://db.apache.org/derby/`
[6]List of supported databases: `http://wiki.eclipse.org/EclipseLink/FAQ/JPA`

architecture, so that new algorithms can be integrated and removed quickly without the need for extensive dependency checks. The separation into bundles also provides the possibility of visibility borders between bundles, so that separate bundles remain independent and, thus, the system remains maintainable. The modularity also provides the possibility to concatenate different algorithms into processing chains so that algorithms can act both as sources and targets of processed entities (cf. Figure 5.1). Raw data for example first could pass one or more filtering components before being processed by a classification or clustering component.

The creation of a processing chain (a.k.a. model) of different, concatenated algorithms and data-conditioning filters is supported by GUI controls, so that different parameters or concatenations can be tested easily. After a model has proved to fit the needs of a use case, the model can be bundled and later be bounded to other views of the GUI, so that the cost for porting, adapting and integration is minimized to binding components and models together. Hence, porting and adapting algorithms and other components from different APIs is not needed.

This architecture provides the possibility to integrate algorithms from other sources like [1, 56, 102, 175], so that existing knowledge can be reused without having to reimplement algorithms from scratch. This also provides the possibility to quickly replace components by different implementations if performance or licensing issues require to do so.

In the data mining part of the application, *Knowing* does not only support plain Java but also relies on the use of the Scala programming language[7]. Scala is a functional and object-oriented programming language which is based on the Java Virtual Machine, so that it seamlessly integrates into *Knowing*. The advantage of Scala in this part of the application lies in the simple possibility of writing functional code shorter than in regular Java code. By using the Akka actor model[8], it is easy to create processing chains which are executed in a parallel way so that *Knowing* can make use of multi-core systems.

5.2.4 User Interface

Using the well-established Eclipse RCP and its powerful concept of views enables developers to easily replace the view of the data mining scientists with different views for end users or prototypes. Thus, the task of porting data mining algorithms and the data model to the final application is replaced by just combining the binding model with the view components. As Eclipse itself is designed as an RCP using OSGi, it is comparatively easy to unregister the original *Knowing* GUI and replace it with an interface representing the final application.

[7]Scala programming language: `http://www.scala-lang.org/`
[8]Project Akka: `http://akka.io/`

5.3 Application Scenario

The application is motivated by an early stage of the application prototype *MedMon* (*Med*ical *Mon*itoring), which is based on *Knowing*. *MedMon* is a prototype of a use case for monitoring a patient's activity to support his or her convalescence and is presented in [186].

Physical activity in this case includes various types of motion like walking, running and cycling. The task is to perform data mining on long-term temporal sensor data provided by people wearing the accelerometer, which is recording and storing acceleration data in all three axes with a frequency of 25 Hz. When the sensor is connected to a computer, the data is parsed and transferred to the *Knowing* framework, where it is stored in the underlying database. *Knowing* is able to deal with different types of time series which are not limited to the medical field but can be applied to different types of scenarios where time series data is being produced and needs to be analyzed. Analyzing the data in this use case means the application of classification techniques in order to detect motion patterns of activities and, thus, to separate the different types of motions. Available algorithms as well as additionally implemented techniques for data mining and the preconditioning of the temporal data (e.g., filtering of specific information, dimensionality reduction or removing noise) can efficiently be tested and evaluated on the data and can furthermore be applied to the data by taking advantage of the OSGi compliant architecture (cf. Section 5.2). By using the standardized OSGi plug-in system, *Knowing* integrates and embeds well-known data mining tools and, thus, avoids the reimplementation of already tested algorithms. The requirement of a quick migration of the final data mining process chain to a prototype system is accomplished by using different graphical views on a common platform. Thus, neither the process models nor the algorithms need to be ported. Instead, only a different view of the same base model needs to be activated to enable the prototype. Finally, the demo provides a generic visualization model to present the results of the data mining process to the user. An exemplary GUI frame is depicted in Figure 5.2. The import of the raw data is simplified by a wizard, which includes a preview of the time series.

Working with *MedMon*, the user is enabled to switch between different roles. The prototype allows several views on the recorded data and the results of the data mining process:

- the *data mining view*, where DPUs can be combined to processing chains and which allows to employ newly developed algorithms;
- the *physician view*, which provides a more detailed view on the data for multiple users' activities and the possibility to add and modify electronic health records;
- and the *patient view*, which displays only a very brief summarization in order to give feedback to the user about his or her achieved activity pensum each day.

In the presented use case, it is possible to analyze the daily activity processes and perform long-term data analysis, as performed by the activity recognition solution in Chapter 6. In

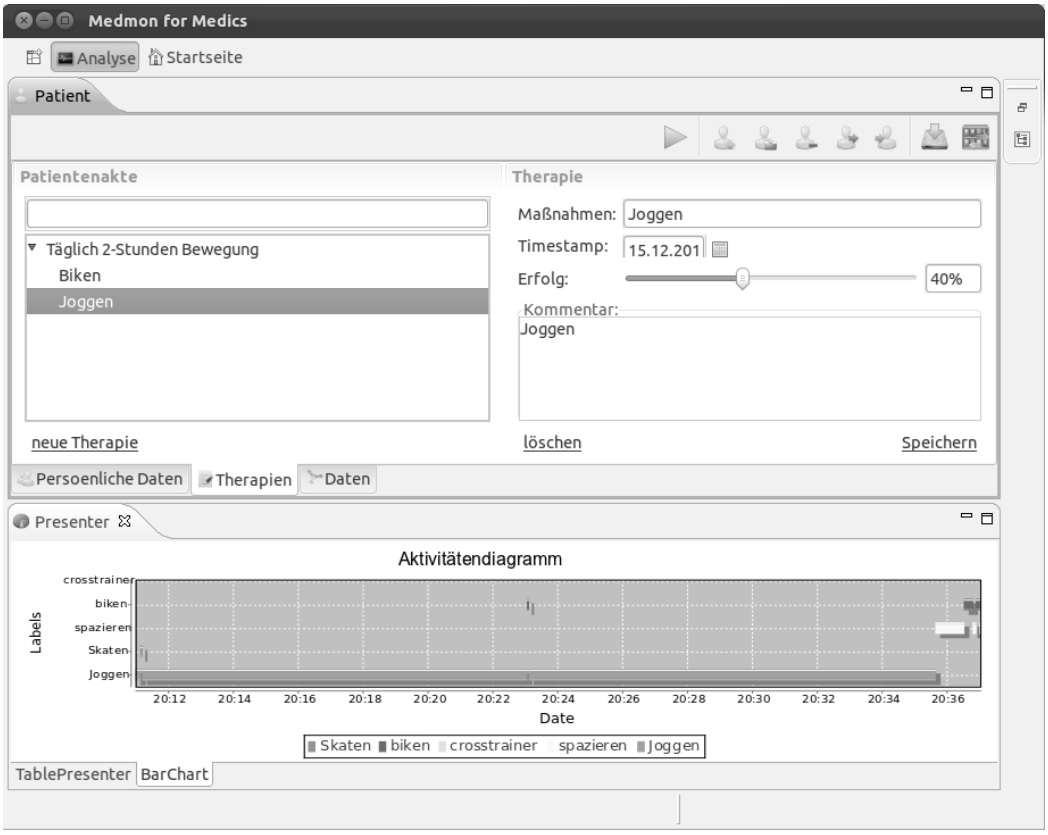


Figure 5.2: The *MedMon* prototype GUI.

MedMon, this analysis uses an aggregated view of the results of the data mining process from the physician view and the patient view. In particular, the data mining view offers possibilities of the integration of newly developed algorithms and methods for data mining on the recorded database. More precisely, the current process comprises the import of sensor data from binary files, followed by the steps performed with the approach of Chapter 6: data preprocessing, the segmentation of the data, the derivation of features from the extracted segments, dimensionality reduction techniques and the classification of the results (which can additionally be augmented by postprocessing steps, such as the reclassification). Here, the user is able to decide whether to add noise filters for the raw data, select appropriate features to represent the segments or to choose from different classification methods.

The *MedMon* prototype system is not limited to medical applications, but provides a valuable tool for scientists having to deal with large amounts of time series data. The source code of the *Knowing* framework, the *MedMon* prototype in its current state and the project wiki are available via GitHub[9].

[9] *Knowing* on GitHub: `https://github.com/knowing/`

5.4 Summary

This chapter presented the data mining framework *Knowing* which allows faster integration of data mining techniques into the development process of scientific processing methods, so that information and data can be managed more effectively. The application scenario showed the integration of *Knowing* in the application of medical monitoring and outline the bridge between data mining and development.

Chapter 6

Activity Recognition on Periodic Time Series

6.1 Introduction

Many applications that take advantage of the ability to examine the evolution of periodic patterns in time series can be found in the biological, environmental or medical domain. In the latter, an important application where the analysis of time series supports the detection and prevention of diseases is the field of activity recognition. In order to analyze activity, there is a need of appropriate techniques that detect different activity patterns and to assign them to predefined activity labels.

This chapter will present an algorithm for activity recognition, where time series data derived from three-dimensional accelerometers is analyzed in order to classify the recorded patterns w.r.t. different types of activities. Here, the three dimensions reflect the direction of physical movement in each axis, i.e., forward/backward, left/right and up/down. The application scenario was published in a medical context [197, 198] within a collaboration with the *Sendsor GmbH*[1], who also provided the accelerometers used to record the datasets for the experimental evaluation of this chapter. These sensors measure acceleration amplitudes of $\pm 2g$.

Diverse pieces of work have shown that the position of an accelerometer has a significant influence on the results obtained while measuring physical activity [20, 173, 196]. The results of this research leads to varying interpretations, as the set of activities that have to be classified strongly depends on the problem definition. There is still no dedicated position where the measurements of an accelerometer provide globally best results that are independent of the set of activities; however, it has been shown that accelerometers positioned at the ankle achieve superior recordings to other body positions [196].

Activity recognition requires several steps of preprocessing before the classifier can separate different activities properly (cf. Chapter 4). Avci et al. [17] provide a detailed survey of these steps. An overview of the general processing chain is shown in Figure 6.1.

[1] `http://www.sendsor.de`

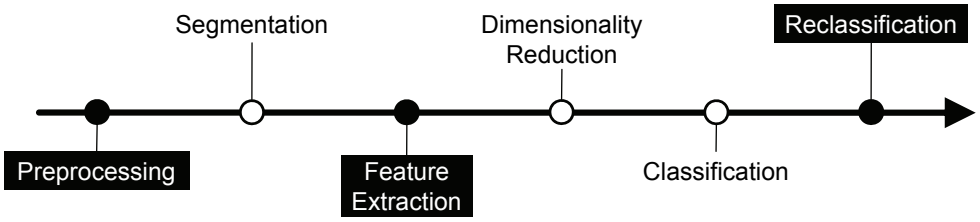


Figure 6.1: A visualization of the activity recognition process.

The main contributions of this chapter, highlighted in Figure 6.1, consist of

- a reconstruction of the data peaks within the preprocessing steps for the case that the measured acceleration values exceed the amplitude range,
- the introduction of additional features to represent the recorded physical activities,
- and a reclassification step that corrects classification errors.

The rest of this chapter is organized as follows. The preprocessing steps performed on the time series will be summarized in Section 6.2. Section 6.3 will present the segmentation algorithm. Section 6.4 will give details about the examined features for the classification. Applied techniques for dimensionality reduction of feature vectors will be presented in Section 6.5. A postprocessing step that corrects classification errors will be presented in Section 6.6. Section 6.7 will provide a broad experimental part containing the evaluation of the process chain against an existing approach. Finally, Section 6.8 will conclude this chapter.

The experimental evaluation was performed using the *Knowing* framework [41] (cf. Chapter 5), which was developed in conjunction with the approach that will be proposed in this chapter.

Table 6.1 provides an overview of the most frequent notations used in this chapter. The processing chain is started with data preprocessing in the following section.

6.2 Preprocessing Steps

6.2.1 Outlier Removal

In order to remove outliers in the data that emerged from measurement or transmission errors, an average filter is applied [127]. Thus, further processing techniques that are applied to the data are not influenced by noise. This is, in particular, important for time series segmentation (cf. Section 6.3) and feature extraction, e.g., the computation of the *Average Peak Amplitude* feature (cf. Section 6.4).

Notation	Description
X	a one-dimensional time series
n	the length (number of observations) of X
t_i	the ith time stamp of X
x_i	the observation of x occurring at time t_i
X_{t_i}	the subsequence of X starting at time t_i
$\widehat{X}$	a reference sample of X
$\widehat{n}$	the length of $\widehat{X}$
Δ_{opt}	the optimal pattern length
Δ_{max}	the maximum pattern length
ρ_{opt}	the correlation obtained with Δ_{opt}
τ_ρ	the correlation threshold
S	a segment (subsequence) of X
n'	the length of S
S_{max}	the maximum amplitude value of S
τ_{min}	the required minimum amplitude for peaks
v	a feature vector
d_v	the dimensionality of v

Table 6.1: Table of notations frequently used in this chapter.

6.2.2 Peak Reconstruction

In some cases, the sensor recordings may be incomplete. The approach that will be presented in this chapter uses a sensor that measures acceleration amplitudes in the range of $\pm 2g$. However, very intense or fast movements with a higher acceleration value create amplitudes that exceed this range. For consecutive values that are beyond this range, these intervals are cut, yielding significant gaps. One solution to overcome this problem is, of course, to use a sensor that supports measurements of a higher amplitude range up to $\pm 12g$ [58]. However, in order to be independent of technical constraints, the following method provides a reconstruction of the original signal.

The first step is to identify time intervals where the measured acceleration has exceeded the amplitude range. In these parts, at least two observations must exist with a maximum (minimum) amplitude of exactly $+2g$ ($-2g$). As this scenario is an improbable case, such a sequence is likely to be the result of truncated data. The missing data after truncations can be reconstructed using the preceding and following observations. Based on these values, the average gradients before and after a peak (Δ_{before} and Δ_{after}) are derived (cf. Equations (6.1) and (6.2)) and the average total gradient Δ_{total} can be computed (cf. Equation (6.3)).

$$\Delta_{before} = \frac{1}{o} \sum_{i=a-o}^{a} x_{i+1} - x_i \tag{6.1}$$

Algorithm 1 Peak Reconstruction: reconstructPeaks(X, o, *maxAmp*)

Require: X, o, *maxAmp*

1: $n \leftarrow |X|$ $i \leftarrow 1$, $j \leftarrow i + 1$
2: **while** $j \leq n$ **do**
3: **while** $x_i = maxAmp$ **and** $x_j = maxAmp$ **do**
4: $j \leftarrow j + 1$
5: **end while**
6: $a \leftarrow i$, $b \leftarrow j - 1$
7: compute Δ_{before}, Δ_{after} and Δ_{total} according to Eqs. (6.1), (6.2) and (6.3)
8: $c \leftarrow b - a$
9: $h \leftarrow \lfloor \frac{c}{2} \rfloor$
10: **for** $i = 1 \rightarrow h - 1$ **do**
11: $x_{a+i} \leftarrow x_{a+i} + (\sqrt{i} \cdot \Delta_{total})$
12: $x_{b-i} \leftarrow x_{b-i} + (\sqrt{i} \cdot \Delta_{total})$
13: **end for**
14: **if** $c \mod 2 = 1$ **then**
15: $x_{a+h} \leftarrow x_{a+h} + (\sqrt{h} \cdot \Delta_{total})$
16: **else if** $|\Delta_{before}| > |\Delta_{after}|$ **then**
17: $x_{b-(h-1)} \leftarrow x_{b-(h-1)} + \Delta_{total}$
18: **else**
19: $x_{a+(h-1)} \leftarrow x_{a+(h-1)} + \Delta_{total}$
20: **end if**
21: $i \leftarrow j + 1$, $j \leftarrow j + 2$
22: **end while**

$$\Delta_{after} = \frac{1}{o} \sum_{i=b}^{b+o} x_i - x_{i+1} \tag{6.2}$$

$$\Delta_{total} = \frac{\Delta_{before} + \Delta_{after}}{2} \tag{6.3}$$

Here, x_i corresponds to the observation measured at time t_i ($1 \leq i \leq n$) and n corresponds to the length of the time series. A truncated data peak is defined to start at time t_a and to end at time t_b. The variable o denotes the number of observations before and after the peak (respectively) that are considered for the computation of the missing data.

Algorithm 1 presents the steps of the peak reconstruction for one axis. The truncated peaks are determined while scanning the time series once (condition in line 2). If two successive maximum amplitudes are detected (the maximum depends on the sensor characteristics and can be set as a parameter *maxAmp*), the algorithm finds the whole truncated time range (line 4) and computes the average gradients (line 7). The variable c denotes the number of subsequent occurring maximum (minimum) values, which is 2 in the most simple case. Using the variable h, the first (line 11) and last $\lfloor \frac{c}{2} \rfloor$ values (line 12) are interpolated based on a weighted total gradient. If c is odd, then the central value in

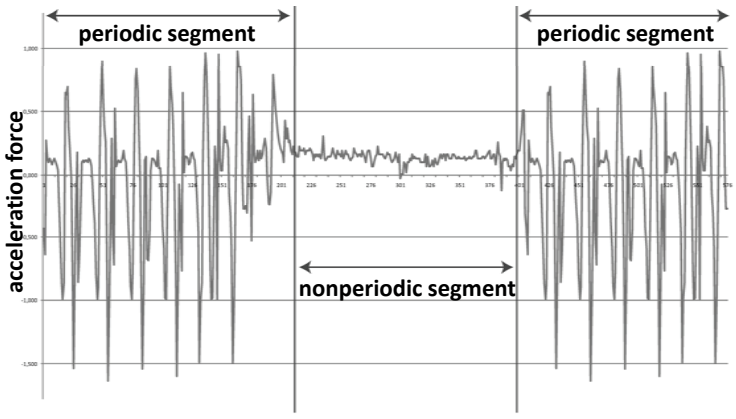


Figure 6.2: An example time series with periodic and nonperiodic segments.

the peak, now represented by x_{a+h}, has to be recomputed in addition (line 15). Otherwise, both amplitudes $x_{a+(h-1)}$ and $x_{b-(h-1)}$ are equal. Then, a global extremal value within this segment is ensured by increasing one of the middle values (lines 16ff).

6.3 Segmentation

Detecting periodic parts in time series is performed via segmentation, where a time series is divided into periodic and nonperiodic sequences (cf. Figure 6.2), in the following called *segments*. Both periodic and nonperiodic sequences are then processed separately based on extracted features (cf. Section 6.4). For a periodic segment, it can be assumed that the detected activity holds for the entire time period, which is due to the periodicity in the signal. A nonperiodic segment may contain activity changes or indefinable acceleration recordings. A feature vector describing a segment is independent of the segment's length.

The first step of the segmentation is the detection of present periodicity. For this purpose, a commonly used method in the community of activity recognition is to apply a sliding window algorithm [131, 174, 201, 202], where autocorrelation [60] is used in order to measure the self-similarity of time series segments. In general, the autocorrelation $\rho(X, t_1, t_2)$ of a time series X at two time stamps t_1 and t_2 is defined as

$$\rho(X, t_1, t_2) = \frac{E[(X_{t_1} - \mu_{t_1})(X_{t_2} - \mu_{t_2})]}{\sigma_{t_1}\sigma_{t_2}}, \tag{6.4}$$

where X_{t_i} denotes the subsequence of X starting at time t_i and μ_{t_i} (σ_{t_i}) denotes the mean (variance) of X_{t_i} ($i = 1, 2$). Hereby, the length of the subsequence is limited to the current reference sample length $\widehat{n}$, which is defined below. The pseudocode of the segmentation algorithm is illustrated in Algorithm 2.

For each starting point of a potential period (denoted by the time stamp t_i, which is initially set to the first time stamp 1), the algorithm consists of two phases: Phase 1 determines the periodicity of the time series X of length n (lines 15-21), and Phase

Algorithm 2 Time Series Segmentation: segmentation(X, τ_ρ, Δ_{max}, n_p)

Require: X, τ_ρ, Δ_{max}, n_p
$i \leftarrow 1$, $L_{np} \leftarrow []$, $n \leftarrow |X|$
while $i \leq n$ **do**
  $\widehat{X} \leftarrow$ extractReferenceSample(X, t_i), $\widehat{n} \leftarrow |\widehat{X}|$
  if $i + \Delta_{max} = n$ **then**
    $\Delta_{max} \leftarrow n - i - \widehat{n} + 1$ {cut Δ_{max} if too long}
  end if
  if $\Delta_{max} < 2 \cdot \widehat{n}$ **then**
    while $i \leq n$ **do**
      L_{np}.add(x_i) {assign remaining observations to a nonperiodic segment}
      $i \leftarrow i + 1$
    end while
    createNonPeriodicSegment(L_{np})
  else
    $\Delta_t \leftarrow l$, $\rho \leftarrow 0$, $\rho_{opt} \leftarrow 0$, $\Delta_{opt} \leftarrow i$
    while $\Delta_t \leq \Delta_{max}$ **do**
      $\rho \leftarrow \rho(\widehat{X}, t_i, t_i + \Delta_t)$ {search for optimal shift Δ_{opt}}
      if $\rho > \rho_{opt}$ **then**
        $\rho_{opt} \leftarrow \rho$, $\Delta_{opt} \leftarrow \Delta_t$
      end if
      $\Delta_t \leftarrow \Delta_t + 1$
    end while
    if $\rho_{opt} < \tau_\rho$ **then**
      L_{np}.add(x_i) {τ_ρ was never exceeded}
      $i \leftarrow i + 1$
    else
      createNonPeriodicSegment(L_{np}) {materialize nonperiodic segment}
      L_{np}.clear()
      $\rho \leftarrow 0$, $j \leftarrow i$
      while $\rho \geq \tau_\rho$ **and** $j \leq n - \widehat{n} + 1$ **do**
        $\rho \leftarrow \rho(\widehat{X}, t_i, t_j)$
        $j \leftarrow j + \Delta_{opt}$ {extend periodic segment}
      end while
      if $j - i + \widehat{n} \geq n_p$ **then**
        createPeriodicSegment(X, t_i, t_{j-1}) {materialize periodic segment}
      else
        createNonPeriodicSegment(X, t_i, t_{j-1}) {materialize nonperiodic segment}
      end if
      $i \leftarrow j$
    end if
  end if
end while

2 extracts the periodic segments (lines 29-32). Therefore, X has to be scanned once completely (condition in line 2). In each iteration, a potential period is represented by a reference sample $\widehat{X}$ of length $|\widehat{X}| = \widehat{n}$, which is extracted from X (line 3). Before starting the process in the actual iteration, a maximum shift range has to be set in order to limit the length of a pattern period and, thus, to increase the matching chances. This range will be denoted by Δ_{max} in the following. Generally, Δ_{max} is set as an input parameter. However, its value has to be decreased if it exceeds the time series length (line 4f). Also, if $\Delta_{max} < 2 \cdot \widehat{n}$ holds, i.e., a shift of the reference sample length $\widehat{n}$ cannot be performed, all remaining observations are inserted in a temporary list called L_{np}, which collects all observations that are regarded to be nonperiodic (line 9). The elements will later be materialized as a nonperiodic segment (line 12). Otherwise, Phase 1 is started.

Phase 1, starting from line 15, detects the optimal window shift between periodic occurrences of patterns. The reference sample $\widehat{X}$ is now matched with a sliding window that contains the following observations of the time series within the maximum shift range Δ_{max}. The window shift Δ_{opt} yielding the highest correlation between $\widehat{X}$ and the pattern occurring in the window is considered as optimal. Thus, the optimal distance between patterns of a periodic cycle is given by Δ_{opt} observations. A sufficiently high correlation value is defined by a threshold τ_ρ, which is also an input parameter of the algorithm. Thus, if no correlation higher than τ_ρ is found within the maximum shift range Δ_{max}, the current optimal shift Δ_{opt} does not contain any significant periodic pattern. Then, the algorithm considers the current start time t_i to be nonperiodic (line 23) and continues with the next iteration using the respective sample sequence shifted by one time stamp, now starting at time stamp t_{i+1} (line 24).

If a periodic pattern (with $\rho_{opt} \geq \tau_\rho$) is found, the remaining observations in the L_{np} list, i.e., the observations that have before been marked as nonperiodic, are combined to a nonperiodic segment (line 26) and removed from the L_{np} list (line 27). Then, Phase 2 extracts periodic segments from the time series X, starting from line 29. The pattern is shifted by Δ_{opt}, yielding the new starting time t_j, and the correlation between the current reference sample and the following subsequence is computed. If a minimum correlation of τ_ρ between X_{t_i} and X_{t_j} is obtained (which is obvious with the first shift, as this was the "highest" correlation value that yielded the value for Δ_{opt}), the periodic segment is extended performing further shifts of length Δ_{opt}. This shifting procedure is continued until a correlation value less than τ_ρ is obtained or the periodic pattern reoccurs until the end of the time series is reached (condition in line 29). A periodic segment is required to have a minimum length n_p. If the extracted segment, thus, consists of n_p or more observations, a periodic segment is created (line 34). Otherwise, a nonperiodic segment is created (line 36). Afterwards, the algorithm restarts from the observation at the first time stamp after the extracted segment (line 38). The segmentation algorithm is finished if all observations have been assigned to a (periodic or nonperiodic) segment. This condition is satisfied after all time stamps of the time series have been tested (line 2).

The presented segmentation algorithm is generally restricted to a single time series. An extension for three-dimensional accelerometer data requires a slight modification. Here,

the optimal pattern distance is chosen from the time series where the highest correlation value is obtained, and it is then applied to all time series. In order to mark a periodic segment, the correlation values ρ_{opt} of all three time series have to exceed τ_ρ (line 22).

In general, the autocorrelation method is sensitive w.r.t. slight deviations of the periodicity. Thus, an activity might be divided into several consecutive periodic segments showing different behavior w.r.t. frequency or intensity of the acceleration.

The runtime complexity of the segmentation algorithm is $O(n \cdot (\Delta_{max} - \widehat{n}))$. In the worst case, the segmenting process has to find the optimal shift within a range of $\Delta_{max} - \widehat{n}$ for each of the n observations.

6.4 Feature Extraction

As already stated in Section 6.3, the segments are of different length, so that classification cannot be directly performed based on the raw segments, since subsequence matching would require a high computational cost. Also, this step will not perform efficiently having segments that consist of a high number of observations. In order to overcome these problems, each segment is represented by a feature vector of the same dimensionality, where the choice of features can vary for periodic and nonperiodic segments. An overview of the final choice of features will be given in Subsection 6.7.2.

The feature vectors used for the approach presented in this chapter contain time-domain features and heuristic features, which will be summarized in the following. Frequency-domain features will not be considered in order to the save computational cost w.r.t. the transformation of the time series to the frequency domain.

For each periodic segment S, it is sufficient to derive only one feature vector v, as common characteristics for a segment follow directly from the periodicity of the observations contained in the segment. If a minimum length of n_p observations for S is assumed for each axis recorded by the accelerometer, the usage of feature vectors now reduces the dimensionality from at least $3 \cdot n_p$ values (regarding all three axes) to d_v, where d_v denotes the dimensionality of v and in general $d_v \ll n_p$ holds. In an exemplary case of $n_p = 100$ and $d_v = 15$ (which will be the final size for the experimental part), this corresponds to a reduction of at least 95%. For nonperiodic segments, it cannot be assumed that all observations were captured with the same physical activity, as no periodicity has been detected. Thus, a single feature vector would not represent the entire segment that well. In order to minimize this error, a nonperiodic segment is again split up into subsegments having a number of n_{np} observations, each represented by its own feature vector. In most cases, the last subsegment contains less than n_{np} observations, since a nonperiodic segment, in general, varies in its length. If the last subsegment contains more than $\frac{n_{np}}{2}$ observations, another feature vector is computed. If this is not the case, this subsegment will be neglected in the classification step, as it hardly contains sufficiently enough information for the creation of some features, such as the ARC feature, which will be explained in the following. However, this subsegment will be considered again in the reclassification step, which is performed after the classification (cf. Section 6.6).

The following features will be examined in the context of the proposed approach.

Auto Regression Coefficients (ARC)

The autoregressive model is commonly used in the context of modeling time series [60]. Let x_i be the observation of a time series X at time t_i. x_i is modeled as the sum of p linearly weighted recent values. The autoregressive model of order p is given by

$$x_i = \sum_{j=1}^{p} a_j \cdot x_{(i-j)} + \varepsilon_i, \tag{6.5}$$

where a_j is the jth autoregressive coefficient and ε_i is a noise term at time t_i. Given a time series X, the coefficients a_j can be estimated in various ways, such as the method of least squares [57].

Autoregressive coefficients predict the prospective course of a signal based on recent values and have been used in [126, 127] in the context of activity recognition. For the solution of this work, the first three coefficients for each axis of the accelerometer data are used, yielding nine feature values.

Signal Magnitude Area (SMA)

The *Signal Magnitude Area* (*SMA*) is a well-known heuristic energy measure in the context of activity classification [13, 58, 118, 127, 212]. It is computed by summing up the absolute observation amplitudes of the three accelerometer axes and by normalizing the result w.r.t. the length of the corresponding segment S, i.e.,

$$SMA = \frac{1}{n'} \sum_{i=1}^{n'} (|x_i| + |y_i| + |z_i|). \tag{6.6}$$

Hereby, n' corresponds to the length of S, and x_i, y_i and z_i are the values of the respective axis at time t_i. As the maximum amplitudes might vary for each axis, activities showing intense acceleration values occurring with high frequency contribute to a high SMA value, whereas low-acceleration activities result in a low SMA.

Tilt Angle (TA)

The *Tilt Angle* (*TA*) feature is described by the average tilt angle of the lower leg over time. The accelerometer is supposed to be worn in the same position at the ankle. Hence, physical activity can be described by the angle ϑ between the gravitational vector and the positive z-axis of the sensor, i.e., $\vartheta = \arccos(z)$. Recognition of activities like swimming, which enforces a different tilt angle of the lower leg, takes considerable advantage of the TA feature. The TA feature has been used in [127] and will, thus, be examined in the experimental evaluation.

Algorithm 3 Peak Detection: detectPeaks(S, p_{min}, τ_{min}, Δ_τ, *minDist*)

Require: S, p_{min}, τ_{min}, Δ_τ, *minDist*

1: $S_{max} \leftarrow \arg\max_{x \in S}(S)$
2: $\tau \leftarrow S_{max}$
3: $P_{Peak} \leftarrow \{x \in S : x \geq \tau\}$
4: **while** $|P_{Peak}| < p_{min}$ **and** $\tau > \tau_{min}$ **do**
5: $\quad \tau_{old} \leftarrow \tau$
6: $\quad \tau \leftarrow \tau - \Delta_\tau \cdot S_{max}$
7: $\quad P_{Peak}.add(\{x \in S : \tau_{old} > x \geq \tau\})$
8: $\quad$ **for all** $x_i, x_j \in P_{Peak}$ **do**
9: $\quad\quad$ **if** $t_i - t_j \leq minDist$ **then**
10: $\quad\quad\quad P_{Peak} \leftarrow P_{Peak} \setminus \{\min(x_i, x_j)\}$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end while**
14: $avg \leftarrow \frac{1}{|P_{Peak}|} \sum_{x \in P_{Peak}} x$
15: **if** $|P_{Peak}| \geq p_{min}$ **then**
16: $\quad$ **return** avg
17: **else**
18: $\quad$ **return** $sgn(avg)$
19: **end if**

Average Peak Amplitude (APA)

The *Average Peak Amplitude* (*APA*) will be introduced, which is an energy measure and is, in contrast to the SMA, restricted to the (positive and negative) peak amplitudes within a temporal window of the current segment S.

The process of identifying the peaks is outlined in Algorithm 3. The signal has been cleaned w.r.t. outliers beforehand (cf. Subsection 6.2.1), so that the set of detected peaks is not influenced by erroneous observations. The actual identification step first determines the global absolute extremum of S, denoted by S_{max} (line 1).

Next, a threshold τ is introduced, which defines a minimum amplitude for all potential peaks contained in S. τ is initialized with the value of S_{max} (line 2). The set of observations w.r.t. τ, denoted by P_{Peak}, is initially created (line 3) . As long as the algorithm has not yet detected a mandatory minimum number of peaks p_{min}, τ is decreased by $\Delta_\tau \cdot S_{max}$ (line 6), yielding new elements for P_{Peak} (line 7).

P_{Peak} may contain neighboring observations that actually belong to the same peak. In order to reduce the result to a single observation per peak, a minimum distance *minDist* between peaks is introduced that has to hold. If two amplitudes identified as peaks show a distance less than *minDist*, the observation with the lower amplitude value will be removed (line 10; here, a numerical ordering w.r.t. the amplitudes is assumed for the observations).

The described procedure is repeated until p_{min} is reached or τ has reached a minimum

value of τ_{min} (condition in line 4). In the latter case, S contains only few significant amplidudes.

Finally, the feature value represents the average of all peak amplitudes in the current segment, where the APA is only considered as significant if the number of peaks is at least p_{min} (lines 15ff). Otherwise, a default value depending on the amplitude sign is returned.

Surrounding Segmentation Rate (SSR)

Physical activity classes can also differ in their fraction of periodic segments. For example, in the context of this work, the sensor recordings for the activity *Cycling* showed to contain more nonperiodic segments than *Running*, as periodic movements often have been interrupted by external influencing factors. This observation leads to the derivation of a simple, but suitable heuristic feature describing the *Surrounding Segmentation Rate* (*SSR*). The computation of the SSR is performed for a temporal window of w_{SSR} seconds surrounding the current segment, which is in particular suitable for long-term activities. Thus, for a window containing overall s segments, s_p and s_{np} denote the numbers of periodic and nonperiodic segments, respectively; it holds that $s = s_p + s_{np}$. Then, the SSR is computed by $SSR = \frac{s_p}{s}$.

6.5 Dimensionality Reduction

6.5.1 Feature Selection

In order to reduce the computational effort of the actual classification process, a dimensionality reduction of feature vectors is typically applied in order to remove redundant information; this decreases the size of the feature vectors. In the context of accelerometer data, the literature distinguishes between methods of *feature selection* and *feature transformation*, which can also be used in combination. The selection of most relevant features for the feature vector was performed using the *forward-backward search* [218]. This method can be applied to reduce the effort of testing all feature combinations, which would be exponential in the number of features. This is achieved by alternately adding features to the feature vector with the currently best quality (which yields a new feature combination) and removing features (to examine subsets of combinations that have not been considered before).

6.5.2 Feature Transformation

The separation of different activity classes can further be supported applying the *Linear Discriminant Analysis* (*LDA*) [95] on the feature vector after the feature selection step. This is called *feature transformation*. The LDA minimizes the variance of features within a class and maximizes the variance of features between different classes, having the side effect of slight performance gain. Applying the benefits of the LDA to the current application scenario, this step neglects person-specific differences with the same physical activity, which

Algorithm 4 Reclassification of Observation x_i: reclassObservation(x_i, W, cl)

Require: x_i, W, cl
1: $w \leftarrow |W|$
2: $\mathcal{I} \leftarrow \{0, \ldots, cl-1\}$
3: $\mathcal{W} \leftarrow [0, \ldots, 0]$
4: $L_{old} \leftarrow x_i.label, L_{new} \leftarrow -1$
5: **for all** $x_j \in W \setminus \{x_i\}$ **do**
6: $x_j.weight \leftarrow \frac{w-1}{2} - |j-i| + 1$
7: $\mathcal{W}[x_j.label] \leftarrow \mathcal{W}[x_j.label] + x_j.weight$
8: **end for**
9: $L_{new} \leftarrow \arg\max_{l \in \mathcal{I}}(\mathcal{W})$
10: **if** $L_{old} \neq -1$ **or** $L_{old} \neq L_{new}$ **and** $\mathcal{W}[L_{new}] > \sum_{l \in \mathcal{I} \setminus \{L_{new}\}}(\mathcal{W}[l])$ **then**
11: $x_i.label \leftarrow L_{new}$
12: **end if**

is caused by different body heights or variations in the execution of movements. Finally, the LDA leads to a robustness of the classification w.r.t. the exact position of the sensor. Despite the fact that the sensor is fixed on the ankle, continuous movements can lead to a rotation or a shift of the sensor, which influences the quality of the data and, thus, the quality of the classification results. Applying the LDA, these errors can be corrected.

6.6 Reclassification

In order to classify short subsegments where no features were extracted (cf. Section 6.4), a postprocessing step is applied which assigns the most likely class label to these subsegments. This likelihood depends on the classification results obtained for the surrounding segments. Furthermore, this step detects errors that occurred in the actual classification step. These errors may contain highly improbable results. The application of activity recognition provides sufficiently interpretable information for these cases. For example, if two significantly long segments classified as *Cycling* contain a short segment classified as *Elliptical Trainer*, the classification result of the latter segment will be revised. A formal description of the unsupervised reclassification is outlined in Algorithm 4.

Hereby, the variable cl denotes the number of activity classes; also, the activity class labels are represented as indices. Temporally used data structures are the list of class label indices $\mathcal{I}$ and the list of weights $\mathcal{W}$ of the class labels; the weight values are referenced by the respective class label index $i \in \{0, \ldots, cl-1\}$. The reclassification step takes, for each observation, the available information of the neighboring observations into account by considering a temporal window W of size $w = |W|$, containing the current observation x_i as well as $\frac{w-1}{2}$ preceding and $\frac{w-1}{2}$ successive observations. For observations that are close to the border of the time series, W is cut off accordingly. Based on the class labels of each observation x_j contained in W ($j \neq i$), a weighted linear distribution of the occurring

Activity	# Datasets	Duration **(hh:mm:ss)**
Ell. Trainer	3	00:55:07
Walking	6	02:42:09
IL Skating	3	01:55:50
Running	6	02:37:13
Cycling	4	02:32:46
Total	22	10:43:05

Table 6.2: Datasets used for the experimental evaluation.

labels is computed, which considers more recent observations to have more impact. Thus, the distance-based weight $x_j.weight$ of a neighboring observation x_j corresponds to $\frac{w-1}{2}$, whereas the weight of the most distant observation is 1 (line 6). The distribution of the weights of the observations x_j corresponds to a linear time-fading function. A quadratic or general distribution-based fading function would also be applicable here. If x_i has not been classified before (the label obtained in the classification is denoted by L_{old}, where a value of -1 implies no assignment to a class label) or the class label L_{new} that shows the highest weighted occurrence has a significant influence on x_i (i.e., its relative weighted occurrence is higher than the sum of all other classes), the reclassification was successful and L_{new} is assigned to x_i (line 11).

For the reclassification of each of the n observations, the surrounding $w-1$ observations within the window W have to be regarded; thus, this algorithm requires a runtime complexity of $O(n \cdot w)$. In order to obtain a speed-up of this process for practical use, this step can be parallelized by the *Knowing* framework (cf. Chapter 5), which is able to exploit multi-processor systems.

6.7 Experimental Evaluation

6.7.1 Datasets

The application scenario for the presented approach was given by a collaboration with the *Sendsor GmbH*, who also provided the accelerometers used to record the datasets that will be used in this section. The accelerometers record amplitudes in the range of $\pm 2g$ with a rate of 25 Hz. In order to obtain accurate and representative acceleration measurements, the accelerometer is worn by the patients at the ankle [196].

In the context of this chapter, five different activity classes were examined: *Walking*, *Running*, *Cycling*, *In-line Skating* (*IL Skating*) and *Elliptical Trainer* (*Ell. Trainer*). The datasets used for the following experiments are summarized in Table 6.2. The evaluation was performed using the *Knowing* framework [41] (cf. Chapter 5).

6.7.2 Experimental Setup

Choice of the Classifier

The evaluation of the presented activity classification approach was performed using the *Naïve Bayes* classifier [106]. In the context of implementing the *Knowing* framework, an evaluation of overall 32 classifiers that are available in *WEKA* [102] has been performed, where Naïve Bayes turned out to provide solutions of superior quality on periodic and nonperiodic segments to the other classifiers. Hereby, the effectiveness of the classifiers was measured by the *classification accuracy* or *recognition rate*, which is a common quality measure for classifiers for assigning new data to a specific class label [103] and which is in particular used in the field activity recognition [17]. According to the semantics of a classification [103], objects are divided into positive (P) and negative (N) objects, which denote the number of objects that are returned by a classifier w.r.t. a label and the number of objects that have been discarded, respectively. The classification accuracy yields values between 0 and 1 (i.e., 1 if all objects are correctly classified) and is computed as

$$accuracy = \frac{TP + TN}{P + N},$$

where TP denotes the number of positive objects that are correctly recognized (i.e., that are expected to belong to the result) and TN denotes the number of negative objects that have been rightly discarded. In the case of activity recognition, the accuracy denotes the amount of correctly labeled segments.

The first experiment was performed without applying reclassification. Naïve Bayes yielded a classification accuracy of 97.18% (more details will be provided in Subsection 6.7.3). Results of slightly minor quality were obtained using Sequential Minimum Optimization [172] (accuracy of 96.67%) and a normalized Gaussian radial basis function network (accuracy of 94.88%).

In addition, two methods based on *Artificial Neural Networks* (*ANNs*) were tested in order to provide the comparability to the approach of [127]. The latter uses a multilevel perceptron based on *backpropagation learning* [181, 208], which is available in *WEKA*. The second evaluated ANN, which is based on *resilient propagation learning* [179], is available in the *Encog Machine Learning Framework*[2]. In the evaluated settings, each of the neural networks consisted of a hidden layer of ten neurons and an output layer of five neurons, which corresponds to the number of evaluated activities. While the resilient propagation classifier yielded a total classification accuracy of 93.43%, the backpropagation classifier achieved a very low accuracy value of 27.7%. In addition, the backpropagation classifier required extensive computational cost on the used dataset. Thus, for the comparison experiments (cf. Subsection 6.7.3), the resilient propagation classifier was applied to be used with the approach of [127] instead.

[2] `http://www.heatonresearch.com/encog`

Description	Abbreviation	# Values	[127]	This Approach [39]
Auto Regression Coefficients	ARC	9	✓	✓
Signal Magnitude Area	SMA	1	✓	✓
Tilt Angle	TA	1	✓	✓
Average Peak Amplitude	APA	3	-	✓
Surrounding Segmentation Rate	SSR	1	-	✓

Table 6.3: Set of used features.

Feature Selection

The selection of most relevant features for the feature vector was performed using the forward-backward search technique [218] (cf. Subsection 6.5.1). For periodic segments, the feature selection step yielded a feature vector dimensionality of $d_v = 15$. For the nonperiodic segments, the selection process yielded the same feature vector as for periodic segments. An overview of the used features for the evaluation of the current approach and the competing approach of [127] is given in Table 6.3. In addition to the features presented in Section 6.4, the simple features *Arithmetic Mean*, *Variance* and *Inter-Axis Correlation*, used in [20], were included into the selection process, but proved to contain no significant information.

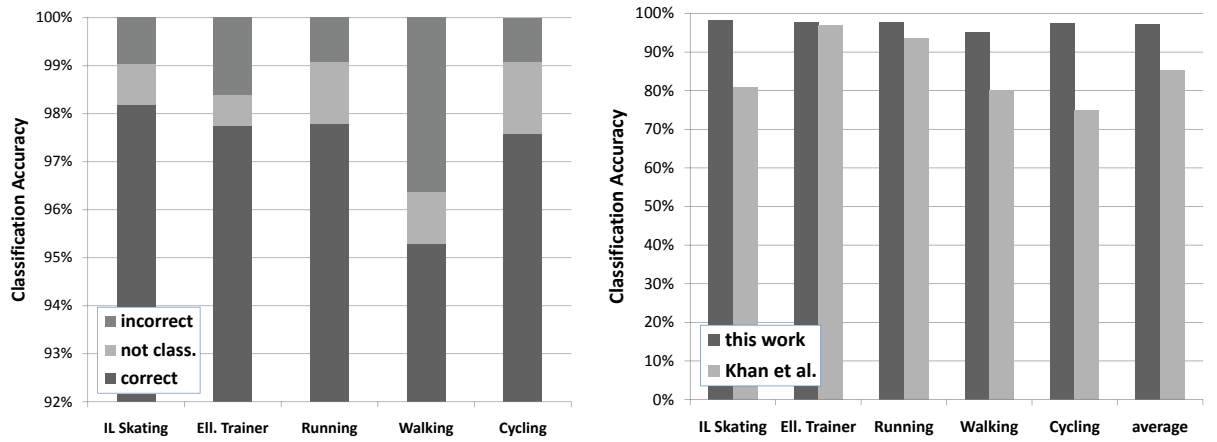
Further Parameter Settings

For creating the experimental setup for the following experiments, some parameters were set to default values. The window size for the average filter that is applied as a preprocessing step to remove outliers (cf. Section 6.2) was set to 3. The number of observations o that are considered for the peak reconstruction was set to 2.

The settings for the extracted features (cf. Section 6.4) are the following: The length n_{np} of nonperiodic subsegments was set to 80 (3.2 seconds). For the APA feature, values for the minimum peak threshold ($\tau_{min} = 0.7$), the peak threshold step ($\Delta_\tau = 0.02$), the minimum number of peaks ($p_{min} = 3$) and the minimum distance between peaks ($minDist = 10$) were set. For the SSR feature, the window size w_{SSR} was set to 1500, which corresponds to a temporal window of 60 seconds.

For the segmentation (cf. Section 6.3), the following default values were used: The reference sample $\widehat{X}$ consisted of $\widehat{n} = 25$ observations, which corresponds to one second. The required minimum correlation τ_ρ was set to 75%. The minimum length n_p of periodic segments was set to 100 observations, which corresponds to four seconds.

In the reclassification step (cf. Section 6.6), the size w for the temporal window, which is used to capture the weighted occurrences of the class information of the surrounding observations, consists of 751 observations (750 plus the observation that is to reclassify), which corresponds to 15 seconds before and after the current observation, respectively. Furthermore, as five activity classes are evaluated in this chapter, $cl = 5$.



(a) Overview. (b) Comparison of the results of this chapter and [127].

Figure 6.3: Classification result.

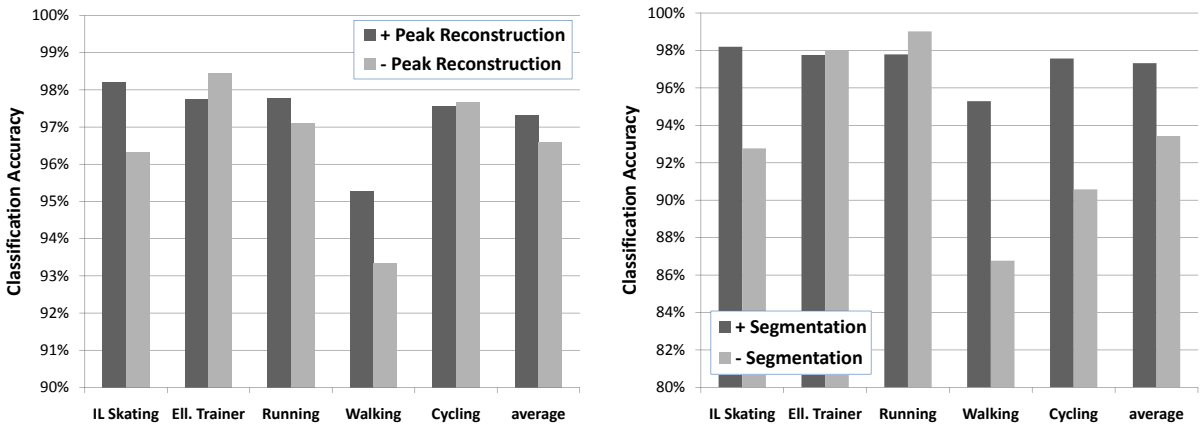
6.7.3 Classification Results

The classification algorithm was evaluated by performing a cross validation on the activity classes. For each time series segment containing one of the five activities (cf. Subsection 6.7.1), the obtained classification accuracy using the default settings of Subsection 6.7.2 are depicted in Figure 6.3(a). The classification yields accuracies of more than 95% for each activity. The highest classification error was obtained with the activity *Walking*, which was classified as *Cycling* with 3.56%, which can simply be explained by the observation that these activities are likely to create similar accelerations. In order to visualize the percentage of segments that were incorrectly classified or could not be classified at all, the reclassification step was omitted in the first experiment. The following experiments also neglect this step. Finally, the effect of the reclassification will be examined in Subsection 6.7.7.

In [127], the classification of 15 different activities yielded an accuracy of 97.9%. For the evaluation in the context of this chapter, a slight adaption of this approach was implemented: the resilient propagation algorithm was used instead of the usually applied backpropagation algorithm due to performance reasons (cf. Subsection 6.7.2). Figure 6.3(b) illustrates the classification results of the approach introduced in this chapter in comparison with the results of [127]. It can be observed that, for each class, the approach of [127] achieves less accuracy than the approach presented in this chapter.

6.7.4 Effect of the Preprocessing Steps

The next experiment will examine the effect of the preprocessing steps. Evaluating the peak reconstruction (cf. Subsection 6.2.2), the classification results could be improved for three out of five activities (*Walking*, *Running* and *In-line Skating*). This can be explained by the fact that these activities take advantage of their significant peaks because of significant movements, whereas the movements of *Cycling* and *Elliptical Trainer* are indirectly



(a) Effect of the peak reconstruction.

(b) Effect of the segmentation.

Figure 6.4: Effect of the peak reconstruction and the segmentation.

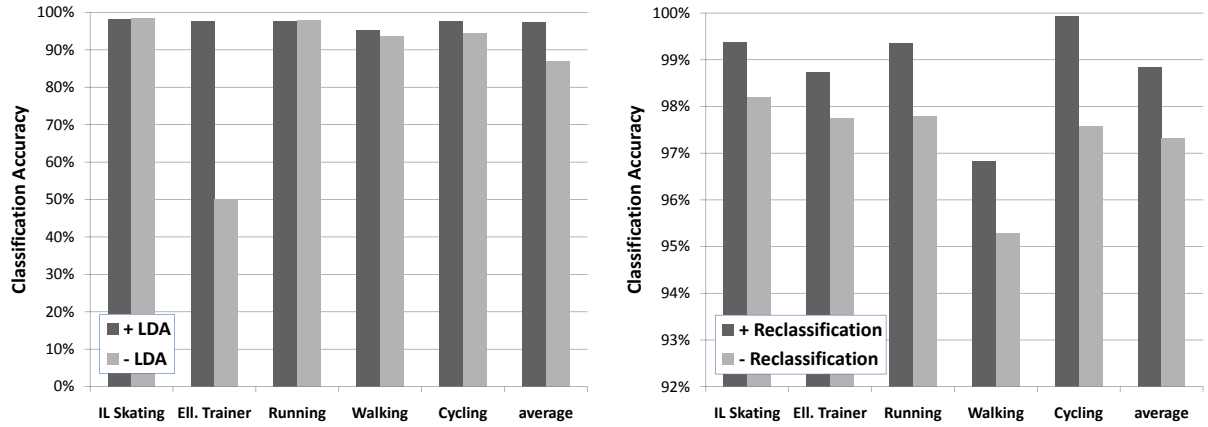
supported by the sports equipment, which may lead to rather smooth movements. Overall, this step yielded an overall precision gain of almost 1% (cf. Figure 6.4(a)).

6.7.5 Effect of the Segmentation

Next, the effect of the segmentation (cf. Section 6.3) was evaluated. Instead of choosing a naïve solution without a segmentation that extracts only one feature vector for the time series, the time series was divided into non-overlapping subsequences of equal length, thus neglecting the periodicity. Each subsequence contained 80 observations, analogously to the size of nonperiodic segments in the original approach (cf. the experimental setup in Subsection 6.7.2), and a feature vector was derived for each segment. The SSR feature could not be applied here, as, for this segmentation variant, no information about the amount of surrounding periodic segments is available. Hence, the used feature vector consisted of 14 features for this variant of the classification approach. The results are shown in Figure 6.4(b). For long-term activities that are very constant over time, such as *Running* and *Elliptical Trainer*, the equal-length segmentation yields comparable results, as there are no gaps in the data which are hard to classify. For activities consisting of short-term periods interrupted by several breaks due to external influence factors, e.g., in the case of *Cycling*, where pedaling is often noncontinuous, a classification supported by a segmentation into periodic and nonperiodic parts achieves a significant improvement of 4% in average. Similar observations explain the significant improvement with the activities *Walking* and *In-line Skating*, as the step length is not homogeneous.

6.7.6 Effect of the Feature Transformation

In the next experiment, the effect of the LDA (cf. Subsection 6.5.2) was examined. With the most activity classes, the LDA improves the results only slightly. This shows that the



(a) Effect of the LDA.

(b) Effect of the reclassification.

Figure 6.5: Effect of the LDA and the reclassification.

combination of features done by the forward-backward search already yielded representative feature vectors with almost no redundant information. In the processing chain, the forward-backward search step is performed before the application of the LDA.

The only activity that obtains a significant performance gain with applying the LDA is the activity *Elliptical Trainer*. As this activity is, intuitively, very similar to both activities *Walking* and *Cycling*, an accurate separation among these classes is not always possible. The classification errors(27% classified as *Walking*, 23% classified as *Cycling*) prove this intuition. Moreover, the training datasets for this activity class seem to be very inhomogeneous due to significantly different velocities. Here, the LDA maximizes the differences to the other activity classes successfully. Thus, these errors can be corrected. The results of this comparison are depicted in Figure 6.5(a).

6.7.7 Effect of the Reclassification

The reclassification step was omitted with the evaluations of Subsections 6.7.3 to 6.7.6 in order to get the amount of unclassified data returned. Finally, the observed results with an additional application of the reclassification step are illustrated in Figure 6.5(b)). Here, a slight improvement of 1.6% was achieved. Most nonperiodic segments that could not be classified in the actual classification step seem to contain many activity changes within a short time period, which leads to errors in the reclassification step.

6.7.8 Conclusions

Concluding, it can be stated that the proposed approach achieves results of high quality, since a state-of-the-art activity recognition method could be outperformed. The evaluation of the processing steps showed that each of them supports the actual classification step in order to provide reliable evidence about the performed activity. For the case of limited

resources, i.e., if the classification algorithm has to run on the firmware of the sensor in order to provide a real-time recognition and to return results quickly, a trade-off solution between storage and processing time requirements and classification accuracy has to be found. Among further potentials for future work, this issue will be picked up again in Chapter 18.

6.8 Summary

This chapter provided an effective solution for the application scenario of activity recognition on periodic time series that are collected from accelerometers. The proposed solution extends existing methods by integrating additional processing steps, such as a reconstruction of the data peaks and a reclassification step as well as a utilization of suitable features to improve the classification results. The experimental part showed an improved recognition quality in comparison with existing work.

Chapter 7

Accelerating Similarity Processing in High-Dimensional Feature Spaces

7.1 Introduction

Chapter 6 focused on similarity processing on time series in the context of activity recognition. An essential step of this approach is the extraction of features, which allows to accelerate similarity processing by indexing techniques. Therefore, common solutions like the R^*-tree [23] are appropriate. With increasing dimensionality, however, index structures degrade rapidly due to the curse of dimensionality [24]. A solution is provided by commonly applied methods enhancing the sequential scan [207] or by using a hybrid structure, which is tree-based, but requires to scan successive blocks or node elements [27, 28].

BOND [85] is a search strategy enhancing the sequential scan. Contrary to the aforementioned techniques, *BOND* exploits modifications w.r.t. the physical database design. The basic idea is to use a columnstore architecture (as known from NoSQL database systems), sort the columns according to their potential impact on distances and prune columns if their impact becomes too small to change the query result. By the design of this method, subspace queries can also be facilitated implicitly with the same architecture.

However, *BOND* is motivated by the application of metrics for image retrieval and, thus, requires particular properties of a dataset which restricts the application considerably:

1. The first metric proposed in *BOND* is only applicable to normalized histogram data.

2. Using the Euclidean distance, still the length of each vector is required for pruning columns with low impact.

3. Stricter bounds for the Euclidean distance metric further improve the pruning, but require Zipfian distributed data (similarly to hyperbolic functions, e.g., color or gray scale histograms) and a particular resolve order of the columns in the database.

This chapter specifically focuses on extending *BOND* by loosening the restrictions of its use for datasets and by improving the pruning power, still providing a scan-based solution.

The rest of this chapter is organized as follows: Section 7.2 will summarize the benefits of *BOND*, but also outline its deficiencies w.r.t. the mentioned aspects. Afterwards, the proposed extensions of this chapter will be described in Section 7.3. The experiments in Section 7.4 will include an extensive evaluation and will demonstrate the improved performance in full-dimensional search. Finally, Section 7.5 will conclude this chapter.

7.2 *BOND* revisited

7.2.1 General Processing

Processing multi-step queries using a filter-refinement framework, traditional index approaches resolve the data of feature vectors row-wise (horizontally) in order to obtain their exact representation. The main advantage of *BOND* is that feature vectors are resolved column-wise (vertically) so that the values of a feature vector x are obtained successively. Thus, the resolved part of x is known exactly, whereas the unresolved part has to be approximated. This approach is inherently different from traditional tree-indexing approaches where a feature vector is either completely approximated or completely available. In order to avoid possibly unnecessary I/O-operations, traditional tree-indexing techniques aim at avoiding to resolve as many feature vectors as possible which are not part of the result set. On the contrary, *BOND* starts with resolving all feature vectors column by column and tries to approximate the remaining part of the feature vectors. As soon as the approximation yields a sufficiently high pruning power, false candidates can be pruned, so that the remaining dimensions of these feature vectors do not have to be resolved. *BOND* supports regular k-nearest neighbor (k-NN) queries on the full dimensionality as well as on weighted subspaces. In this chapter, the full-dimensional case will be analyzed.

The main goal of the pruning statistics used in *BOND* is to tighten the distance approximations of the yet unresolved parts of the feature vectors in order to prune false candidates as soon as possible before unnecessarily resolving additional columns for these vectors.

The rest of this chapter follows the notation of [85], where $q \in \mathbb{R}^d$ denotes a d-dimensional query vector and $x \in \mathbb{R}^d$ denotes an arbitrary d-dimensional feature vector of the database. Any database vector x can be split into a resolved part $x^- \in \mathbb{R}^{d^-}$ and an unresolved part $x^+ \in \mathbb{R}^{d^+}$, so that $x = x^- \cup x^+$ and $d = d^- + d^+$. The variable $d^- \in [1, d]$ denotes the number of columns that have been resolved so far. The distance $dist(q, x)$ between q and x can, thus, be approximated by a composition of the exact distance w.r.t. the resolved part plus the approximation of the unresolved part. Depending on the type of the bound, the approximation of $dist(q, x)$ is the following:

$$maxDist(q, x) = dist(q^-, x^-) + maxDist(q^+, x^+) \geq dist(q, x) \tag{7.1}$$

$$minDist(q, x) = dist(q^-, x^-) + minDist(q^+, x^+) \leq dist(q, x) \tag{7.2}$$

Performing a k-NN query, the resulting distance bounds are then used to refine the candidate set in a traditional way, where all candidates are pruned if their lower distance

bound is greater than the kth smallest upper bound. The distance $dist(q^-, x^-)$ between the known parts of q and x can be computed precisely. Concerning the unknown part (x^+), an approximation for the lower and upper distance bounds to the query vector q needs to be created.

In the following, the approximations of *BOND* based on the Euclidean distance will be summarized. The additionally introduced approximations for the histogram intersection metric will be omitted, as the solution that will be provided in this chapter aims at being independent of the application to histogram data.

7.2.2 Simple Approximation

The basic approach of *BOND* uses the application scenario of histogram data, where the length of each data vector can safely be assumed to be 1. Relaxing this condition, an extension of the basic approach assumes the unit hypercube $[0, 1]^d$ as the data space and is based on the Euclidean distance. Thus, a lower and an upper approximation for the exact distance

$$dist(q, x) = \sum_{i=1}^{d} (q_i - x_i)^2 \tag{7.3}$$

between the query vector q and a database vector x has to be found, where the resolved part of the approximations corresponds to

$$dist(q^-, x^-) = \sum_{i=1}^{d^-} (q_i^- - x_i^-)^2. \tag{7.4}$$

The extension proposed in [85] does not rely on any distribution or assumption of the data, as it only depends on q, so that the approximations *maxDist* and *minDist*, which denote the upper and lower distance bound between q and x, respectively, are derived by

$$maxDist(q^+, x^+) = \sum_{i=d^-+1}^{d} \max(q_i^+, 1 - q_i^+)^2 \text{ and} \tag{7.5}$$

$$minDist(q^+, x^+) = 0. \tag{7.6}$$

The advantage of being independent of the data distribution and of the resolve order is paid with the loss of pruning power. The weakness of these bounds is obvious, in particular for the lower bound, which assumes the distance of 0 for the remaining (unresolved) subspace, and the upper bound only takes the values of the query vector into account and does not make any assumptions on the database vectors.

7.2.3 Advanced Approximation

An extension yielding tighter approximations relies on the partial sums of the query vector q and each database vector x, which are stored as the aggregated values of q and x for the

dimensions that have not been resolved yet. In the following, these sums will denoted by $T(q^+)$ and $T(x^+)$. Furthermore, this extension assumes a skewed Zipfian distribution of the data, which allows resolving the query vector q in decreasing order and applying the obtained resolve order to x. Considering a d-dimensional database and d^- already resolved dimensions, Equations (7.5) and (7.6) are replaced by

$$maxDist(q^+, x^+) = \sum_{i=d^-+1}^{l-1} q_i^2 + (u_l - q_l)^2 + \sum_{i=l+1}^{d} (1 - q_i)^2 \text{ and} \tag{7.7}$$

$$minDist(q^+, x^+) = \frac{(T(x^+) - T(q^+))^2}{d^+}, \tag{7.8}$$

where $l = d - \lfloor T(x^+) \rfloor$ and $u_l = T(x^+) - \lfloor T(x^+) \rfloor$. Although this method provides the best results in [85], the bounds computed by this method quickly lose their pruning power if the data distribution changes. This method strictly requires a particular resolve order of the columns in the database, which is not optimal in the case of other distributions or in case of correlated dimensions. However, changing the resolve order is not an option, because this would invalidate the proof for the correctness of the pruning bounds suggested by [85]. Concerning the advantages and disadvantages of *BOND*, it can be stated that *BOND* achieves very good performance on datasets that follow a skewed Zipfian distribution (like color or gray scale histograms). In this case, a large number of distance computations and I/O-operations can be saved compared to the sequential scan.

7.3 Beyond *BOND*

7.3.1 Restrictions of *BOND*

One of the main limitations of *BOND* is the dependency of the data distribution. The distance approximations proposed in [85] work well as long as the data follows a skewed Zipfian distribution like in the case of color histograms and if the database columns are resolved in decreasing order of the query feature values. If either of the conditions is not satisfied, *BOND* quickly degenerates, i.e., possibly the complete dataset needs to be resolved to answer the query. The approach *BeyOND* extends the original idea of *BOND* in order to supply a query system that allows efficient execution of k-NN queries on datasets that follow an arbitrary or unknown distribution, so that the following restrictions are removed:

1. *BeyOND* does not depend on the data distribution, so any distance metric can be employed that provides valid upper and lower distance approximations.

2. The values x_i ($i \in \{1, \ldots, d\}$) of the feature vectors are no more restricted to $x_i \in [0, 1]$ in each dimension.

3. *BeyOND* does not rely on a specific resolve order of the query vector, so more sophisticated resolve techniques can be applied to further increase the pruning power.

Removing the first and third restriction also disables the possibility to use the improved distance approximations of [85] summarized in Section 7.2.3. Thus, the starting point is to improve the weak approximations shown in Subsection 7.2.2 and in particular in Equations (7.5) and (7.6).

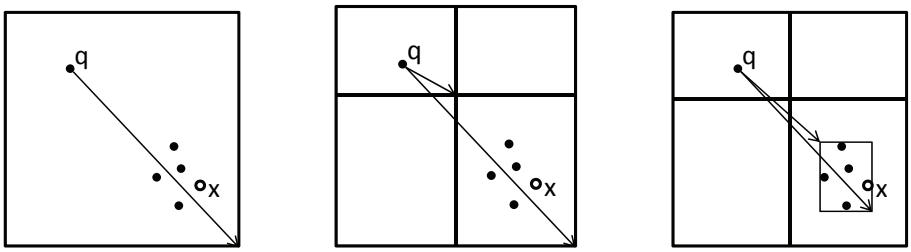
In the following, this section will describe how *BeyOND* combines the concepts of *BOND* and the VA-file [207] by introducing *subcubes* (Subsection 7.3.2), which approximate sets of feature vectors, and by gaining further approximation quality by minimum bounding rectangles (MBRs) for particular subcubes (Subsection 7.3.3). Thereby, the concepts will be based on a *BOND*-style columnstore architecture, such that it is still possible to resolve the dataset in a column-wise manner. A restriction that remains in *BeyOND*, however, is the embedding into a non-unit hypercube, so that the minimum and maximum values of each dimension need to be known.

7.3.2 Subcubes

The first proposed extension is to pick up the idea of the VA-file [207] by splitting the cube once in each dimension w.r.t. the median of the data. Thus, the hypercube describing the feature space is partitioned into 2^d pairwise disjunct subcubes. Each subcube can be identified by the according Z-Order ID (Z-ID), which is stored as a memory-efficient bit representation. This Z-ID is stored additionally to the values of each feature vector. The locations of the split positions in each dimension are stored in separate arrays, so that quantile splits are also supported. Assuming that the feature vectors are composed of 8 byte double values, the memory consumption of a feature vector increases by a value of $\lceil \frac{sd}{8} \rceil$ bytes with s denoting the number of split levels. It would also be possible to increase the split level of the cubes even further. Nevertheless, each additional split also directly increases the size of the Z-IDs. This leads to a trade-off between additional memory consumption from larger Z-IDs and tighter approximations for the upper and lower bounds of the distances due to smaller subcubes. An evaluation of the impact of additional split levels will be shown in the experimental part (cf. Section 7.4). Given a Z-ID of a feature vector and the coordinate arrays containing the split positions, it is a computationally cheap task to recreate the coordinates of the according subcube, so that the bounds of potentially 2^d subcubes need not be kept in memory but can be quickly recomputed on demand.

The subcubes provide the advantage that the upper and lower distance approximations do not have to be computed w.r.t. the complete hypercube that encloses the feature space but only between the cubes containing the query vector and the feature vectors of the database. Thereby, the following two cases have to be considered, where Z_q and Z_x are the Z-IDs of the query vector q and a vector x of the database, respectively:

- $Z_q = Z_x$ indicates that both q and x share the same subcube, so the upper bound of the distance approximation can be lowered to the borders of this subcube (cf. Equation (7.9)). The lower distance remains 0 for all unresolved dimensions (cf. Equation (7.10)).



(a) No split. (b) One split. (c) Split with MBR.

Figure 7.1: Improvement of the upper/lower distance approximation.

- $Z_q \neq Z_x$ implies that q and x are located in different subcubes, so the lower distance approximation can be raised to the minimum distance of q to the subcube containing x (cf. Equation (7.10)). The upper distance approximation is then computed w.r.t. the bounds of the subcube containing x using Equation (7.9). Compared to approximating the upper distance w.r.t. the complete hypercube, this decreases the upper bound when both subcubes share a common plane, which is the case in $d-2$ out of $d-1$ cases (cf. Figures 7.1(a) and 7.1(b)). The obtained bounds now are

$$maxDist(q^+, x^+) = \sum_{i=d^-+1}^{d} \max\left(|q_i - c_{x_i}^{min}|, |q_i - c_{x_i}^{max}|\right)^2 \text{ and} \tag{7.9}$$

$$minDist(q^+, x^+) = \sum_{i=d^-+1}^{d} \begin{cases} 0, \text{ if } q_i \in \left[c_{x_i}^{min}, c_{x_i}^{max}\right] \\ \min\left(|q_i - c_{x_i}^{min}|, |q_i - c_{x_i}^{max}|\right)^2 \text{ otherwise} \end{cases}, \tag{7.10}$$

where $c_{x_i}^{max}$ ($c_{x_i}^{min}$) denotes the maximum (minimum) distance to the subcube c that contains the feature vector x in dimension i.

7.3.3 MBR Caching

In high-dimensional datasets that do not cluster strongly, the majority of the 2^d subcubes is occupied by at most one feature vector. In the few cases that a subcube is occupied by more feature vectors, a solution is to tighten the distance approximation of the subcubes using MBRs (cf. Figure 7.1(c)). Therefore, for all subcubes which are occupied by more than one feature vector, the MBR for the according set of feature vectors is computed. This MBR is stored in a priority queue (PQ) which is sorted in descending order w.r.t. the score function

$$f(\text{MBR}) = \frac{V_{\text{subcube}} \cdot card(\text{MBR})}{V_{\text{MBR}}}, \tag{7.11}$$

where $card$(MBR) denotes the number of feature vectors contained in the according MBR and $V_{subcube}$ (V_{MBR}) denotes the volume of the subcube (MBR).

As the resulting MBRs cannot be derived from any fixed values similarly to the case of the split positions, at least two d-dimensional coordinates are required to define each MBR, so that each MBR requires $d \cdot 16$ bytes (again assuming 8 byte double coordinates). Even though this seems to be a quite large overhead, an MBR can be shared among all feature vectors of the respective set. Thus, the memory increase is reduced to $\frac{d \cdot 16}{card(\text{MBR})}$ per feature vector comprised by the MBR. As the MBR is associated with the respective Z-ID, not even an additional memory pointer is required for the feature vector.

In order to define an upper limit for this additional memory consumption, the size of the MBR queue PQ is limited to 1% of the total number of feature vectors in the database. Combined with the score function of Equation (7.11), it is ensured that only a limited number of MBRs is held, where each MBR contains a large amount of feature vectors on the one hand and also covers a significantly smaller volume than the surrounding subcube on the other hand. This threshold has to be chosen as a trade-off between pruning power and additional memory consumption. Alternatively, the threshold can also be chosen in absolute values if the additionally required amount of memory shall be limited. In any case, the threshold should be chosen low enough, so that either all MBRs can be kept in memory or it should be ensured that only those MBRs are read from disk that approximate a fairly large number of feature vectors, so that the time needed to load the MBRs is still smaller than resolving the respective feature vectors.

In order to use the tighter approximation provided by the MBRs, the variables $c_{x_i}^{min}$ and $c_{x_i}^{max}$ in Equations (7.9) and (7.10) need to be filled with the coordinates of the MBR instead with those of the subcube, so that this second extension integrates seamlessly into the computation of the distance approximations.

7.4 Experimental Evaluation

7.4.1 Datasets and Experimental Setup

In order to measure the impact of the VA-file approach and the MBR caching w.r.t. the pruning power, the following tests were performed. First, both distance approximations of the original implementation of *BOND* were evaluated using the simple distance approximations (in the following denoted as **Bond**) and the improved approximations utilizing the specific resolve order w.r.t. the query vector (**Bond+**). Then, the contribution of the VA-file approach was evaluated by measuring the pruning power of a one- and a two-level VA-file (**Beyond-1**, **Beyond-2**). Finally, the additional impact by adding the MBR caching instead of an additional split was tested with **BeyondMBR-1**. The approaches were evaluated on three datasets (summarized in Table 7.1):

- *ALOI-27*: 27-dimensional reduced versions of originally 216-dimensional Zipfian distributed color histograms extracted from the *Amsterdam Library of Object Images* [99] comprising 110,250 feature vectors. This dataset poses the hardest challenge, as *BOND* is expected to perform best on this dataset as the color histograms follow a Zipfian distribution.

Table 7.1: Datasets used in the evaluation.

Name	Rows	Dimensions	Type
ALOI-27	110,250	27	color hisograms, Zipfian
CLUSTERED-50	300,000	20	synthetic, uniform, 50 Gaussian clusters
PHOG	10,715	110	gradient histograms

- *CLUSTERED-50*: Synthetic dataset of 300,000 20-dimensional feature vectors, organized in 50 clusters. The means of the clusters are uniformly distributed in the data space. Each cluster follows a multivariate Gaussian distribution.
- *PHOG*: 10,715 feature vectors derived from pyramid histograms of oriented gradients (HoG Features) with 110 dimensions. The features were provided from the work of [87] and represent gradient histograms which were extracted from medical computer tomography images. The features were already reduced in dimensionality by applying a Principal Component Analysis (PCA) [116] and the dimensions are ordered by decreasing value of the eigenvalues.

7.4.2 Pruning Power Evaluation

In the experiments, 50 k-NN queries ($k = 10$) were submitted to the database and the number of feature vectors was measured that were pruned after a data column was resolved and the distance approximations were recomputed. The charts in Figures 7.2, 7.3 and 7.4 represent the averaged cumulative amount of feature vectors that were pruned after a column was resolved; the x-axis counts the number of resolved dimensions, whereas the achieved pruning power is marked on the y-axis. The areas under the curves can, thus, be regarded as the amount of data that does not need to be resolved from disk, whereas the areas above the curves indicate the amount of data that needs to be taken into account for further refinement from disk and for the computation of the distance approximations. This observation can be regarded as a simple visual proof that tighter approximations yield higher pruning power, as more feature vectors can be pruned at a very early stage of the computation, so that further data columns of this feature vector do not have to be resolved. In the ideal case, only a few columns have to be resolved until the k final nearest neighbors remain in the dataset.

Comparing *ALOI-27* with the other datasets, it can be observed that **Bond+** performs as expected on Zipfian distributed histogram-like datasets. Nevertheless, **Bond+** resolves about half of the data on the *CLUSTERED-50* dataset and almost all columns on *PHOG*. This behavior proves the strong dependence on the data distribution, which is addressed in this chapter.

In the first improvement step of *BeyOND* (cf. Subsection 7.3.2), the approach presented in this chapter proposed to divide the feature space into subcubes and, thus, to refine the simple Euclidean distance approximation **Bond**. The gain of pruning power on is the *ALOI-27* dataset is clearly visible in Figure 7.2. Nevertheless, **Bond+** still achieves higher

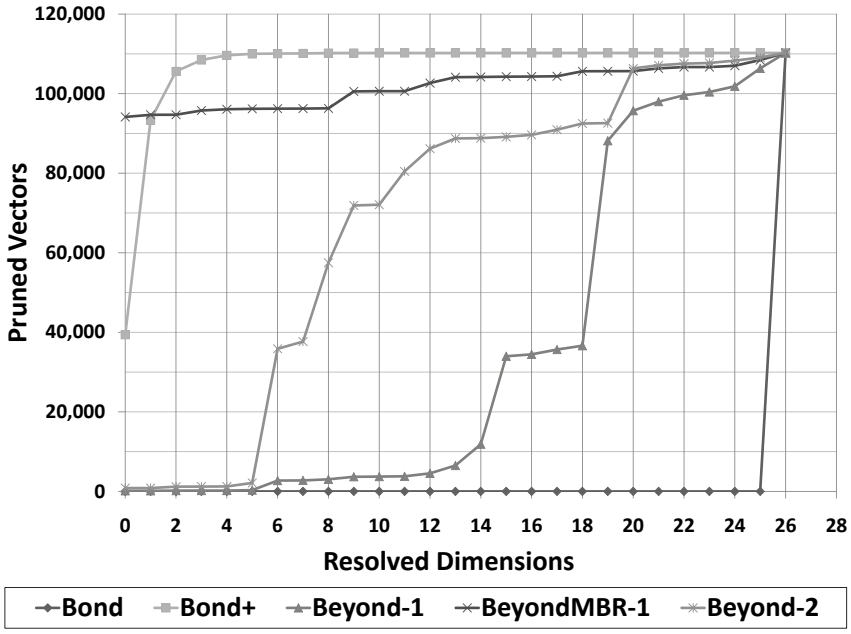


Figure 7.2: Pruning power on *ALOI-27*.

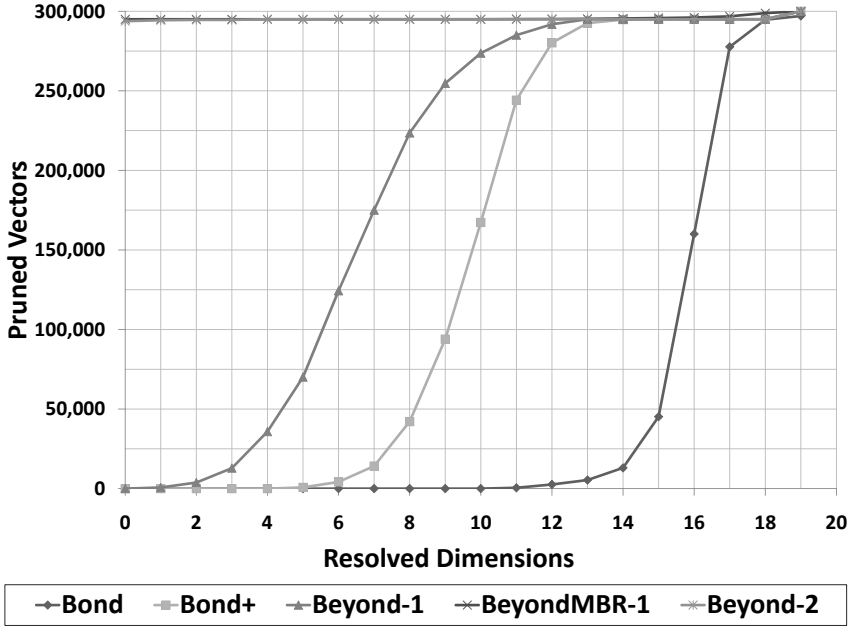


Figure 7.3: Pruning power on *CLUSTERED-50*.

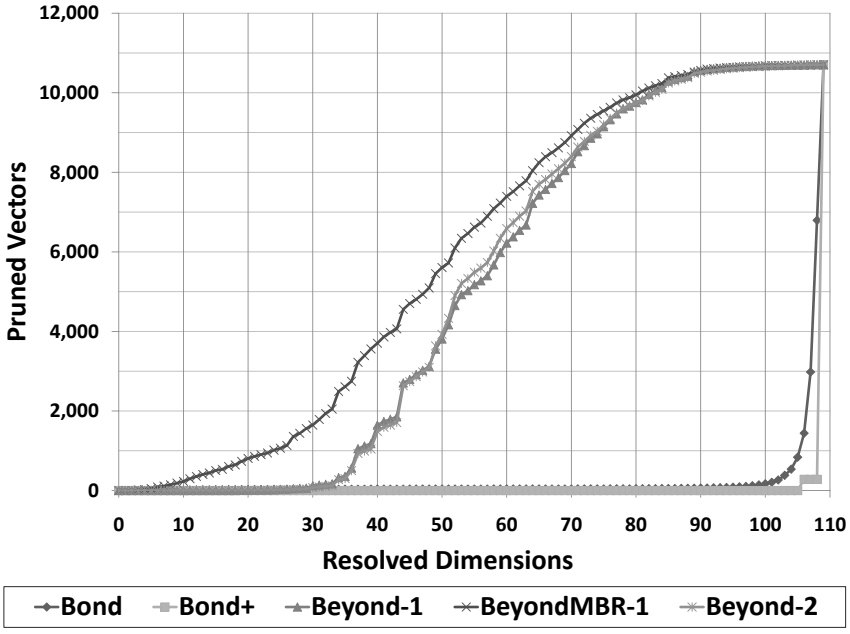


Figure 7.4: Pruning power on *PHOG*.

pruning power. On the contrary, on *CLUSTERED-50*, **Beyond-1** outperforms **Bond+**, as the subcubes provide good approximations of the clusters. These approximations are, in this case, superior to the bounds of **Bond+**, since the underlying distribution of the data is not Zipfian (cf. Figure 7.3). Finally, the impact of **Beyond-1** on *PHOG* is significantly higher than the impact of both **Bond** and **Bond+**, which do not perform on PCA'ed data at all and achieve a comparably high pruning power while resolving the last dimensions.

Table 7.2 provides "snapshots" of the pruning power curves depicted in Figures 7.2, 7.3 and 7.4. The columns show the amount of resolved columns (in percent), where more than 25%, 50% and 90% of the candidates were pruned. The observations from rows 1-3 support the above statements; the best results for **Beyond-1** are achieved on the *CLUSTERED-50* dataset, where only half the number of dimensions have to be resolved to achieve a pruning power of 90%.

The intuitive approach to add more splits per dimension (**Beyond-2**) and, thus, to decrease the size of the subcubes performs well on *ALOI-27* and *CLUSTERED-50*. This can be observed with the curve for the approach in Figures 7.2 and 7.3 and also from the rows 4-6 of Table 7.2. In particular the *CLUSTERED-50* dataset takes most advantage from the quadratic growth of additional subcubes ($2^d \rightarrow 4^d$), which poses a very good approximation of the clusters. Figure 7.4 shows that the improvement on *PHOG* is negligible.

The second improvement with *BeyOND* precomputes the MBRs in the case a subcube contains more than a single feature vector, the MBR would be small enough and the maximum number of MBRs is not reached yet (cf. Subsection 7.3.3). In this case, the portion of created MBRs that yield the largest volume decrease within the respective subcube w.r.t. the score function $f(\text{MBR})$ was limited to 1% of the overall number of

Table 7.2: Pruning power of **Beyond-1**, **Beyond-2** and **BeyondMBR-1**.

Dataset	Approach	25%	50%	90%
ALOI-27	**Beyond-1**	16 (59%)	19 (70%)	23 (85%)
CLUSTERED-50	**Beyond-1**	7 (35%)	8 (40%)	10 (50%)
PHOG	**Beyond-1**	45 (41%)	58 (53%)	80 (73%)
ALOI-27	**Beyond-2**	7 (26%)	9 (33%)	21 (75%)
CLUSTERED-50	**Beyond-2**	1 (5%)	1 (5%)	1 (5%)
PHOG	**Beyond-2**	45 (41%)	55 (50%)	79 (72%)
ALOI-27	**BeyondMBR-1**	1 (4%)	1 (4%)	10 (37%)
CLUSTERED-50	**BeyondMBR-1**	1 (5%)	1 (5%)	1 (5%)
PHOG	**BeyondMBR-1**	37 (34%)	50 (45%)	77 (70%)

Table 7.3: Total amount of data viewed with the different approaches.

Dataset	**Bond**	**Bond+**	**Beyond-1**	**Beyond-2**	**BeyondMBR-1**
ALOI-27	96.3%	3.2%	66.9%	38.3%	7.7%
CLUSTERED-50	81.6%	51.4%	36.3%	1.6%	1.4%
PHOG	97.6%	99%	52.6%	52.3%	45.4%

feature vectors. The results are shown with the approach **BeyondMBR-1**, which denotes the MBR-based variant of **Beyond-1**. Here again, the results can be observed from Figures 7.2, 7.3 and 7.4, where **BeyondMBR-1** is indicated by the dotted line, and in rows 7-9 of Table 7.2. On the *ALOI-27* dataset, the initial pruning power in the first dimension is even comparable to **Bond+**. On *CLUSTERED-50*, **BeyondMBR-1** yields comparable results to **Beyond-2**. Here, 98% of the data could be pruned at once. *PHOG* again poses the hardest challenge due to its very high dimensionality. However, there is a slight improvement compared to the basic subcube approaches **Beyond-1** and **Beyond-2**.

7.4.3 Additional Splits vs. MBRs

Table 7.3 shows the total amount of resolved data, which is computed by

$$r = \frac{\sum_{i=1}^{d} (\#\text{ resolved vectors}) \cdot i}{(\#\text{ vectors}) \cdot d - k}.$$

It can be observed that in case of *ALOI-27* and *PHOG*, it is more profitable to extend the original idea of *BOND* with a 1-level VA-file and additional MBRs (**BeyondMBR-1**) instead of simply adding more layers (**Beyond-2**) which generates more subcubes. On the *CLUSTERED-50* dataset, there is almost no difference between **BeyondMBR-1** and **Beyond-2**. Nevertheless, the solution with **BeyondMBR-1** offers more flexibility regarding the choice of MBRs and the control of additional memory consumption than simply increasing the split level.

Overall, it can clearly be observed that on *ALOI-27*, **Bond+** outperforms the other approaches significantly. This was expected due to the distribution of the dataset. However, the MBR Caching with **BeyondMBR-1** achieves almost similar pruning power. On *CLUSTERED-50*, the improvements of the *BeyOND* approaches are significant, in particular with one more split (**Beyond-2**) or MBR Caching (**BeyondMBR-1**). Finally, *PHOG* is a hard challenge for both **Bond** and **Bond+**, whereas **Beyond-1** provides reasonably tight bounds with one split of the data cube. This result, however, can hardly be further improved.

7.5 Summary

This chapter addressed the variant of indexing the full-dimensional space to enhance k-NN queries and extended a technique for high-dimensional feature spaces based on vertically decomposed data, known as *BOND*. The proposed techniques support the vertical decomposition and a better approximation of vectors in the high-dimensional feature space, while they do not depend on a particular distribution of the data. Combining the techniques of the partitioning the data space, as performed by the VA-file, and tightening the distance bounds using MBRs, the resulting approach achieves superior performance to prior work.

Chapter 8

Enhancing Similarity Processing in Arbitrary Subspaces

8.1 Introduction

Chapter 7 introduced a sequential-scan-based technique that enhances efficient query processing in high-dimensional feature spaces. There, the acceleration of similarity queries w.r.t. the full-dimensional case was considered. In some cases, the similarity of objects is based on a subset of attributes only, which leads to the problem of similarity search in subspaces. This problem has, for example, been explicitly addressed in [136, 156]. These approaches, however, are also based on the sequential scan and, in the case of subspace queries, lacking conditions for efficient processing. The solutions that will be provided in this chapter facilitate efficient *subspace similarity search* in large and potentially high-dimensional datasets where the user or the application can define an interesting subspace for each query independently (that is, similarity is defined ad hoc based on an arbitrary subset of attributes only). To this end, two index-based approaches will be proposed in this chapter to support subspace similarity search. As an example, applying subspace similarity to the activity recognition problem of Chapter 6, the proposed solutions can enhance activity recognition algorithms for the detection of new movement patterns in case the user is aware of a meaningful and relevant collection of features. Then, the feature selection step applying forward-backward search can be omitted in order to reduce the computational effort of the process chain, as further similarity processing can be based on relevant subspaces only by using the techniques proposed in this chapter.

The remainder of this chapter is organized as follows. The problem of subspace similarity search will be formally defined in Section 8.2. Then, Section 8.3 will propose an index-based bottom-up solution using the ideas of ranking and top-k retrieval. A second approach working in a top-down manner as an adaptation of the R-tree will be proposed in Section 8.4. An experimental evaluation of the introduced methods including a general and theoretical comparison will be presented in Section 8.5. Section 8.6 will finally conclude this chapter.

8.2 Subspace Similarity Search (SSS)

A common restriction for the small number of approaches tackling subspace similarity search (cf. Chapter 4) is that L_p-norms are assumed as distance measures. Hence, the problem definition of this chapter also relies on this restriction. In the following, $\mathcal{D}$ denotes a database of N objects in a d-dimensional space $\mathbb{R}^d$, and the distance between points in $\mathcal{D}$ is measured by a distance function $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_0^+$ which is one of the L_p-norms ($p \in [1, \infty)$). In order to perform subspace similarity search, a d_S-dimensional query subspace will be represented by a d-dimensional bit vector S of weights, where d_S weights are 1 and the remaining $d - d_S$ weights are 0; formally:

Definition 8.1 (Subspace) *A* subspace *S of a d-dimensional data space is represented by a vector $S = (S_1, \ldots, S_d) \in \{0, 1\}^d$, where $S_i = 1$, if the ith attribute is an element of the subspace, and $S_i = 0$, otherwise. The number d_S of 1-entries in S, i.e., $d_S = \sum_{i=1}^{d} S_i$ is called the* dimensionality *of S.*

For example, in a three-dimensional data space, the subspace representing the projection on the first and third axis is represented by $S = (1, 0, 1)$, having a dimensionality of 2.

A distance measure for a subspace S can then be figured as a weighted L_p-norm, where the weights can either be 1 (if this particular attribute is relevant to the query) or 0 (if this attribute is irrelevant), formally:

Definition 8.2 (Subspace Distance) *The distance in a subspace S between two points $x, y \in \mathcal{D}$ is given by*

$$dist_S(x, y) = \sqrt[p]{\sum_{i=1}^{d} S_i \, |x_i - y_i|^p}, \tag{8.1}$$

where x_i, y_i, and S_i denote the values of the ith component of the vectors x, y, and S, respectively.

Accordingly, a subspace ε-range query can be formalized as:

Definition 8.3 (Subspace ε-Range Query) *Given a query object q and a d_S-dimensional query subspace ($d_S \leq d$) represented by a corresponding vector S of weights, a subspace ε-range query retrieves the set $RQ(\varepsilon, S, q)$ that contains all objects from $\mathcal{D}$ for which the following condition holds:*

$$\forall x \in RQ(\varepsilon, S, q) : dist_S(x, q) \leq \varepsilon. \tag{8.2}$$

The related problem of subspace k-nearest neighbor (k-NN) queries can be formally defined as follows.

Definition 8.4 (Subspace k-NN Query) *Given a query object q and a d_S-dimensional query subspace ($d_S \leq d$) represented by a corresponding vector S of weights, a subspace k-NN query retrieves the set $NN(k, S, q)$ that contains k objects from $\mathcal{D}$ for which the following condition holds:*

$$\forall x \in NN(k, S, q), \forall y \in \mathcal{D} \setminus NN(k, S, q) : dist_S(x, q) \leq dist_S(y, q). \tag{8.3}$$

Some of the rare existing approaches for subspace similarity search focus on ε-range queries, which is a considerable lack (cf. Chapter 1). This is even more evident when searching subspaces of different dimensionality, because in this case, also the value of ε needs to be adjusted to the subspace dimensionality in order to produce meaningful results. This is a non-trivial task even for expert users, since recall and precision of an ε-sphere becomes highly sensitive to even small changes of ε depending on the dimensionality of the data space. Also, many applications like data mining algorithms that further process the results of subspace similarity queries require to control the cardinality of such query results [137]. Therefore, the approaches that will be introduced in this chapter will focus on k-NN queries.

8.3 Index-Based SSS – Bottom-Up

8.3.1 The Dimension-Merge Index

The solution to subspace similarity search that will be proposed in this section (referred to as *Dimension-Merge Index*) is based on the ad hoc combination of one-dimensional index structures. The combination technique is algorithmically inspired by top-k queries on a number of different rankings of objects according to different criteria. In the current scenario, if the objects are assumed to be ranked w.r.t. the distance to the query object in each dimension, respectively, it is possible to apply top-k methods to solve subspace k-NN queries with the rankings of the given subspace dimensions.

8.3.2 Data Structures

The key idea is to vertically decompose the data contained in $\mathcal{D}$ for the organization of full-dimensional space, as also performed in Chapter 7. For the restriction to subspaces, each dimension is organized separately in an index $\mathcal{I}_i$ ($1 \leq i \leq d$), using the feature value of the dimension as spatial key and the ID of the corresponding object as value. For this purpose, a B^+-tree seems adequate, as it is specialized for indexing one-dimensional data. In this problem, however, it is even more appropriate to use a (one-dimensional) R^*-tree [23], as it heuristically tries to minimize the extension of nodes, which was shown to be important for spatial queries. The used R^*-tree has the following two modifications:

- Each leaf node has a link to its left and right neighbor. This relation is well defined since the tree only organizes a single dimension on which a canonical order is defined.
- Each leaf node stores the values of the facing boundaries of its two neighbors.

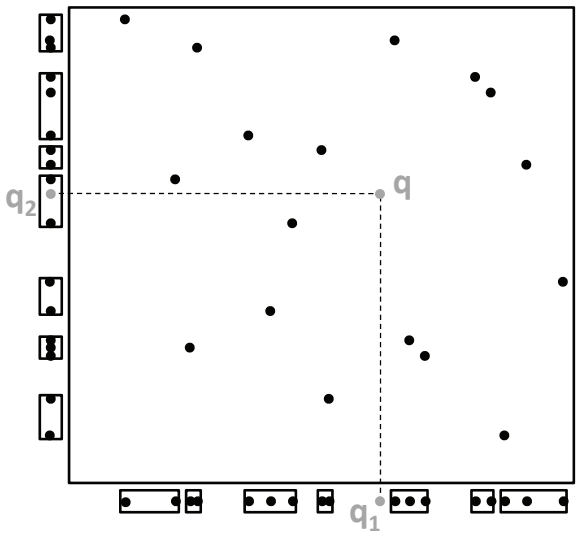


Figure 8.1: Initial situation of the data space.

objectTable				
Object	$dist_{\mathcal{I}_1}$	$dist_{\mathcal{I}_2}$	$minDist_S$	$maxDist_S$

Index Bounds		
Index	$\mathcal{I}^{min}$	$\mathcal{I}^{max}$
$\mathcal{I}_1$	0.3	9.0
$\mathcal{I}_2$	0.0	8.5
L_2	0.3	12.4

Table 8.1: Initial situation of indexes and *objectTable*.

The second data structure needed is a hash table for storing (possibly incomplete) object information with the object ID as key. This table will be referred to as *objectTable*. It is used to store, for each object, the distance to the query object in each dimension. If this information is not known, the corresponding field remains empty. In Figure 8.1 and Table 8.1, an example for a two-dimensional subspace query is shown. In the example, the leaf nodes (pages) of the two relevant index structures $\mathcal{I}_1$ and $\mathcal{I}_2$ organizing the objects in the dimensions of the subspace are illustrated at the borders of the data space. Initially, the *objectTable* is empty. Along the fields for distance values for each dimension in the *objectTable*, the values for lower and upper bounds can be computed, using the current information of the index bounds for the one-dimensional indexes $\mathcal{I}_1$ and $\mathcal{I}_2$. The computation of these bounds will be detailed in the following.

8.3.3 Query Processing

When a subspace query (q, S) arrives, only those indexes $\mathcal{I}_i$ are considered where $S_i = 1$. On these one-dimensional indexes, incremental NN queries (where q_i is the query for $\mathcal{I}_i$) are performed. A call of the function getNext() on the index $\mathcal{I}_i$ returns the leaf node closest to the query q_i in dimension i, whose contained objects have not yet been reported. The challenge is to combine the results of the single dimensions to a result on the whole subspace. This is done by the *objectTable*, which is empty at the beginning of the query process. For each object x, which was reported by an index $\mathcal{I}_i$, an entry in the *objectTable* is created. If it already exists, the corresponding entry is updated (i.e., the distance w.r.t. dimension i of object x is set to $dist_{\mathcal{I}_i}(q_i, x_i)$, where the distance is restricted to the subspace corresponding to the current dimension i). If an object x has not yet been seen in index $\mathcal{I}_j$ $(j \neq i)$, its value in dimension j in the *objectTable* is undefined. The distance $dist_S(q, x)$

Algorithm 5 k-NN Query on Dimension-Merge Index: kNN-DMI(q, S, k, $\mathcal{I}$)

Require: q, S, k, $\mathcal{I}$
1: $maxKnnDist \leftarrow \infty$
2: **while** $maxKnnDist \geq minObjectDist_S(q, \mathcal{I})$ **do**
3: $i \leftarrow$ chooseIndex($\mathcal{I}$)
4: $leafNode \leftarrow \mathcal{I}_i$.getNextNode($q_i$)
5: $objectTable$.insert($leafNode.elements$)
6: $maxKnnDist \leftarrow objectTable$.getMaxKnnDist($k$)
7: **end while**
8: $objectTable$.refine()

between an object $x \in \mathcal{D}$ and q in the subspace S can be upper bounded by

$$maxDist_S(q,x) = \sqrt[p]{\sum_{i=1}^{d} S_i \cdot \begin{cases} |x_i - q_i|^p & \text{(Case 1)} \\ \max(|\mathcal{I}_i^{min} - q_i|, |\mathcal{I}_i^{max} - q_i|)^p & \text{(Case 2)} \end{cases}}, \tag{8.4}$$

where $\mathcal{I}_i^{min}$ and $\mathcal{I}_i^{max}$ are the lower and upper bound of the data contained in $\mathcal{D}$ in dimension i, respectively. These bounds can be obtained directly from the index $\mathcal{I}_i$, as this corresponds to the boundaries of the root node. Obviously, it holds that $maxDist_S(q,x) \geq dist_S(q,x)$. For the calculation of $maxDist_S(q,x)$, two cases have to be considered: if object x has been found in index $\mathcal{I}_i$ (Case 1), the exact value in this dimension can be used. Otherwise, the bounds of the data space have to be used in order to approximate the value in this dimension (Case 2). Using Equation (8.4) and the information contained in the *objectTable*, an upper bound for the distance $dist_S(q,x)$ can be obtained for each object $x \in \mathcal{D}$. Therefore, it is also possible to calculate an upper bound for the distance of the kth-nearest neighbor to the query object, which can be used as pruning distance. The upper bound is recorded in the *objectTable*, and updated if necessary.

Analogously, a lower bound for each object in the *objectTable* can be obtained by

$$minDist_S(q,x) = \sqrt[p]{\sum_{i=1}^{d} S_i \cdot \begin{cases} |x_i - q_i|^p & \text{(Case 1)} \\ |\mathcal{I}_i^{next} - q_i|^p & \text{(Case 2)} \end{cases}}, \tag{8.5}$$

where $\mathcal{I}_i^{next}$ is the position of the query-facing boundary of the page obtained by the next call of the function getNext() on $\mathcal{I}_i$. Again, it is necessary distinguish the cases where x_i has been reported (Case 1) or where it is undefined at the moment (Case 2). This lower bound is important for the refinement step of the query algorithm and it is recorded in the *objectTable*, and updated if necessary.

The pseudocode for a subspace k-NN query on the Dimension-Merge Index is given in Algorithm 5. Initially, the upper bound of the kth-nearest neighbor distance (*maxKnnDist*) is set to infinity. As long as there exists an object which could have a lower distance than the current *maxKnnDist* and which is not in the *objectTable*, the filter step has to be

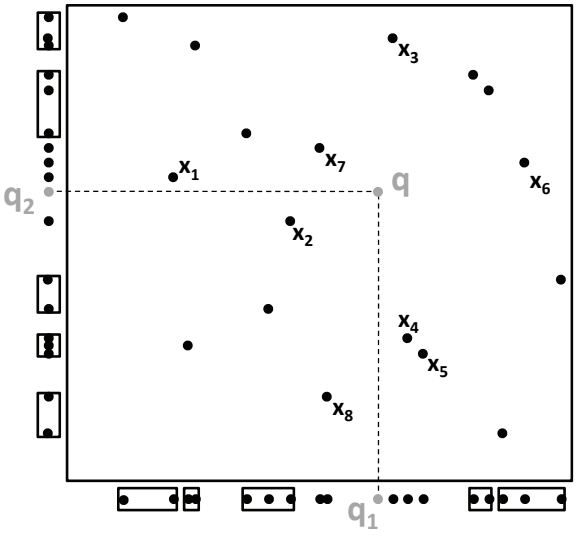


Figure 8.2: Situation of the data space after four getNext()-calls.

objectTable				
Object	$dist_{\mathcal{I}_1}$	$dist_{\mathcal{I}_2}$	$minDist_S$	$maxDist_S$
x_1		0.5	2.94	9.01
x_2		1.0	3.06	9.06
x_3	0.4		1.94	8.51
x_4	1.0		2.15	8.56
x_5	1.5		2.42	8.63
x_6		1.0	3.07	9.06
x_7	1.9	1.5	2.42	**2.42**
x_8	1.7		2.55	8.67

Index Bounds		
Index	$\mathcal{I}^{min}$	$\mathcal{I}^{max}$
$\mathcal{I}_1$	2.9	9.0
$\mathcal{I}_2$	1.9	8.5
L_2	**3.47**	12.4

Table 8.2: Situation of indexes and *objectTable* after four getNext()-calls.

continued and, thus, more points have to be inserted into the *objectTable*. The minimum distance of an object which is not in the *objectTable* is given by

$$minObjectDist_S(q, \mathcal{I}) = \sqrt[p]{\sum_{i=1}^{d} S_i \cdot |\mathcal{I}_i^{next} - q_i|^p}. \tag{8.6}$$

At the point where *minObjectDist* is larger than the *maxKnnDist* (as seen in Figure 8.2 and Table 8.2 for $k = 1$), the algorithm enters the refinement step. The objects were retrieved in ascending order of their indices. In the current state, *minObjectDist* exceeds $maxKnnDist = maxDist_S(q, x_7)$ for the first time. Now, no object which is not in the *objectTable* can be part of the result, therefore only objects contained in the *objectTable* at this time have to be considered. In order to keep the number of resolved objects (corresponding to the number of expensive page accesses) low, the technique for refinement from optimal multi-step processing, proposed in [135], is used.

Algorithm 5 can easily be adapted to ε-range queries. Only the *maxKnnDist* has to be set to ε and does not have to be updated (i.e., line 6 is to be omitted).

8.3.4 Index Selection Heuristics

The most important part of the algorithm considering the performance is the *chooseIndex* method (line 3). For a fast termination of the filter step it is necessary to

- find and minimize the upper bound of *maxKnnDist* and
- increase the minimum distance a page can have

as fast as possible. This section proposes three heuristics for choosing the appropriate index in each step.

The first heuristic (*Round Robin*) sequentially chooses the index in a round robin fashion and can be seen as a simple baseline. The problem with this heuristic is that it does not take the data distribution into account. Thus, it does not meet the two requirements for a fast processing of the filter step (see above).

The second heuristic, called *GlobalMinDist*-heuristic, aims at the first point: it always chooses the index $\mathcal{I}_i$ which has the closest page to the query q_i considering dimension i. As will be shown in the experimental evaluation, this heuristic yields a superior performance of the query processing. However, the *GlobalMinDist*-heuristic will perform very bad in a subspace where one dimension has a much larger scale than the other dimensions. In this setting the *GlobalMinDist*-heuristic will prefer resolving pages from the indexes organizing the dimensions with a small extent, as in these dimensions, the *minDist* from the query will be very low compared to the dimensions with a higher extent. Thus, the second requirement is not met, and many pages get resolved without much information gain.

To overcome this drawback, a third heuristic is proposed, which will be referred to as *MinScore*-heuristic. For each index $\mathcal{I}_i$, the score

$$f_{score}(\mathcal{I}_i) = \frac{|q_i - \mathcal{I}_i^{next}|}{\mathcal{I}_i^{max} - \mathcal{I}_i^{min}} \tag{8.7}$$

is computed and the index minimizing $f_{score}(\mathcal{I}_i)$ is chosen. This normalization prevents the algorithm from preferring dimensions with small extent.

8.4 Index-Based SSS – Top-Down

8.4.1 The Projected R-Tree

This section proposes the *Projected R-Tree*, a redefinition of the R-tree to answer subspace queries. Though, the provided solution can be integrated into any hierarchical index structure and is not necessarily restricted to R-trees.

The idea of the top-down approach is to apply one index on the full-dimensional data space. The key issue is that, for a subspace similarity query, the minimum distance between an index page P and the query object q in subspace S has to be properly defined because then, it is possible to just use the best-first search algorithm [107] without any changes. The minimum distance between an index page P and the query object q in the subspace S can be computed as

$$minDist_S(q,P) = \sqrt[p]{\sum_{i=1}^{d} S_i \cdot \begin{cases} |P_i^{min} - q_i|^p & \text{if} \quad P_i^{min} > q_i \\ |q_i - P_i^{max}|^p & \text{if} \quad P_i^{max} < q_i \\ 0 & \text{otherwise} \end{cases}}, \tag{8.8}$$

where P_i^{min} and P_i^{max} are the lower and upper bound of the page P in the ith dimension,

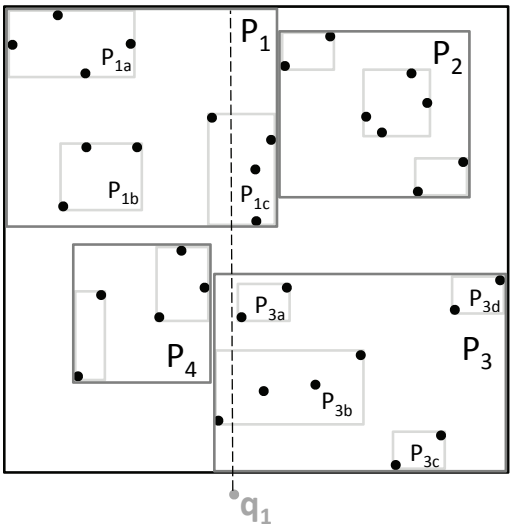


Figure 8.3: One-dimensional subspace query on a two-dimensional space using a Projected R-Tree.

Iteration	*APL*	*maxKnnDist*
0	(0.0,*root*)	∞
1	(0.0,P_1), (0.0,P_3), (0.7,P_4), (1.6,P_2)	∞
2	(0.0,P_{1c}), (0.0,P_3), (0.7,P_4), (1.6,P_2), (3.0,P_{1b}), (3.2,P_{1a})	∞
3	(0.0,P_3), (0.7,P_4), (1.6,P_2), (3.0,P_{1b}), (3.2,P_{1a})	0.8
4	(0.0,P_{3b}), (0.2,P_{3a}), (0.7,P_4), (1.6,P_2), (3.0,P_{1b}), (3.2,P_{1a})	0.8
5	(0.2,P_{3a}), (0.7,P_4), (1.6,P_2), (3.0,P_{1b}), (3.2,P_{1a})	0.5
6	(0.7,P_4), (1.6,P_2), (3.0,P_{1b}), (3.2,P_{1a})	0.2

Table 8.3: Processing of a sample query.

respectively. Equation (8.8) is designed for the rectangular page region of R-trees. For the implementation here, an R^*-tree has been used as underlying tree index.

8.4.2 Query Processing

When a query q arrives, it is processed in a best-first manner, as proposed in the well-known best-first k-NN search algorithm [107]. The algorithm maintains an active page list (APL) which contains pages of the index structure ordered ascending by their *minDist* to the query. Since a subspace query is processed, only the dimensions defined by the query are taken into account for the calculation of the *minDist*. The algorithm starts inserting the root of the index structure into the APL. In each iteration, the first page from the APL is processed. If it is a directory node, its children are inserted into the APL. If it is a leaf node, the distance of each point contained in the page to the query is computed. Each point may therefore update the *maxKnnDist*, which is the distance of the kth-nearest point found so far. The process stops if the *minDist* of the first entry of the APL is larger than the *maxKnnDist*. In this case, none of the pages in the APL can contain an object being part of the k-nearest neighbors of the query. Figure 8.3 and Table 8.3 illustrate an example of a subspace query.

8.4.3 Discussion

The Projected R-Tree is a relatively straightforward adaptation and can be regarded as complementary to the Dimension-Merge Index. In contrast to the Dimension-Merge Index using one index for each dimension, the Projected R-Tree just needs one index applied to

Table 8.4: Datasets used in the evaluation.

Name	Rows	Dimensions	Type
WEATHER	1,128,186	9	meteorological data
ALOI-8/ALOI-64	110,250	8/64	color histograms, Zipfian
UNIFORM	100,000	20	synthetic, uniform
CLUSTERED-1000	100,000	20	synthetic, uniform, 1,000 Gaussian clusters

the full-dimensional data space. As a result, the Projected R-Tree does not need to merge the partial results of the rankings performed for each dimension in the corresponding subspace. Relying on the full-dimensional indexing of a dataset, the Projected R-Tree can be expected to perform superior to the Dimension-Merge Index, in the cases where the dimensionality of the query subspace is approaching the dimensionality of the dataset (if the latter does not disqualify methods based on full-dimensional indexing). On the other hand, as the index used in the Projected R-Tree organizes the data w.r.t. the full-dimensional space, the locality property of a similarity query which might hold for the full-dimensional space does not necessarily hold for the subspace the query relates on. Generally, the more the dimensionality of the original data space differs from that of the query subspace, the smaller is the expected effectiveness of the index for a given subspace query. In summary, depending on the dimensionality of the query subspace, both indexing approaches qualify for subspace similarity search. While the Dimension-Merge Index is more appropriate for lower-dimensional subspace queries, the Projected R-Tree should be used when the dimensionality of the query subspace approaches that of the data space. This statement will be supported by the experimental evaluation, which will be summarized in the following section.

8.5 Experimental Evaluation

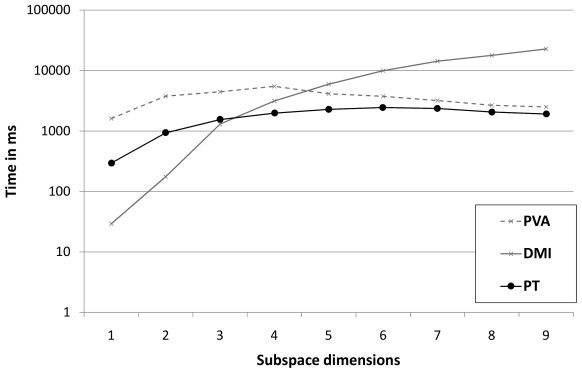
8.5.1 Datasets and Experimental Setup

The approaches were evaluated on three datasets (summarized in Table 8.4):

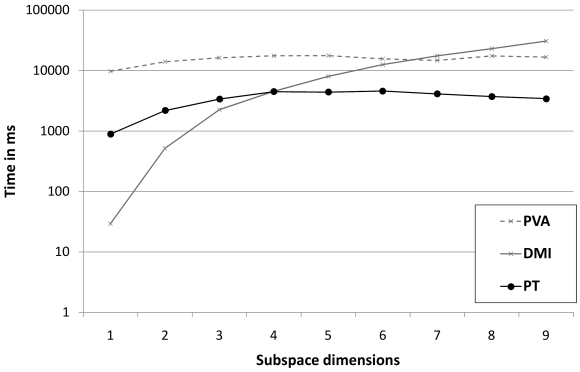
- *WEATHER*: 9-dimensional meteorological dataset, consisting of 1,128,186 feature vectors, used in [27].

- *ALOI-8/ALOI-64*: 8- and 64-dimensional reduced versions of originally 216-dimensional Zipfian distributed color histograms extracted from the *Amsterdam Library of Object Images* [99] comprising 110,250 feature vectors.

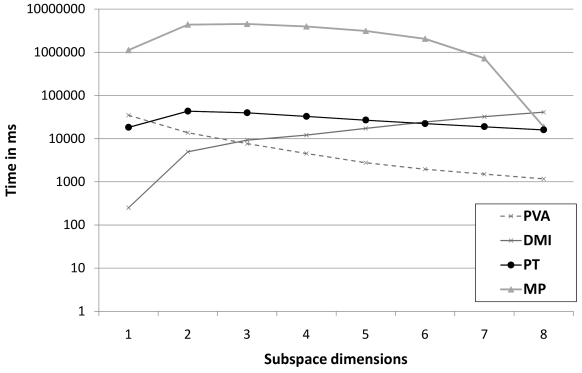
- *UNIFORM*: Synthetic dataset of 100,000 20-dimensional feature vectors which are uniformly distributed in the data space.



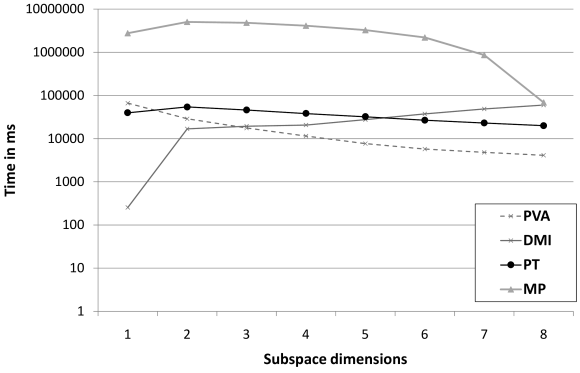
(a) 1-NN queries on *WEATHER*.



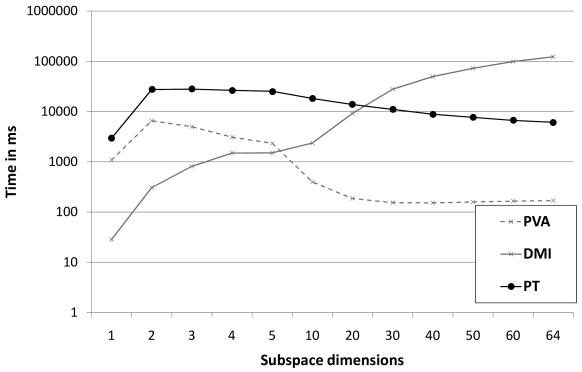
(b) 10-NN queries on *WEATHER*.



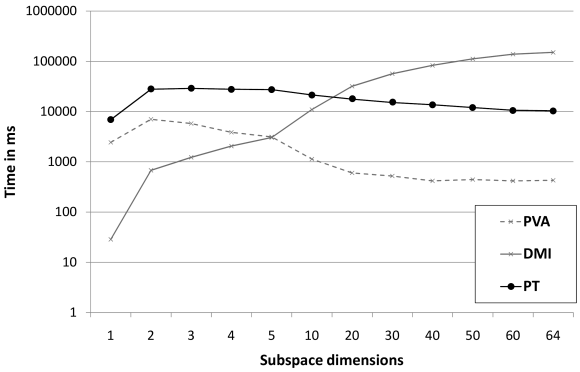
(c) 1-NN queries on *ALOI-8*.



(d) 10-NN queries on *ALOI-8*.



(e) 1-NN queries on *ALOI-64*.



(f) 10-NN queries on *ALOI-64*.

Figure 8.4: Queries with different subspace dimensions on the real-world datasets.

- *CLUSTERED-1000*: Synthetic dataset of 100,000 20-dimensional feature vectors, organized in 1,000 clusters. The means of the clusters are uniformly distributed in the data space. Each cluster follows a multivariate Gaussian distribution.

This section evaluates the methods proposed in this chapter based on the experiments from [32]. In particular, Subsection 8.5.2 compares the different algorithms for subspace indexing on real-world datasets, whereas Subsection 8.5.3 focuses on the performance of the different heuristics for the Dimension-Merge Index on synthetic datasets having different characteristics. The used datasets are summarized in Table 8.4 and in more detail described as follows:

8.5.2 Evaluation of Methods for Subspace Indexing

This section compares the approaches **DMI** (Dimension-Merge Index proposed in Section 8.3), **PT** (Projected R-Tree proposed in Section 8.4), **PVA** (Partial VA-file [136]) and **MP** (a variant of the multiivot-based algorithm [156] for k-NN queries) for subspace indexing. Unless stated otherwise, the different approaches on a dataset were compared with $k = 1$ and $k = 10$ with increasing subspace dimension displayed on the x-axis. Due to the long runtime and the high amount of disc accesses of **MP**, this approach was omitted with *WEATHER* and *ALOI-64*.

In order to compare the different approaches, between 1,000 and 10,000 k-NN queries were performed for each dataset. For **DMI**, **PT** and **MP**, all page accesses that could not be served by the LRU-cache were measured. For all experiments with the mentioned approaches, an infinite cache was assumed. **PVA** does not only perform random page accesses for reading data, but also implements a heuristic that switches to a block read of pages if it turns out that multiple subsequent pages have to be read. Therefore, block read pages and randomly accessed pages of **PVA** were measured separately. In order to make the results of **PVA** comparable to the results of the other approaches, the number of block read pages was combined with the number of randomly accessed pages an estimated read time was calculated. To achieve this, a seek time of 8 ms and a transfer rate of 60 MB/s were assumed.

Figure 8.4 compares the proposed methods on the real-world datasets. For different values of k on *WEATHER* (cf. Figures 8.4(a) and 8.4(b)), it can be observed clearly that **DMI** is superior or equal to the other approaches up to a subspace size of four dimensions. For *ALOI-8* (cf. Figures 8.4(c) and 8.4(d)), using **DMI** is more appropriate for a subspace size of up to three dimensions. On *ALOI-64* (cf. Figures 8.4(e) and 8.4(f)), **DMI** outperforms **PVA** and **PT** for up to four subspace dimensions until it reaches a break-even point with **PVA** at a subspace size of five dimensions. Regarding the dimensionality of the dataset and the subspace dimensions where **DMI** performs better than one of the other methods (three on *ALOI-8*, four on *WEATHER* and five on *ALOI-64*), it can be stated that **DMI** – such as **PVA** – is more effective on datasets with higher dimensionality, depending on the parameter k (exemplarily shown in Figure 8.5 for *ALOI-64*). The obtained results confirm the discussion from Subsection 8.4.3. In all tested settings, **DMI** performs best as

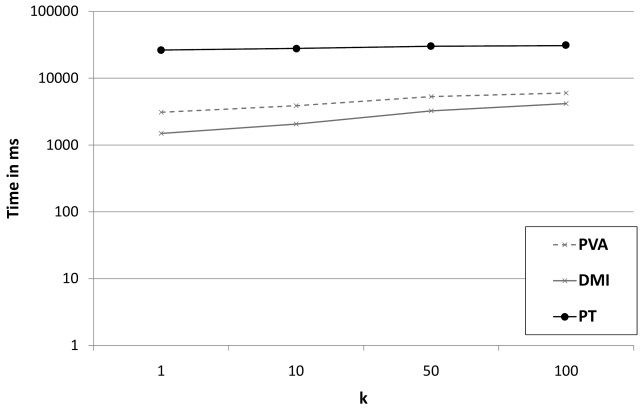


Figure 8.5: k-NN queries on *ALOI-64* ($d_S = 4$, increasing k).

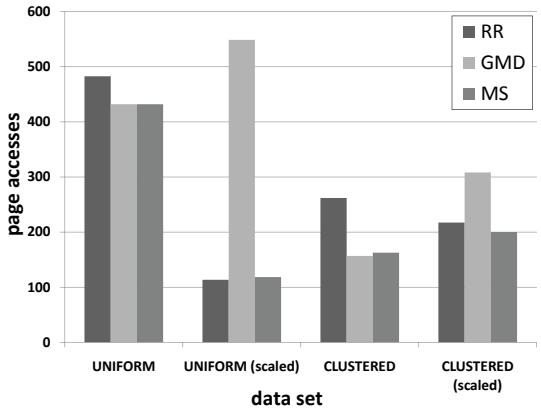


Figure 8.6: Heuristics on datasets with different characteristics.

long as the dimensionality of the subspace query is moderate. When the dimensionality increases, **PT** becomes superior to **DMI**. **PVA** is a scan-based approach and well suited if a dataset is hard to index (e.g. very high-dimensional). *WEATHER* seems to be well indexable by the projected variant of the R^*-tree, therefore **PT** is superior to **PVA**. The *ALOI-8*/*ALOI-64* datasets in contrast are rather hard to index (in particular *ALOI-64* having a very high dimensionality).

8.5.3 Evaluation of the Heuristics

The proposed heuristics for the Dimension-Merge Index (cf. Section 8.3) address different problems of the data distribution. To accurately show their behavior, the heuristics Round-Robin (**RR**), Global-MinDist (**GMD**) and MinScore (**MS**) were tested on synthetic datasets with different characteristics. 1,000 10-NN queries on a three-dimensional subspace were performed, measuring the page accesses needed by each dimension using the **DMI** approach (again assuming an infinite LRU-cache) and averaging the outcomes. The results are illustrated in Figure 8.6. On *UNIFORM* and *CLUSTERED-1000*, the more sophisticated heuristics (**GMD** and **MS**) are superior to the naïve **RR** method, since they try to find a more suitable dimension instead of more or less randomly picking one. If the dimensions are scaled randomly, the **GMD** heuristics favors the dimension with the minimum scale factor. However, this dimension does only increase the minimum distance of all other objects by a small value. Therefore it can stop the filter step very late, which results in many unnecessary page accesses.

8.6 Summary

This chapter proposed and studied index-based solutions for supporting k-NN queries in arbitrary subspaces of a multidimensional feature space. Therefore, two different approaches

were studied. One of the main problems this chapter addressed was how to schedule the available information from the various dimensions in order to obtain good distance approximations of the objects for an early pruning of candidates. The evaluation showed that the proposed solutions perform superior to the competitors.

Part III

Key Property of *Uncertainty*: Uncertain Databases

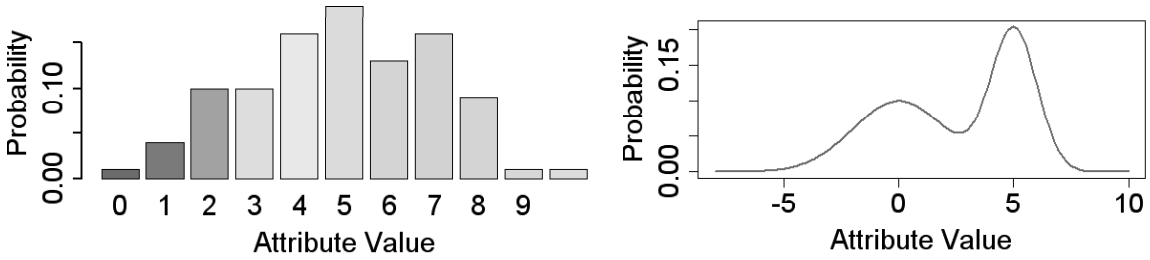
Chapter 9

Introduction

9.1 Preliminaries

In Part II, multi-observation data has been focused w.r.t. the key property of temporal variability. While, with the model of time series, the observations of an object occur consecutively in dependency of the temporal domain, an object incorporating the key property of uncertainty (cf. Chapter 1) is given by multiple states at the same time, modeled by a set of observations; this leads to the research direction of *uncertain* or *probabilistic data*. Here, the basic question arises which of these observations is most likely to represent this object. Materializing this likelihood, the observations are associated with probability values; this creates existential dependencies among the observations, as the existence of an observation affects the existence of the other observations of the same object.

Coping with data uncertainty initiates a need for developing suitable data models and techniques for searching and mining. The models commonly applied for probabilistic databases – and in particular for this work – will be presented in Section 9.2. Then, an introduction to probabilistic similarity queries will be given in Section 9.3, in particular to the interesting problem of probabilistic similarity ranking in spatially uncertain data (Section 9.4). The technique that will be used for efficiently solving the probabilistic ranking problem will prove to be applicable to further challenges that will be tackled, among them are the problem of probabilistic inverse ranking and the task of hot item detection. The final chapters of this part will go beyond the definition of multi-observation data, but remain in the area of uncertainty. There, the prominent problem of probabilistic frequent itemset mining will be addressed by applying techniques that emerged from the use in the context of probabilistic ranking. These problems will be motivated in Section 9.5.



(a) Discrete attribute uncertainty.

(b) Continuous attribute uncertainty.

Figure 9.1: Variants of attribute uncertainty.

9.2 Modeling Uncertain Data

9.2.1 Categorization

Uncertainty in databases can generally be incorporated as *tuple uncertainty* and *attribute uncertainty*. Assuming tuple uncertainty, tuples are associated with a probability to appear in the database. This characteristic is also called *existential uncertainty*. The property of *attribute uncertainty* implies that a tuple has at least one uncertain attribute where the possible values are contained within a defined range. In the literature, probabilistic data models are classified in two types w.r.t. the attribute uncertainty: the *discrete* uncertainty model (cf. Figure 9.1(a)) and the *continuous* uncertainty model (cf. Figure 9.1(b)).

In many real-world applications, uncertain objects are already given by discrete observations, in particular if the objects are derived from sensor signals. This type of representation is motivated by the fact that, in many cases, only discrete but ambiguous object information – as usually returned by common sensor devices – is available, e.g., discrete snapshots of continuously moving objects. Therefore, this part will focus on the discrete uncertainty model by adopting a prominent representative among the discrete uncertainty models. The *ULDB model* or *x-relation model* [25], introduced in the *Trio* system [8], will be presented in the following.

The continuous uncertainty model will not be relevant in the context of this work, but for the sake of completeness, it will briefly be explained within a summary of related work on uncertain data in Chapter 10.

9.2.2 The X-Relation Model

The x-relation model extends the relational database model by incorporating uncertainty and lineage [25] and it supports existential uncertainty and attribute uncertainty. Relations in the x-relation model are called *x-relations* and contain uncertain tuples with alternative instances, which are called *x-tuples*. Each x-tuple T corresponds to a set of tuples. Each tuple $t \in T$ is associated with a probability $P(t)$, denoting the likelihood that it exists in T. This characteristic realizes existential uncertainty of tuples. The probabilities represent a discrete probability distribution of T, which realizes the attribute uncertainty of T; the

(a) Tuples and x-tuples.

Tuples		
Tuple	Location	Prob.
t_1	Renzy's Den	50%
t_2	Waterhole	20%
t_3	Hunting Grounds	30%
t_4	Waterhole	10%
t_5	Hunting Grounds	10%
t_6	The Forest	20%

Tiger X-Relation	
Name	X-Tuple
Renzy	$\{t_1, t_2, t_3\}$
Unknown Tiger **?**	$\{t_4, t_5, t_6\}$

(b) Possible worlds.

Possible Worlds		
World	Tuples	Prob.
W_1	$\{t_1\}$,{}	30%
W_2	$\{t_1\}$, $\{t_4\}$	5%
W_3	$\{t_1\}$, $\{t_5\}$	5%
W_4	$\{t_1\}$, $\{t_6\}$	10%
W_5	$\{t_2\}$,{}	12%
W_6	$\{t_2\}$, $\{t_4\}$	2%
W_7	$\{t_2\}$, $\{t_5\}$	2%
W_8	$\{t_2\}$, $\{t_6\}$	4%
W_9	$\{t_3\}$,{}	18%
W_{10}	$\{t_3\}$, $\{t_4\}$	3%
W_{11}	$\{t_3\}$, $\{t_5\}$	3%
W_{12}	$\{t_3\}$, $\{t_6\}$	6%

Table 9.1: Tuples describing locations of tigers, an x-relation containing x-tuples with their possible locations and corresponding possible worlds with their probabilities.

constraint that $\sum_{t \in T} P(t) \leq 1$ holds. The condition $\sum_{t \in T} P(t) < 1$ implies existential uncertainty of x-tuples, meaning that the x-tuple may not exist at all.

9.2.3 The Possible Worlds Semantics

In relational databases, a popular semantics to cope with the uncertainty of data has been introduced by adopting Saul Kripke's *Possible Worlds Semantics* [145], e.g., as performed in [15]. Incorporating this semantics into the x-relation model, an uncertain database $\mathcal{D}$ is instantiated into a possible world as follows [44]:

Definition 9.1 (Possible Worlds) *Let $\mathcal{D} = \{T_1, \ldots, T_N\}$ be an uncertain database and let $W = \{t_1, \ldots, t_N\}$ be any (certain) database instance which corresponds to a subset of tuples t_i appearing in $\mathcal{D}$ such that $t_i \in T_i, i \in \{1, \ldots, N\}$. The probability of this database instance (world) W to occur is $P(W) = \prod_{i=1}^{N} P(t_i)$. If $P(W) > 0$, W is a* possible world*; the set of all possible worlds is denoted by $\mathcal{W}$.*

The x-relation model is a special type of the possible worlds model that additionally allows mutual independence among the x-tuples. Furthermore, the tuples t of an x-tuple T are assumed to be mutually exclusive, i.e., no more than one instance t of an x-tuple T can appear in a possible world instance at the same time. In the general model description, the possible worlds are constrained by rules that are defined on the tuples in order to incorporate correlations or dependencies between tuples [187].

The x-relation model will be used as a basic object model in the major part of the following chapters. To get an intuition of this model, an example, taken from [134], will be given below.

Example 9.1 *Table 9.1 shows an x-relation that contains information about the possible positions of tigers in a wildlife sanctuary. Here, the first x-tuple describes the tiger named "Renzy", who may be found at three possible (alternative) locations t_1, t_2 and t_3. He may be in his cave with a probability of 50% or located at the water hole and at the hunting grounds with a probability of 20% and 30%, respectively. This x-tuple logically yields three mutually exclusive, possible tuple instances, one for each alternative location. Now, we know that an unknown tiger may have entered the wildlife sanctuary with a probability of 40%. In this case, it is not certain that the unknown tiger exists at all, which is an existential uncertainty of the x-tuple, denoted by a "?" symbol [25]. To incorporate this existential uncertainty, an additional, "empty" tuple is inserted added to the x-tuple of the unknown tiger, such that the probabilities of the possible worlds can be computed according to Definition 9.1. Taking into account the four alternatives (including the alternative of no unknown tiger) for the position of the unknown tiger, there are twelve possible instances (worlds) of the tiger x-relation. In general, the possible worlds of an x-relation $\mathcal{R}$ correspond to all combinations of alternatives for the x-tuples in $\mathcal{R}$. In this model, the probability of the unknown tiger being at the water hole is not affected by the current position of Renzy, due to the independence assumption among x-tuples. Considering the general case with possible tuple dependencies, this example could be extended by the "natural" restriction that male tigers are territorial and the position of a tiger may be affected by the presence of other tigers in its close vicinity. Thus, for example, world W_6 might not occur by rule.*

9.2.4 Translation to Spatial Databases

For the use in this work, the semantics of the x-relation model will be translated into a spatial context. Then, an uncertain database $\mathcal{D}$ consists of N uncertain objects with spatial attributes, where each object X corresponds to an x-tuple and each observation $x \in X$ corresponds to a tuple. The attribute uncertainty of objects of a d-dimensional vector space $\mathbb{R}^d$ is called *positional uncertainty*. Then, objects do not have a unique position in $\mathbb{R}^d$, but have multiple positions associated with a probability value. Thereby, the probability value assigned to a position $x \in \mathbb{R}^d$ of an object X denotes the likelihood that X is located at the position x in the vector space. The existential dependency, which describes the rule that observations belonging to the same object are mutually exclusive, will be assumed for the rest of this part. This dependency realizes the main characteristic of uncertain data.

According to [43, 45], a formal definition of a positional uncertain object within a d-dimensional vector space is given as follows:

Definition 9.2 (Discrete Uncertain Object) *A* discrete uncertain object X *corresponds to a finite set of observations (alternative positions) in a d-dimensional vector space, each associated with a confidence value, i.e., $X = \{(x, P(X = x))\}$, where $x \in \mathbb{R}^d$, and $P(X = x) \in [0, 1]$ indicates the likelihood that object X is represented by observation x.*

Corresponding to the discrete uncertainty model, the probabilities of the observations represent a discrete probability distribution of the alternative positions, such that the condition $\sum_{(x,P(X=x))\in X} P(X = x) \leq 1$ *holds. The collection of observations of all objects forms the uncertain database* $\mathcal{D}$*.*

Analogously to the *x-relation model*, this definition also assumes independence among the uncertain objects as well as mutual exclusiveness among observations of the same object. This object definition will be used in Chapters 11 to 14. In the following, a *positionally uncertain object* will be called *uncertain object* for simplicity and realizes the key property of uncertainty, introduced in Chapter 1. Furthermore, *existentially uncertain objects* will not be considered; it is assumed that $\sum_{(x,P(X=x))\in X} P(X = x) = 1$. In the special case where $m = 1$, this corresponds to a single-observation (in this part also called *certain*) object.

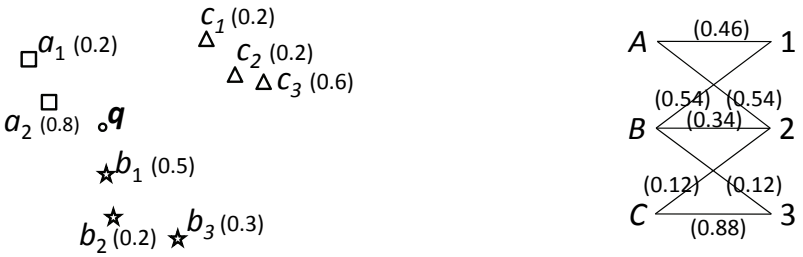
The data model for uncertain transactions applied with probabilistic frequent itemset mining in Chapters 15 and 16 is based on the plain x-relation model and allows the presence of existentially *uncertain items*.

9.3 Probabilistic Similarity Queries

The simplest solution to perform queries on uncertain objects is to represent the objects by an exact object, e.g., the mean of all observations, and perform query processing in a traditional way. The advantage of this straightforward solution is that established similarity processing techniques can be applied. However, this solution is accompanied by information loss, since the similarity between uncertain objects is obviously more meaningful when taking the whole information of the object uncertainty into account. For probabilistic data, special formulations of queries are required in order to take the uncertainty of the data into account, especially if the data attributes, e.g., the locations in the case for moving objects in spatial databases, are changing continuously. Thus, contrary to the single-observation (certain) case, the output of probabilistic similarity queries is usually in form of a set of result objects, each associated with a probability value indicating the likelihood that the object satisfies the query predicate.

Basically, any query predicate that is applicable for single-observation (certain) data is extendible to uncertain data, e.g., *probabilistic ε-range (PεR) queries* [112, 138], *probabilistic k-nearest neighbor (Pk-NN) queries* [162] and *probabilistic reverse k-nearest neighbor (PRk-NN) queries* [37]. The main challenge for Pk-NN and PRk-NN queries is that the neighborhood probability of objects depends on the location of the other objects in the database. Result outputs may be restricted to objects with a sufficiently high result probability. For this purpose, the *probabilistic threshold k-NN query* [75] applies a probability threshold to control a minimum probability for objects to appear in the result set.

Similarly to the single-observation case, a *probabilistic similarity ranking query* ranks database objects in descending order w.r.t. their similarity to a query object. Due to the significance of similarity ranking queries (cf. Chapter 1), the first chapters of this part



(a) Observations. (b) Bipartite Graph.

Figure 9.2: Observations and rank probability graph [43].

will focus on probabilistic ranking queries, which will be discussed in more detail in the following.

9.4 Probabilistic Similarity Ranking

9.4.1 Ranking Semantics

In contrast to PεR queries and Pk-NN queries, probabilistic ranking queries do not have a unique query predicate, since the query predicate changes with each ranking position. In case of a probabilistic ranking query, a set of probability values is assigned to each result object, one for each ranking position. This output will be called *Rank Probability Distribution* (*RPD*) in the following. A formal definition will follow in Chapter 11. Commonly, a small part of the RPD should be sufficient and, thus, more convenient for most applications. Therefore, analogously to the single-observation case, an output limitation is commonly used by the *ranking depth* k.

Due to the variance of the ranking positions of the objects, there does not exist a unique order in which the results are reported. The following example illustrates the problem of the ambiguity of probabilistic ranking.

Example 9.2 *Consider, for example, a set of three two-dimensional objects A, B, and C (e.g., locations of mobile users), and their corresponding uncertain positions $\{a_1, a_2\}$, $\{b_1, b_2, b_3\}$, and $\{c_1, c_2, c_3\}$, as shown in Figure 9.2(a). Each observation carries a probability (shown in brackets); observations of the same object are assumed to be mutually exclusive. According to Definition 9.2, the sum of observation probabilities of each object does not exceed 1. Assume that the objects A, B, and C shall be ranked w.r.t. their distances to the query observation q shown in the figure, where, for example, any L_p-norm can be applied. Clearly, several rankings are possible. In specific, each combination of observations defines an order. For example, for the combination $\{a_1, b_1, c_1\}$ the object ranking is (B, A, C), while for the combination $\{a_2, b_3, c_1\}$ the object ranking is (A, B, C). Each combination corresponds to a possible world (cf. Definition 9.1), whose probability can be*

computed by multiplying the probabilities of the observations that comprise it, assuming independent existence probabilities between the observations of different objects.

However, not all of this information might be of interest for the user and it could be difficult to extract the relevant information. Therefore, most applications require the definition of an unambiguous ranking where each object (or observation) is uniquely assigned to one rank. 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 the robbery took place. The probabilities of these observations depend on various factors (e.g., the time difference of the observation to the robbery event, errors of capturing devices, etc.). As an application, it may be the demand to define a definite probabilistic proximity ordering of the suspects to the event, in order to prioritize interrogations.

9.4.2 This Work in the Context of Probabilistic Ranking

In the first unambiguous probabilistic ranking solution for the x-relation model, Soliman et al. presented in [192] a top-k query processing algorithm for uncertain data in relational databases according to the x-relation model (cf. Subsection 9.2.2). The authors proposed two different variants of uncertain top-k queries: *U-Topk* and *U-kRanks*, where the general qualifying criterion of a ranking or score function f_{score} is used. In probabilistic databases, f_{score} is created by the interaction of the score attributes and the probability of a tuple [111].

In the U-Top$k$ query, the objective is to find the top-k in each possible world for a given score function (Step 1) and then report the result that has the highest aggregated probability over all possible worlds (Step 2). The first step requires a prior score ordering of the tuples in each possible world, such that U-Top$k$ can be classified as an approach of probabilistic ranking. However, analogously to the single-observation case, the result is comparable to the Pk-NN query in spatial data.

In contrast, the U-kRanks query reports a probabilistic ranking of the tuples (i.e., for each ranking position i one tuple t, which is a clear winner on rank i). For this computation, obvious requirements are the probabilities of t to be ranked ith in 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 Chapters 11 and 12, the problem definition of computing the probabilistic ranking corresponds to the U-kRanks query problem. Also, Chapter 11 will include an example for the U-kRanks semantics.

In [192], an efficient solution for U-kRanks is only given for the case where all tuples are mutually independent, which does not hold for the x-relation model (as there is a dependency among all tuples of an x-tuple). Instead, all possible worlds would have to be enumerated. 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 tuples.

Chapter 11 will present the first probabilistic similarity ranking algorithm based on the full probabilistic information given by the inexact object representations according to

Definition 9.2, which incorporates the application of the x-relation model to spatial data. Here, the score function f_{score} is defined by the distance to the query object (i.e., a high score value is reflected by a low uncertain distance value). A divide-and-conquer-based probabilistic ranking algorithm will be introduced within a framework, which significantly reduces the complexity of the computation of the rank probabilities.

The main drawback of U-kRanks is that this query type only considers the probability of an object to be ranked on a particular ranking position. The confidences of prior ranking positions of an object are ignored in the case they are exceeded by another object. However, the latter confidences might also be relevant for the final setting of the ranking position of an object. Therefore, diverse variants of probabilistic ranking schemes will be studied in Chapter 11, which differ in the order in which the results are reported and in the form their confidence values are aggregated.

In order to be independent of any probabilistic ranking output, the result provided by the algorithm of Chapter 11 will be a object-rank bipartite graph containing the probabilities for all objects to be on each ranking position, which corresponds to an RPD (cf. Subsection 9.4.1). Going back to Example 9.2, Figure 9.2(b) illustrates such a bipartite graph. For example, the probability that object A occurs on rank 1 is 0.46 and the probability that object B is first is 0.54. Non-existing edges imply a probability of 0, i.e., it is not possible that the object occurs at the corresponding ranking position. In this example, all observations of A precede all those of C w.r.t. the distance to q, so C cannot occur as first object and A cannot be ranked on the last position.

Despite of the improvement of the computational effort for solving U-kRanks by the divide-and-conquer-based solution of Chapter 11, the overall runtime to obtain the RPD is still exponential in the number of uncertain objects. An improved method based on the x-relation model was given for relational data by Yi et al. in [214]. There, it has been shown that the U-kRanks query result can be computed in polynomial time by introducing a dynamic-programming technique for probabilistic ranking which is known as *Poisson Binomial Recurrence* (*PBR*) [147]. A second approach that will be given in Chapter 11 already exploits this technique and applies it to the spatial object model according to Definition 9.2.

Chapter 12 [43] will introduce a method that achieves a significant improvement of this approach, which further reduces the computational complexity to linear time in the number of uncertain objects for particular assumptions while having the same memory requirements. Furthermore, it will be shown how the output of the proposed method (i.e., the RPD, which corresponds to the output of the solution of Chapter 11) can also be applied to state-of-the-art top-k definitions that generate unambiguous ranking outputs based on the x-relation model. More details of these definitions will be given with related work in Chapter 10 and with the solutions in Chapter 12.

The essential module of the framework of Chapters 11 and 12 is the computation of an observation-based RPD. This requires the computation of the probability for an observation x that i objects are closer to the query observation q than x, for all ranks $i \in \{1, \ldots, k\}$. The RPD is dynamically computed by performing a linear scan over the ordered observations. The resulting probabilities are aggregated to build the probability

of each object at each rank.

In contrast to [192, 214], where the ranking results are built upon a tuple-based ranking, the objective in Chapters 11 and 12 is to return a ranking of uncertain objects, which correspond to the x-tuples in the x-relation model. It will, therefore, also be shown that the cost required to solve the object-based RPD is similar to that required to solve the observation-based RPD. The temporary solution based on observations additionally only requires to build the sum over all observation-based rank probabilities, which can be done on-the-fly without additional cost.

Finally, the cost required to build a final unambiguous ranking result (e.g., U-Top$k$, U-kRanks) from the RPD can be neglected. The unambiguous ranking can also be computed on-the-fly by simple aggregations of the corresponding RPD.

The PBR can be applied to solutions for a variety of problems in probabilistic databases, which will be motivated in the following section. Its incremental processing scheme allows the introduction of a solution of quadratic runtime with linear update costs for the problem of continuous probabilistic inverse ranking (Subsection 9.4.3). Furthermore, data mining applications, in this context the problem of hot item detection and the prominent task of probabilistic frequent itemset mining benefit from the dynamic-programming approach (Section 9.5).

9.4.3 Probabilistic Inverse Ranking

Contrary to the "common" probabilistic ranking query, which reports objects in ascending order w.r.t. their distance to a query object and assigns them to each rank with a probability, a *probabilistic inverse ranking query* monitors a given query object and retrieves all possible ranks of this object according to a given, user-defined score function. A motivating example, taken from [152], is the following.

Example 9.3 *For a newborn, it may be of interest to get information about his or her health in comparison with other babies in terms of height, weight, and so on. In this case, it is possible to infer the newborn's health from his or her rank in comparison with others. Data of newborn babies in a hospital are confidential. Thus, for the sake of privacy-preservation, such information is usually perturbed by adding synthetic noise or generalized by replacing exact values with uncertain intervals, before releasing them for research purposes. Thus, in these situations, a probabilistic inverse ranking query can be conducted over uncertain data (perturbed or generalized) in order to obtain all possible ranks that a newborn may have with high confidence.*

While the probabilistic inverse ranking problem has been tackled for static data [149, 158], Chapter 13 will apply the incremental approach of updating the probability distribution of Chapter 12 on uncertain stream data, i.e., when the data changes with elapsing time. Here, an uncertain stream is a stream of observations with confidences, e.g., observed positions of moving objects derived from multiple sensors. Corresponding to the definition of uncertain objects (cf. Definition 9.2), each observation carries a confidence value which reflects the likelihood that the observation conforms with the current true object state.

In Chapter 13, a framework will be presented that updates the inverse ranking query result very efficiently, as the stream provides new observations of the objects. It will be theoretically and experimentally shown that the query update can be performed in linear time, corresponding to the time complexity of the incremental method presented in Chapter 12.

9.5 Probabilistic Data Mining

9.5.1 Hot Item Detection in Uncertain Data

The detection of objects which build dense regions with other objects within a vector space is a foundation of several density-based data mining techniques, in particular density-based clustering [90, 184], outlier detection and other density-based mining applications [61, 143, 194]. A (certain) object x for which exists a sufficiently large population of other objects in a database $\mathcal{D}$ that are similar to x is called a *hot item*. Intuitively, an item that shares its attributes with many other items could be potentially interesting, as its shows a typical occurrence of items in the database. Application areas where the detection of hot items is potentially important for example include the detection of "hot" research topics, the detection of interesting products for online shopping advertising or the pre-detection of criminal activities.

Chapter 14 [48] will give provide more details for these examples and will then propose an approach for the detection of potentially interesting objects (hot items) of an uncertain database in a probabilistic way. An algorithm for arbitrary probabilistic predicates, again based on the PBR, will be presented which detects hot items, where, to each object x, a confidence value is assigned that reflects the likelihood that x is a hot item. In the context of Chapter 14, hot items can be abstracted to objects that satisfy a given similarity predicate together with a reasonably large set of other items. For example, if the *equality* predicate is assumed, then a hot item satisfies the *frequent item* property, as this item is equal to many other items and, thus, occurs frequently in the database.

9.5.2 Probabilistic Frequent Itemset Mining

The final chapters of this part will go beyond the definition of spatially uncertain objects, but remain in the area of uncertainty. There are various data mining applications that have to cope with the presence of uncertainty. Then, techniques designed for similarity query processing can apply in order to obtain effective and efficient solutions. For example, association rule analysis is one of the most important fields in data mining. It is commonly applied to market-basket databases for the analysis of consumer purchasing behavior. Such databases consist of a set of transactions, each containing the items a customer purchased. The most important and computationally intensive step in the mining process is the extraction of *frequent itemsets* – sets of items that occur in a minimum number of transactions. It is generally assumed that the items occurring in a transaction are known for certain.

However, also in transaction databases, this is not always the case – as already outlined in Part I –, due to several reasons:

- In many applications, the data is inherently noisy, such as data collected by sensors or in satellite images.
- In privacy protection applications, artificial noise can be added deliberately [210]. Finding patterns despite this noise is a challenging problem.
- By aggregating transactions by customer, it is possible to mine patterns across customers instead of transactions. This produces estimated purchase probabilities per item per customer rather than certain items per transaction.

Probabilistic frequent itemset mining in uncertain transaction databases semantically and computationally differs from traditional techniques applied to standard (certain) transaction databases. The consideration of existential uncertainty of item(sets), indicating the probability that an item(set) occurs in a transaction, makes traditional techniques inapplicable. Chapter 15 [46] will introduce probabilistic formulations of frequent itemsets based on the possible worlds semantics (cf. Definition 9.1), allowing existentially uncertain items. In this probabilistic context, an itemset X is called frequent if the probability that X occurs in a minimum number of transactions is above a given threshold. The presented solution has been the first approach addressing this problem under the possible worlds semantics. In consideration of the probabilistic formulations, a framework will be presented which solves the probabilistic frequent itemset mining problem efficiently, applying the PBR.

As a follow-up work of Chapter 15, Chapter 16 [47] will propose the first *Probabilistic Frequent Pattern Growth* (*ProFP-Growth*) algorithm. Here, an approach based on the FP-tree [104] will be used in order to mine all probabilistic frequent itemsets in uncertain transaction databases without candidate generation in a probabilistic way, thus providing even a faster and more memory-efficient solution. The method to compute the support probability distribution will apply the concept of *Generating Functions*, as proposed in the context of probabilistic ranking in [154]. This concept has similar objectives to the PBR w.r.t. dynamic-programming techniques and incremental processing.

Chapter 10

Related Work

10.1 Categorization

Existing approaches in the field dealing with uncertain data can be categorized into diverse directions. Work on uncertain data models will be reviewed in Section 10.2. Related work in the context of probabilistic query processing and mining on uncertain data will be summarized in Sections 10.3 and 10.4.

10.2 Modeling and Managing Uncertain Data

Uncertain data models are classified in *discrete* and *continuous* uncertainty models. Discrete models regard the composition of databases with uncertain tuples [84], where each tuple is associated with a probability denoting the likelihood that it exists in the relation. This model adopts the *Possible Worlds Semantics* [145] (cf. Definition 9.1 of Chapter 9). The *x-relation model* proposed in [8, 25] supports uncertain tuples with alternative instances which are assumed to be mutually exclusive, i.e., no more than one instance of an uncertain tuple can appear in a possible world instance at the same time. The discrete uncertainty model has been adopted by many approaches dealing with probabilistic data, e.g., [108, 114, 154, 191].

Applications where there exists at least one uncertain attribute which is assumed to follow a continuous probability density function (PDF) have to cope with the continuous uncertainty model.

Following the convention of uncertain databases with continuous uncertainty, the multidimensional PDF f_X of an object X is (minimally) bounded by an *uncertainty region* R_X, such that $\forall x \notin R_X : f_X(x) = 0$ and $\int_{R_X} f_X(x)dx \leq 1$, where, similarly to the discrete uncertainty model, the case $\int_{R_X} f_X(x)dx < 1$ implies the presence of existential uncertainty. The assumption of a bounded PDF is realistic, because the spectrum of possible values of attributes is usually bounded and it is commonly used in related work, e.g., [50, 68, 74, 76, 78]. Even if f_X is given as an unbounded PDF, e.g., a Gaussian PDF, a common strategy is to truncate PDF tails with negligible probabilities and normalize the

resulting PDF. In specific, [50] shows that, for a reasonable low truncation threshold, the impact on the accuracy of probabilistic ranking queries is quite low while having a very high impact on the query performance.

Methods for similarity processing that are based on the continuous uncertainty model involve expensive integrations of the PDFs. In order to reduce the high computational effort, the generally applicable concept of Monte Carlo sampling is used, which generates the set of (discrete) observations according to a given PDF [114]. Thus, this concept allows the application of methods originally designed for discrete uncertainty models.

A popular solution in the field of spatial data is to approximate the uncertainty region by MBRs, which allows the application of spatial pruning methods, e.g., [34]. Other works apply hypersphere approximations, e.g., [155, 195].

Popular indexing methods to enhance probabilistic query processing comprise the *Gauss-tree* [54] for Gaussian distributed PDFs, the *U-tree* [195] for arbitrary distributions, the *UI-tree* [220] and the *APLA-tree* [162], which uses piecewise-linear approximations.

10.3 Probabilistic Query Processing

10.3.1 Probabilistic Similarity Ranking

In the context of probabilistic ranking, significant work has been done in the field of probabilistic top-k retrieval yielding unambiguous rankings from probabilistic data. A detailed summary of the most approaches can be found in [114].

[55] applies the Gauss-tree [54] in order to incrementally retrieve those k objects which have a sufficiently high probability of being located inside a given query area.

Probabilistic top-k queries have been studied first by Soliman et al. [192] on the x-relation model. The authors propose two different ways of ranking tuples: the *uncertain top-k query* (*U-Topk*) and the *uncertain k-ranks query* (*U-kRanks*). At the same time, Ré et al. proposed in [177] an efficient but approximate probabilistic ranking based on the concept of Monte Carlo simulation.

The approach proposed in [214] was the first efficient exact probabilistic ranking approach for the x-relation model. The results for U-Top$k$ and U-kRanks are computed by means of a dynamic-programming technique, known as *Poisson Binomial Recurrence* (*PBR*) [147] and early stopping conditions for accessing the tuples. The work proposed in Chapters 11 and 12 uses this technique as a module of computing the object-rank probabilities which can, among others, be used to solve the U-kRanks problem efficiently.

In [155], the *probabilistic ranked query* for the context of distributions over spatial data is based upon the same definition as U-kRanks.

The *Probabilistic Threshold Top-k* (*PT-k*) query problem [108] aggregates the probabilities of an observation appearing on rank k or higher. Given a user-specified probability threshold p, PT-k returns all observations that have a probability of at least p of being on rank k or higher. In this definition, the number of results is not limited by k, but depends on the threshold parameter p. This approach also utilizes the PBR.

The *Global top-k* approach [219] is very similar to PT-k. It ranks the observations by their top-k probability and then takes the top-k of these. The advantage here is that, unlike in PT-k, the number of results is fixed, and there is no user-specified threshold parameter that has to be set.

Cormode et al. [83] reviewed alternative top-k ranking approaches for uncertain data, including U-Top$k$ and U-kRanks, and argued for a more robust definition of ranking, namely the *expected rank* for each tuple (or x-tuple). The expected rank 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, it could be found by experimentation that such a definition may not be appropriate for ranking objects (i.e., x-tuples), whose observations have large variance (i.e., they are scattered far from each other in space). Therefore, a follow-up work [114] computes the *median rank* and the *quantile rank* in order to obtain more robust measures against outliers and high variances. These approaches run in loglinear time and, thus, outperform exact approaches that do not use any estimation. The main drawback of the approaches is that, by using an aggregated estimator, information is lost about the distribution of the objects. This is the reason why Chapters 11 and 12 focus on the computation of the RPD, at the and presenting a solution which also requires loglinear runtime complexity.

The goal of [191] is to rank uncertain objects (i.e., x-tuples) where the scores are 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 [191] has similar objectives to the approaches of Chapters 11 and 12, it operates on a different input, where the distribution of uncertain scores is already known, as opposed to the ranking approaches of this work, which dynamically computes this distribution by performing a linear scan over the ordered observations.

The work of [215] studies probabilistic ranking of objects according to their distance to a query point. However, the solutions are limited to existentially uncertain data with a single observation.

Related to probabilistic top-k queries, [180] introduced queries on uncertain data with aggregations, such as probabilistic count and probabilistic sum queries, where the number of tuples is determined that have a higher (uncertain) score than the current tuple. The consideration of all possible worlds is, however, again very inefficient. The authors of [96] apply the *continuous probabilistic count query* on wireless sensor network environments, which, in this context, reports the number of sensor nodes whose measured values satisfy a given query predicate. An efficient result computation is achieved by applying the PBR. The work [109] proposes the *continuous probabilistic sum query* in wireless sensor networks applying *Generating Functions*, which have been introduced by [154] while solving a wide class of probabilistic top-k queries in the same time complexity. This concept has similar objectives to the PBR w.r.t. dynamic-programming techniques and incremental processing and will also be used in this work, namely in the context of probabilistic frequent itemset mining in Chapter 15.

10.3.2 Probabilistic Inverse Ranking

In contrast to ranking in uncertain data, where there exists abundant work, there is limited research on the inverse variant of ranking uncertain data. The *inverse ranking query* on certain data was first introduced by Li [152]. Chen et al. [158] apply inverse ranking to probabilistic databases by introducing the *probabilistic inverse ranking (PIR) query*. According to [158], the output of a PIR query consists of all possible ranks for a (certain) query object q, for which q has a probability higher than a given threshold. Another approach for answering PIR queries has been proposed by [149], which computes the expected inverse rank of an object. The expected inverse rank can be computed very efficiently, however, it lacks from a semantic point of view. In particular, an object that has a very high chance to be on rank 1, may indeed have an expected rank far from rank 1, and may not be in the result using expected ranks. Thus, no conclusion can be made about the actual rank probabilities if the expected rank is used, since the expected rank is an aggregation that drops important information. The first exact PIR approach for continuously changing data in the context of observation streams will be presented in Chapter 13.

In order to deal with massive datasets that arrive online and have to be monitored, managed and mined in real time, the data stream model has become popular. Surveys of systems and algorithms for data stream management are given in [18, 168]. A generalized stream model, the probabilistic stream model, was introduced in [113]. In this model, each item of a stream represents a discrete probability distribution together with a probability that the element is actually present in the stream. There has been work of interest on clustering uncertain streams [7], as well as on processing more complex event queries over streams of uncertain data [178]. [82] presents algorithms that capture essential features of the stream, such as quantiles, heavy hitters, and frequency moments. In [115], the authors propose a framework for processing continuous top-k queries on uncertain streams.

10.3.3 Further Probabilistic Query Types

Beyond probabilistic ranking, there is a variety of work tackling other query types in uncertain data, including *probabilistic range queries*, *probabilistic nearest neighbor (PNN) queries* and some variants, and *probabilistic reverse nearest neighbor (PRNN) queries*. Probabilistic range queries have been addressed in [76, 78, 112, 138, 195].

There exist approaches for PNN queries based on certain query objects [77] and for uncertain queries [110, 139]. The authors of [74] add threshold constraints and propose the constrained PNN query for certain query points in order to retrieve only objects whose probability of being the nearest neighbor exceeds a user-specified threshold. A combination of the concepts of PNN queries and top-k retrieval in probabilistic databases is provided by top-k PNN queries [50]. Here, the idea is to return the k most probable result objects of being the nearest neighbor to a single-observation (certain) query point.

[162] proposed a solution for *probabilistic k-nearest neighbor (Pk-NN) queries* based on expected distances. [75] introduced the *probabilistic threshold k-NN (PTk-NN) query*, which requires an uncertain object to exceed a probability threshold of being part of the

Pk-NN result. Here, the query is assumed to be a single-observation object.

The framework that is proposed in [34] introduced the concept of *probabilistic domination* in order to efficiently answer Pk-NN and PTk-NN queries as well as probabilistic ranking and inverse ranking queries for uncertain query objects, applying *Uncertain Generating Functions*, an extended variant of the *Generating Functions* introduced in [154].

The PRNN query returns the set of all objects for which the probability that an uncertain query object Q is their nearest neighbor exceeds a user-defined probability threshold. This problem has been tackled by by [157] for the continuous model and by [67] for the discrete case. The work [37] showed to achieve superior results to the mentioned approaches; furthermore, an extension to PRk-NN queries is proposed.

10.4 Probabilistic Data Mining

The aspect of identifying hot items, i.e., objects that are similar to a given amount of other objects, is the basis of several density-based mining applications [61, 90, 143, 184, 194]. The detection of hot items can be efficiently supported by a similarity join query used in a preprocessing step, in particular the distance-range self-join. A survey of probabilistic join queries in uncertain databases can be found in [134]. Approaches for an efficient join are proposed in [138]. The main advantage of this approach is that discrete positions in space can efficiently be indexed using traditional spatial access methods, thus allowing to reduce the high computational cost to process complex query types.

Apart from the analysis of spatial objects, there are various data mining applications that have to cope with the presence of uncertainty. For example, the detection of frequent itemsets as a preprocessing step for rule mining is one of the most important problems in data mining. There is a large body of research on Frequent Itemset Mining (FIM), but very little work has recently been addressing FIM in uncertain databases [79, 80, 150]. The approach proposed in [80] computes the expected support of itemsets by summing all itemset probabilities in the *U-Apriori* algorithm. Later, in [79], they additionally proposed a probabilistic filter in order to prune candidates early. In [150], the *UF-growth* algorithm is proposed. Like U-Apriori, UF-growth computes frequent itemsets by means of the expected support, but it uses the FP-tree approach [104] in order to avoid expensive candidate generation. UF-growth considers itemsets to be frequent if the expected support exceeds a minimum support threshold. The main drawback of this estimator is that information about the uncertainty of the support is lost; [79, 80, 150] ignore the number of possible worlds in which an itemset is frequent. [217] proposes exact and sampling-based algorithms to find likely frequent items in streaming probabilistic data. However, they do not consider itemsets with more than one item. Finally, except for [199], existing FIM algorithms assume binary-valued items which precludes simple adaptation to uncertain databases. The approaches proposed in Chapters 15 [46] and 16 [47] have been the first methods that find frequent itemsets in an uncertain transaction database in a probabilistic way.

A different tree-based algorithm is presented in [151], which suggests an upper bound of the expected support by dealing with projected transactions.

Chapter 11

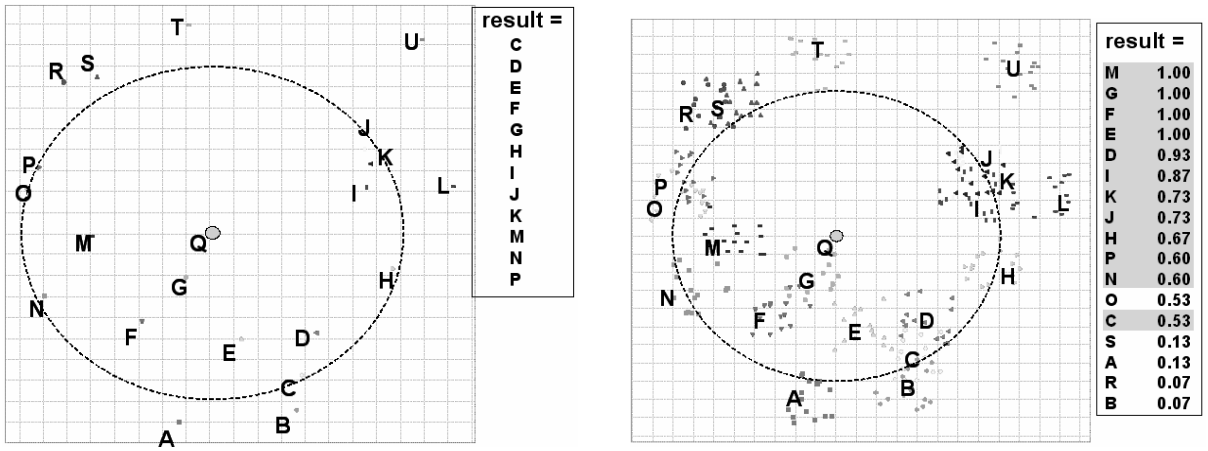
Probabilistic Similarity Ranking on Spatially Uncertain Data

11.1 Introduction

A probabilistic ranking query on uncertain objects computes for each object $X \in \mathcal{D}$ the probability that X is the ith nearest neighbor ($1 \leq i \leq |\mathcal{D}|$) of a given query object Q. The simplest solution to perform queries on spatially uncertain objects is to represent each object by exactly one observation, e.g., the mean vector, and perform query processing in a traditional way. The advantage of this straightforward solution is that established query and indexing techniques can be applied. However, this solution is accompanied by information loss, since the similarity between uncertain objects is obviously more meaningful when taking the full information of the object uncertainty into account. An example of the latter case is shown in Figure 11.1(a), which depicts uncertain objects $A, \ldots, U$, each represented by its mean value. The results of a distance range query w.r.t. the query object Q are shown in the upper right box. There are critical objects like P, which is included in the result, and O, which is not included, though they are very close to each other. The result based on the full probabilistic object representation is shown in Figure 11.1(b). Here, the gray shaded fields indicate those objects which are also included in the non-probabilistic result. Obviously, the objects O and P have quite similar probabilities ($P(O) = 53\%, P(P) = 60\%$) of belonging to the result. Additionally, it can be observed that the objects E, F, G and M are certain results, i.e., have a probability of 1 to appear in the result set.

This chapter will tackle the problem of similarity ranking on spatially uncertain data exploiting the full probabilistic information of uncertain objects. First, diverse forms of ranking outputs will be suggested which differ in the order the objects are reported to the user. Then, a framework based on iterative distance browsing will be proposed that supports efficient computation of the probabilistic similarity ranking.

The representation of uncertain objects will be based on Definition 9.2 (cf. Chapter 9), where the objects are assumed to incorporate positional uncertainty, i.e., each uncertain



(a) Query on objects represented by mean positions.

(b) Query on objects with full uncertainty.

Figure 11.1: Distance range query on objects with different uncertainty representations.

object consists of a set of multiple observations which are mutually exclusive.

In the following, Section 11.2 will formally define different semantics of probabilistic ranking on uncertain objects. Then, Section 11.3 will introduce a framework containing the essential modules for computing the rank probabilities of uncertain observations. In Section 11.4, two approaches will be presented to speed-up the computation of the rank probability distribution. These approaches will be evaluated w.r.t. effectiveness and efficiency in Section 11.5. Finally, Section 11.6 will conclude this chapter.

11.2 Problem Definition

11.2.1 Distance Computation for Uncertain Objects

In order to compute a rank probability distribution for positionally uncertain objects w.r.t. their spatial distances to a query object, also these distances have to incorporate the uncertainty. Like the uncertain position, the distance between two uncertain objects (or between two objects where at least one of them is an uncertain object) can be described by a PDF that reflects the probability for each possible distance value. However, for uncertain objects with discrete uncertainty representations, another form of distance is needed. This distance is defined as follows:

Definition 11.1 (Uncertain Distance) *Let $dist : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_0^+$ be an L_p-norm-based distance function, and let $X \in \mathcal{D}$ and $Y \in \mathcal{D}$ be two uncertain objects, where X and Y are assumed to be independent of each other. Then, an uncertain distance in the discrete uncertainty model is a collection*

$$dist_u(X,Y) = \{(d,p) \in \mathbb{R}_0^+, \forall (x, P(X=x)) \in X, \forall (y, P(Y=y)) \in Y :$$

$$d = dist(x,y), p = P(X=x) \cdot P(Y=y)\}.$$

Here, the condition $\sum_{(d,p)\in dist_u(X,Y)} p = 1$ *holds.*

Since distance computations between uncertain objects are very expensive, computationally inexpensive distance approximations are required in order to reduce the candidate set in a filter step. For this reason, it makes sense to introduce distance approximations that lower and upper bound the uncertain distance between two uncertain objects.

Definition 11.2 (Minimum and Maximum Object Distance) *Let* $X = \{x_1, \ldots, x_m\}$ *and* $Y = \{y_1, \ldots, y_{m'}\}$ *be two uncertain objects. Then, the distance*

$$minDist(X, Y) = \min_{i\in\{1,\ldots,m\}, j\in\{1,\ldots,m'\}} (dist(x_i, y_j))$$

is called minimum distance *between the objects* X *and* Y*, and*

$$maxDist(X, Y) = \max_{i\in\{1,\ldots,m\}, j\in\{1,\ldots,'\}} (dist(x_i, y_j))$$

is called maximum distance *between* X *and* Y*.*

11.2.2 Probabilistic Ranking on Uncertain Objects

A probabilistic ranking query assigns a set of probability values to each result object, one value for each ranking position. This *Rank Probability Distribution* (*RPD*) is defined as follows:

Definition 11.3 (Rank Probability Distribution (RPD)) *Let* Q *be an uncertain query object and let* $\mathcal{D}$ *be a database containing* N *uncertain objects. A* Rank Probability Distribution (RPD) *is a function* $P_Q : \mathcal{D} \times \{1, \ldots, k\} \to [0, 1]$ *that reports, for a database object* $X \in \mathcal{D}$ *and a ranking position* $i \in \{1, \ldots, k\}$*, the probability which reflects the likelihood that* X *is on the* i*th ranking position w.r.t. the uncertain distance* $dist_u(X, Q)$ *between* X *and the query object* Q *in ascending order.*

Table 11.1 summarizes the most frequently used notations of this chapter on the following page. Chapter 12 will also fall back on these notations.

Assuming $k = N$, the RPD represents a complete probabilistic assignment of each database object to its possible ranking positions, which can be visualized by a bipartite graph, where the zero probabilities $P_Q(X, i) = 0$ $(1 \leq i \leq N)$ can be omitted (cf. Figure 9.2 in Chapter 9). For this reason, diverse variants of query definitions will be proposed that can be easily motivated by the fact that the user could be overstrained with ambiguous ranking results. They specify how the results of an RPD can be aggregated and reported in a more comfortable form which is more easy to read. In particular, for each ranking position, only one object is reported, i.e., for each ranking position i, the object which is most likely to appear on the given position i is reported. The final unambiguous rankings can be built in a postprocessing step.

Notation	Description
$\mathcal{D}$	an uncertain database
N	the cardinality of $\mathcal{D}$
k	the ranking depth that determines the number of ranking positions of the ranking query result
m	the number of observations belonging to an object
Q	an uncertain query object in respect to which a rank probability distribution (RPD) is computed
q	a query observation belonging to Q in respect to which an RPD is computed
$\mathcal{B}$	a distance browsing of $\mathcal{D}$ w.r.t q
X, Y, Z	uncertain objects, each corresponding to a finite set of alternative observations
x, y, z	observations belonging to the objects X, Y, Z respectively
$P(X = x)$	the probability that object X is represented by observation x
$\mathcal{S}$	a set of objects that have already been retrieved, i.e., the set that contains an object X *iff* at least one observation of X has already been returned by the distance browsing $\mathcal{B}$
$P_q(X, i)$	the probability that object X is assigned to the ith ranking position i, i.e., the probability that exactly $i-1$ objects in $(\mathcal{D} \setminus \{X\})$ are closer to q than X
$P_q(x, i)$	the probability that an observation x of object X is assigned to the ith ranking position i, i.e., the probability that exactly $i-1$ objects in $\mathcal{D} \setminus \{X\}$ are closer to q than x
$P_{i,\mathcal{S},x}$	the probability that exactly i objects $Z \in \mathcal{S}$ are closer to q than observation x
$P_x(Z)$	the probability that object Z is closer to the query observation q than the observation x; computable using Lemma 11.3

Table 11.1: Table of notations used in this chapter and in Chapter 12.

U-kRanks Query

A *U-kRanks* query is defined according to [192, 214] as follows:

Definition 11.4 (U-kRanks) *A* U-kRanks query *incrementally retrieves for a ranking position i a result tuple of the form $(X, P_Q(X, i))$, where $X \in \mathcal{D}$ has a higher probability then all other objects $\forall Z \in \mathcal{D} \setminus \{X\}$ to appear on the ranking position i, formally*

$$P_Q(X, i) \geq P_Q(Z, i).$$

In a U-kRanks query, an object can be assigned to multiple ranking positions, or it can not occur at all.

Probabilistic Ranking Query Based on Maximum Confidence

A similar query definition reports the objects in such a way that the ith reported object has the highest confidence to be at the given ranking position i, but without multiple or empty assignments of objects.

Definition 11.5 (*PRQ_MC*) *A* probabilistic ranking query based on maximum confidence (PRQ_MC) *incrementally retrieves for a ranking position i a result tuple of the form $(X, P_Q(X,i))$, where $X \in \mathcal{D}$ has not been reported at previous ranking iterations (i.e., at ranking positions $j < i$) and $Z \in \mathcal{D} \setminus \{X\}$ which have not been reported at previous ranking iterations, formally*

$$P_Q(X,i) \geq P_Q(Z,i),\ X \text{ has not been reported at previous ranking iterations.}$$

These two types of queries only consider the probability of an object to be ranked on a particular ranking position. The confidences of prior ranking positions of an object are ignored in the case they are exceeded by another object. However, the confidences of prior ranking positions might also be relevant for the final setting of the ranking position of an object. This assumption will be taken into account with the next query definition.

Probabilistic Ranking Query Based on Maximum Aggregated Confidence

The next query definition *PRQ_MAC* takes aggregated confidence values of ranking positions into account. Contrary to the previous definition, this query assigns to each object X a unique ranking position i by aggregating over the confidences of all prior ranking positions $j < i$ according to X. Thus, this definition extends the semantics of *PRQ_MC* by aggregation.

Definition 11.6 (*PRQ_MAC*) *A* probabilistic ranking query based on maximum aggregated confidence (PRQ_MAC) *incrementally retrieves for a ranking position i a result tuple of the form $(X, \sum_{j \in \{1,\ldots,i\}} P_Q(X,j))$, where $X \in \mathcal{D}$ has not been reported at previous ranking iterations (i.e., at ranking positions $j < i$) and $Z \in \mathcal{D} \setminus \{X\}$ which have not been reported at previous ranking iterations, formally*

$$\sum_{j=1}^{i} P_Q(X,j) \geq \sum_{j=1}^{i} P_Q(Z,j).$$

The query types defined above specify the ranking position of each object X by comparing the ranking position confidence of X with the confidences of the other objects.

Probabilistic Ranking Query Based on Expected Matching

This query assigns to each object its expected ranking position without taking the confidences of the other objects into account.

Rank	A	B	C	U-kRanks	*PRQ_MC*	*PRQ_MAC*	*EM*
1	0.8	0.0	0.2	A (0.8)	A (0.8)	A (0.8)	A (1.3)
2	0.1	0.5	0.4	B (0.5)	B (0.5)	C (0.6)	C (2.2)
3	0.1	0.5	0.4	B (0.5)	C (0.4)	B (1.0)	B (2.5)

Table 11.2: Object-rank probabilities from Example 11.1.

Definition 11.7 (*PRQ_EM*) *A* probabilistic ranking query based on expected matching (PRQ_EM) *globally retrieves a result tuple of the form* $(X, \mu(X))$ *and assigns to the ranking position* i *the object* $X \in \mathcal{D}$ *which has the* i*th highest expected rank; formally*

$$\mu(X) = \sum_{i=1}^{N} i \cdot P_Q(X, i).$$

In other words, the objects are reported in ascending order of their expected ranking position. This corresponds to the *expected rank* semantics [83].

Discussion

As already stated, the suggested unambiguous probabilistic ranking output types contain different semantics. The following example provides an overview of the advantages and drawbacks.

Example 11.1 *Consider three uncertain objects A, B and C, for which an RPD according to Definition 11.3 has been computed. These ranking probabilities are illustrated in Table 11.2. Object A has a probability of 80% to appear on rank 1 and of 10% to appear at ranks 2 and 3, respectively. Object B will never appear on rank 1, but with 50% on ranks 2 and 3, respectively. The probabilities of object C are 20% for rank 1, 40% for rank 2 and 40% for rank 3. According to the definition of U-kRanks [192, 214], the object with the highest probability will appear on the corresponding ranking position, even if it has already been returned for a previous ranking position. Thus, this output assigns A to rank 1, B to rank 2 and again B to rank 3. The drawback here is that B appears twice, whereas C does not appear in the result at all. The* PRQ_MC *semantics tackles this lack, as no object can be reported for a ranking position if it has already been reported before. Thus, for rank 3, the only object left is C. This approach avoids multiple assignments of objects. However, it does not consider the probability distribution of the other objects and the prior ranks. The* PRQ_MAC *semantics provides a solution using the aggregated probabilities, also considering the values for the previous ranks. Then, A remains on rank 1 due to the highest probability. For rank 2, the probability of C is* $0.2 + 0.4 = 0.6$ *and therefore higher than the probability of B, which is* $0 + 0.5 = 0.5$*; hence, C is ranked second. A has already been assigned before and is not considered here. Finally, B is ranked third, as it is the only remaining object. The expected rank approach EM assigns the objects to the ranking positions w.r.t. an ascending order of their expected ranks; thus, A is ranked first with an*

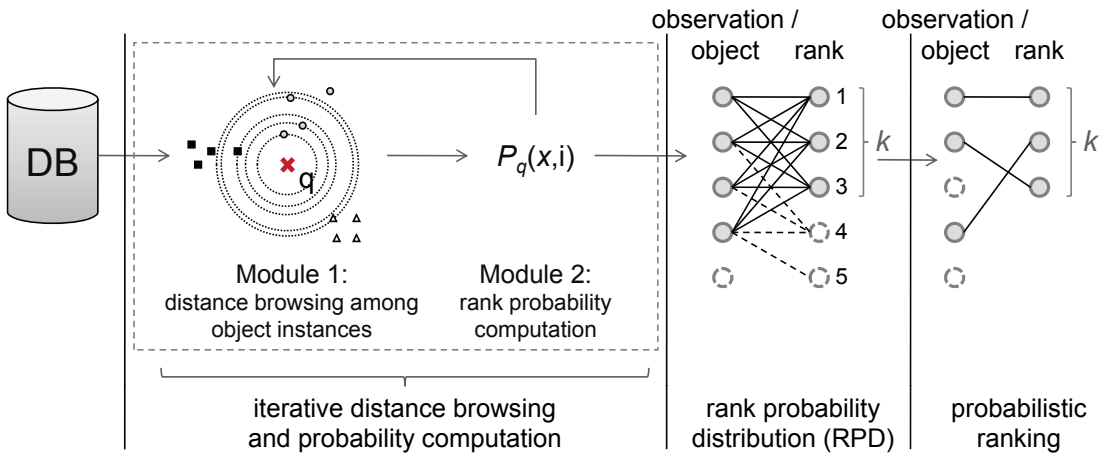


Figure 11.2: Framework for probabilistic similarity ranking.

expected rank of 1.3, C is ranked second (2.2) and B is ranked third (2.5), providing the same definite ranking result as PRQ_MAC. *However, EM has also some drawbacks [114].*

The user is now able to decide which of the semantics fits to his or her problem definition. Nevertheless, the drawback of some approaches can be a decisive factor.

11.3 Probabilistic Ranking Framework

11.3.1 Framework Modules

The basic framework on which the solutions of this and the next chapter will rely consists of two modules, which are performed in an iterative way (cf. Figure 11.2):

- **Module 1**: The first module (*distance browsing* $\mathcal{B}$) incrementally retrieves the observations of all objects in order of their distance to a query observation q. More details will be given in Subsection 11.3.2.
- **Module 2**: The second module computes the RPD, i.e., the rank probability $P_q(x, i)$ of each observation x to be ranked on the ith ranking position w.r.t. the distance to a query observation q, reported from the distance browsing for all $1 \leq i \leq k$. Details will be provided in Subsection 11.3.3. In this chapter, the ranking depth k will be assumed to be equal to the database size N, i.e., a complete object ranking will be retrieved.

The computation of the RPD is iteratively processed within a loop. First, a distance browsing is initialized among the observations starting from a given query point q. For each observation fetched from the distance browsing (Module 1), the corresponding rank probabilities are computed (Module 2) and the RPD, generated from the probabilistic ranking routine, is updated.

The rank probabilities of the observations (i.e., tuples in the x-relation model) reported from the second module can optionally be aggregated into rank probabilities of the objects (i.e., x-tuples in the x-relation model), as described in Section 11.4. Moreover, having computed the (object- or observation-based) RPD for each query observation $q \in Q$, the results can finally be merged by weighing the partial results for each $q \in Q$ by the confidence of q of representing Q, i.e.,

$$P_Q(X, i) = \sum_{q \in Q} P_q(X, i) \cdot P(q \in Q).$$

Finally, in a postprocessing step, the RPDs computed by the proposed framework can be used to generate an unambiguous ranking of the objects or observations, as proposed in Subsection 11.2.2.

11.3.2 Iterative Probability Computation

Distance Browsing ($\mathcal{B}$)

Initially, a complete ordering of all observations w.r.t. the distance to q is initialized, which is called *distance browsing* $\mathcal{B}$. This can be facilitated by storing the observations in a spatial index structure like the R^*-tree [23] and by using an incremental nearest neighbor search algorithm [107]. Then, the observations are iteratively picked from the distance browsing $\mathcal{B}$. For each observation $x \in X$ returned from $\mathcal{B}$ follows the immediate computation of $P_q(x, i)$, which denotes the probability that x is on rank i w.r.t. the distance to q for all i $(1 \leq i \leq N)$. Thereby, all other observations $x' \neq x$ of object X have to be ignored due to the observation dependency of mutual exclusiveness, which is assumed by the data model (cf. Definition 9.2 in Chapter 9). In general, other orders used for the observation browsing, e.g., descending probability, as discussed in [214], might possibly lead to faster algorithms if the probability distribution favors them. However, the distance-based order is somewhat natural for incremental nearest-neighbor 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 restricted memory.

For the probability computation, two auxiliary data structures are needed: the *Observation Table* (OT), required to compute the probabilities by incorporating the other objects $Z \in \mathcal{D} \setminus \{X\}$, and the *Probability Table* (PT), used to maintain the intermediate results w.r.t. x and which finally contains the RPD. In the following, both data structures OT and PT will be introduced in detail.

Observation Table (OT)

The observation table OT stores, for each accessed object separately, the portion of observations already returned from $\mathcal{B}$. Additionally, for each accessed object, the portion

of observations is required that has not been accessed so far. Entries of OT according to object X_i are defined by

$$OT[i][1] = \sum_{j=1}^{h} P(X_i = x_{i,j}),$$

where $x_{i,j}, 1 \leq j \leq h \leq m$ are the observations of X_i fetched by $\mathcal{B}$ in previous processing iterations. With $\{x_{i,j} : x_{i,j} \in X_i, 1 \leq j \leq m\}$ denoting the complete set of observations of X_i, it holds that

$$\sum_{i=1}^{h} P(X_i = x_{i,j}) \leq \sum_{i=1}^{m} P(X_i = x_{i,j}).$$

Consequently, $OT[i][0]$ denotes the portion of remaining, not yet returned observations and can be directly computed by $OT[i][0] = 1 - OT[i][1]$, such that, in fact, only entries for $OT[i][1]$ have to be maintained.

Probability Table (PT)

The probability table PT stores for each object X_i $(1 \leq i \leq N)$ and each $r \in \{1, \ldots, N\}$ the actual probability that $r-1$ objects in $\mathcal{D} \setminus \{X_i\}$ are closer to the query observation q than X_i. The entries of PT according to the jth observation of object X_i are defined as follows:

$$PT[r][i][j] = P_q(x_{i,j}, r)$$
$$= P[(r-1) \text{ objects } Z \in \mathcal{D} \setminus \{X_i\}) \text{ are closer to } q \text{ than the observation } x_{i,j}].$$

Here, the assumption is made that object X_i is the ith object for which $\mathcal{B}$ has reported at least one observation. The same assumption is made for the observations of an uncertain object (i.e., the observation $x_{i,j}$ is the jth closest observation of object X_i according to q). These assumptions hold for the rest of this chapter.

11.3.3 Probability Computation

This subsection will show how to compute an entry $PT[r][i][j]$ of the probability table using the information stored in the observation table OT. Let OT be an observation table of size N (i.e., OT stores the information corresponding to all N objects of the database $\mathcal{D}$). Let $\sigma_r(i) \subseteq \{Z \in \mathcal{D} : Z \neq X_i\}$ denote the set, called *r-set* of X_i, containing exactly $r-1$ objects. If $r < N$ is assumed, obviously $\binom{N-1}{r-1}$ different r-set permutations $\sigma_r(i)$ exist. For the computation of $PT[r][i][j]$, it is important to consider the set S_r of all possible r-set permutations according to X_i. The probability that exactly $r-1$ objects are closer to the query observation q than the observation $x_{i,j}$, can be computed as follows:

$$PT[r][i][j] = \sum_{\sigma_r(i) \in S_r} \prod_{\substack{h \in \{1,\ldots,N\} \\ h \neq i}} \begin{cases} OT[h][1] & \text{if } X_h \in \sigma_r(i) \\ OT[h][0] & \text{if } X_h \notin \sigma_r(i) \end{cases}$$

Assuming that the observation $x_{i,j}$ is currently processed, the following characteristics hold: since the observations are processed in ascending order w.r.t. the distance to q, the observation table entry $OT[h][1]$ reflects the probability that object X_h is closer to q than the observation $x_{i,j}$. On the other hand, $OT[h][0]$ reflects the probability that $x_{i,j}$ is closer to q than X_h.

The entries of the probability table can now be computed by iteratively fetching the observations from $\mathcal{B}$. Thereby, all entries of the probability table are initially set to 0. Then, the distance browsing $\mathcal{B}$, which reports one observation of an uncertain object in each iteration, is started. Each reported observation $x_{i,j}$ is used to compute for all r $(1 \leq r \leq N)$ the probability value that corresponds to the table entry $PT[r][i][j]$. After filling the (i,j)-column of the probability table, the next observation is fetched from $\mathcal{B}$ in the same way as this was done with $x_{i,j}$. This procedure is repeated until all observations are fetched from $\mathcal{B}$.

The computation of the probability table can be very costly in space and time. One reason is the size of the table that grows drastically with the number of objects and the number of observations for each object. Another problem is the very expensive computation of the probability table entries $PT[r][i][j]$, which is the computational bottleneck of the proposed probabilistic ranking algorithm. For each entry of $PT[r][i][j]$, the computation is required for the probabilities according to $\begin{pmatrix} N-1 \\ r-1 \end{pmatrix}$ different r-set permutations which have to be summed up to the final probability value. For example, assuming $N-1=100$ and $r-1=20$, about $1.73 \cdot 10^{13}$ r-set permutations need to be considered. Therefore, Section 11.4 will propose methods that achieve a considerable reduction of the overall query cost.

11.4 Accelerated Probability Computation

11.4.1 Table Pruning

If it is the final goal of probabilistic ranking to rank uncertain objects – not observations –, it is not needed to maintain the separate results according to each observation of an object. Instead of maintaining a table entry for each observation, the solution at hand is to on-the-fly summing up the iteratively computed observation probabilities and weighing them by the occurrence probabilities (confidences) of the respective observations. If a final RPD for observations is required (e.g., for the original U-kRanks output as retrieved in [192, 214]), the separate entries have to be maintained.

An additional reduction of both tables OT and PT (i.e., a reduction to those parts of the table that should be available at once) can be achieved by only maintaining those parts of the table that are required for further computations. Firstly, it is required to maintain a table column only for those objects for which at least one observation has been reported from $\mathcal{B}$, whereas the columns for those objects, for which all observations have already been fetched or for which no observation has been retrieved, can be skipped. Secondly,

each row of PT that corresponds to a ranking position which is not within a particular ranking range can be skipped as well. This range is given by the minimum and maximum ranking position of uncertain objects for which currently a column of the probability table has to be maintained. The following lemmata utilize the bounds for uncertain distances that were introduced in Definition 11.2. A lower bound for the ranking position of an uncertain object is defined as follows.

Lemma 11.1 (Minimum Ranking Position) *Let $X \in \mathcal{D}$ be an uncertain object and let q be the query observation. Furthermore, let $N_1 < N$ objects $Y \in \mathcal{D} \setminus \{X\}$ have a maximum distance that is smaller than the minimum distance of X, i.e., $|\{Y \in \mathcal{D} \setminus \{X\} : maxDist(Y, q) < minDist(X, q)\}| = N_1$. Then, the ranking position of object X must be at least $N_1 + 1$.*

Analogously, an upper bound for the ranking position of an uncertain object is defined as follows.

Lemma 11.2 (Maximum Ranking Position) *Let $X \in \mathcal{D}$ be an uncertain object and let q be the query observation. Furthermore, let $N_2 < N$ objects $Y \in \mathcal{D} \setminus \{X\}$ have a minimum distance that is higher than the maximum distance of X, i.e., $|\{Y \in \mathcal{D} \setminus \{X\} : minDist(Y, q) > maxDist(X, q)\}| = N_2$. Then, the ranking position of object X must be at most $N - N_2$.*

As mentioned above, the computation of the object probabilities according to the ranking position i only requires to consider those objects whose minimum and maximum ranking position cover the ranking position i. This holds for those objects having at least one observation within the current ranking position range. For all other objects, this rule of spatial pruning can be applied. Usually, in practice, this is the case for only a small set of objects, depending on their spatial variance, also referred to as *degree of uncertainty*. The experimental section of this chapter (Section 11.5) will reflect the degree uncertainty of an object by the spatial variance of its observations. This definition will slightly vary in Chapters 12 and 13, where the degree of uncertainty will correspond to the side length of the hyperrectangle in which the observations are distributed and to the standard deviation of the observations. However, the information contained in these different semantics can be regarded as similar.

11.4.2 Bisection-Based Algorithm

In the case of subsequently fetching observations belonging to the same object, the ranking probabilities according to this object do not change. Hence, obviously only one computation of the probability value is required. However, the general case where two adjacent observations reported from the ranking belong to different objects occurs more frequently. For this case, the computational cost can be significantly reduced if a bisection-based algorithm is utilized, as proposed in [45]. The bisection-based algorithm uses a divide-and-conquer technique which computes, for a query observation q and a database object X, the

Algorithm 6 Bisection-Based Algorithm: bisection(OT, min, max, r)

Require: OT, min, max, r

```
 1: result ← 0
 2: N ← max − min + 1
 3: if r = 1 then
 4:    result ← ∏_{i=min}^{max} OT[i][0]
 5: else if r ≥ N then
 6:    result ← ∏_{i=min}^{max} OT[i][1]
 7: else
 8:    mid ← ⌈(min + max)/2⌉
 9:    for (i = 0 → min(⌈(max − min)/2⌉, r − 1)) do
10:       P_left ← bisection(OT, min, mid − 1, r − i − 1)
11:       P_right ← bisection(OT, mid, max, i)
12:       result ← result + (P_left · P_right)
13:    end for
14: end if
15: return result
```

probability that the object X is on rank r w.r.t. the distance to the query observation q, i.e., that exactly $r-1$ other objects are closer to q than the object X. Hence, the number of r-set permutations that have to be computed can be reduced drastically. The main idea is to recursively perform a binary split of the set of relevant objects, i.e., objects which have to be taken into account for the probability computation. Instead of considering all $r-1$ out of $N-1$ permutations, the r-set is split into two subsets of equal size. Then, only $r-i-1$ out of $\frac{N-1}{2}$ permutations for $i \in \{0,\ldots,r-1\}$ have to be considered for the one subset, combined with the i out of $\frac{N-1}{2}$ permutations of the other subset. As a consequence, instead of considering $\begin{pmatrix} N-1 \\ r-1 \end{pmatrix}$ r-set permutations, the number of r-set permutations to be considered can be reduced to

$$\sum_{i=0}^{r-1} \left(\begin{pmatrix} \frac{N-1}{2} \\ r-i-1 \end{pmatrix} + \begin{pmatrix} \frac{N-1}{2} \\ i \end{pmatrix} \right).$$

The pseudocode for the computation of the rank probability is illustrated in Algorithm 6. The bucket range of the r-set that is currently worked on is limited by the parameters min and max. The observation table, which is used for probability computation (cf. Subsection 11.3.2), is denoted by the additional parameter OT. The r-set split can be recursively repeated for each subset. The recursive decomposition of a subset into two buckets for each recursion, from which $r-1$ ($0 < r < N$) out of $N-1$ permutations have to be computed, stops if $r \geq N$. Then, there exists only one permutation $\sigma_r(i)$ in the current bucket that can be immediately computed and reported to the calling function of the recursion (line 6). Otherwise, the actual recursive splitting, that computes the results for the two summands P_{left} and P_{right} in each recursion, is performed in lines 9ff. The

size of the divided permutations $\sigma_r(i)$ is determined by the minimum of the bucket size $\lceil \frac{max-min}{2} \rceil$ and $r-1$. If $r=1$, the probability that there is no object closer to q than $x_{i,j}$ is computed (line 4).

Afterwards, the corresponding results can be efficiently merged into the final result. Although this approach accelerates the computational cost of the $PT[r][i][j]$ significantly, the asymptotical cost is still exponential in the ranking range.

11.4.3 Dynamic-Programming-Based Algorithm

In the following, an algorithm will be introduced that accelerates the computation by several orders of magnitude. This algorithm utilizes a dynamic-programming scheme, also known as *Poisson Binomial Recurrence*, first introduced in [147]. For the context uncertain top-k queries, this scheme was originally proposed in [214] on the *x-relation model*, which was the first approach that solves probabilistic queries efficiently by means of dynamic-programming techniques. Here, this scheme is extended to the use with spatial data and computes the probability that an uncertain object $X \in \mathcal{D}$ is assigned to a certain ranking position w.r.t. the distance to a query observation q.

The probabilities of PT can be efficiently computed requiring a complexity of $O(N^3)$. The key idea of this approach is based on the following property. Given a query observation q, an observation x of an uncertain database object X and a set of h objects $\mathcal{S} = \{Z_1, Z_2, \ldots, Z_h\}$ for which the probability $P_x(Z)$ that $Z \in \mathcal{S}$ is closer to the query observation q than x (i.e., that Z is closer to q than x) is known (i.e., all objects Z for which at least one observation has been retrieved from $\mathcal{B}$). The probability $P_x(Z)$ can be computed according to the following lemma.

Lemma 11.3 *Let q be the query object and let $(x, P(X=x))$ be the observation x of an object X fetched from the distance browsing $\mathcal{B}$ in the current processing iteration. The probability that an object $Z \neq X$ is closer to q than x is*

$$P_x(Z) = \sum_{i=1}^{j} P(Z = z_i),$$

where $z_i \in Z, 1 \leq i \leq j$ are the observations of Z fetched in previous processing iterations.

Lemma 11.3 says that it is possible to accumulate, in overall linear space, the probabilities of all observations for all objects which have been seen so far and to use them to compute $P_x(Z)$, given the current observation x and any object $Z \in \mathcal{D} \setminus \{X\}$.

Now, the probability $P_{i,\mathcal{S},x}$ that exactly i objects $Z \in \mathcal{S}$ are ranked higher than x w.r.t. the distance to q can be computed efficiently, utilizing the following lemma.

Lemma 11.4 *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 the assumption is made that object $Z \in \mathcal{S}$ is farther from q than x, then i objects of $\mathcal{S} \setminus \{Z\}$ must be closer to q.*

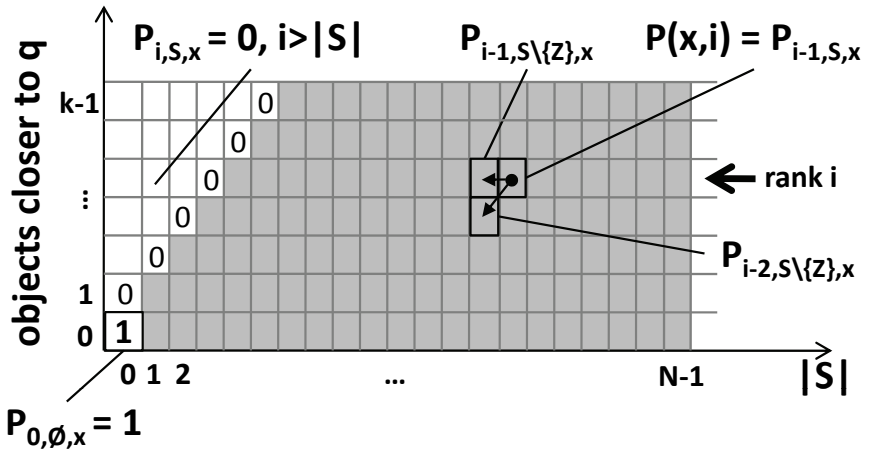


Figure 11.3: Visualization of the dynamic-programming scheme.

The above lemma leads to the following recursion that allows to compute $P_{i,\mathcal{S},x}$ by means of the paradigm of dynamic programming:

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

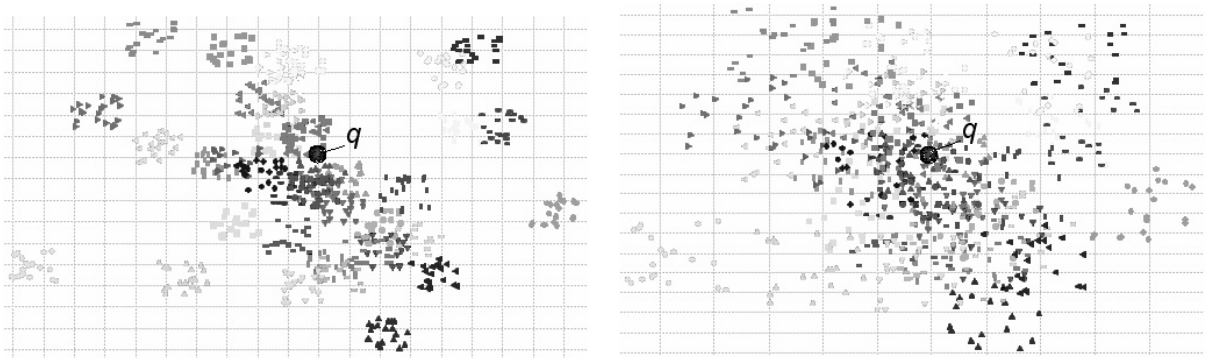
where

$$P_{0,\emptyset,x} = 1 \text{ and } P_{i,\mathcal{S},x} = 0 \text{ if } i < 0 \vee i > |\mathcal{S}|. \tag{11.1}$$

An illustration of this dynamic-programming scheme is given in Figure 11.3, where the size of $\mathcal{S}$ is marked along the x-axis and the number of objects that are closer to q than the currently processed observation x is marked along the y-axis. The shaded cells represent the probabilities that have to be determined during the process of the RPD computation. As illustrated, each grid cell (which is exemplary marked with a dot in Figure 11.3) can be computed using the values contained in the left and the lower left cells. If the ranking depth is restricted to k, all probabilities are needed that up to $k-1$ out of $N-1$ objects – not N objects, as x cannot be preceded by the object it belongs to – are closer to x. In each iteration of the dynamic-programming algorithm, $O(N \cdot k)$ cells have to be computed (which is $O(N^2)$ in the setting of this chapter). Performing this for each observation that is retrieved from the distance browsing $\mathcal{B}$, this yields an overall runtime of (N^3), as it can be assumed that the total number of observations in the database is linear in the number of database objects.

Regarding the storage requirements for the probability values, the computation of each probability $P_{i,\mathcal{S},x}$ only requires information stored in the current line and the previous line to access the probabilities $P_{i-1,\mathcal{S}\setminus\{Z\},x}$ and $P_{i,\mathcal{S}\setminus\{Z\},x}$. Therefore, only these two lines (of length N) need to be preserved requiring $O(N)$ space. The probability table PT used in the straightforward and in the divide-and-conquer-based approach (cf. Subsection 11.3.2), in contrary, had to store $N^2 \cdot m$ values, resulting in an overall space requirement of $O(N^3)$.

While the bisection-based algorithm still requires exponential asymptotical runtime for the computation of the RPD, the dynamic-programming-based algorithm only requires a



(a) $UD = 2.0$. (b) $UD = 5.0$.

Figure 11.4: Uncertain object distribution in 60×60 space for different degrees of uncertainty ($N = 40$, $m = 20$).

worst-case runtime of $O(N^3)$. This can be further reduced to a quadratic runtime w.r.t. N, if the ranking depth k is assumed to be a small constant, which yields a complexity of $O(k \cdot N^2)$. In Chapter 12, a solution will be presented which computes the RPD in linear time w.r.t. the database size. Therefore, the above dynamic-programming scheme will be enhanced. Chapter 12 will also show how the proposed framework, enhanced with the linear-time solution, can be used to support and significantly boost the performance of state-of-the-art probabilistic ranking queries.

11.5 Experimental Evaluation

11.5.1 Datasets and Experimental Setup

This section will examine the effectiveness and efficiency of the proposed probabilistic similarity ranking approaches. [45] only provides a sparse experimental part; therefore, this section comprises the evaluation provided in [49]. Since the computation is highly CPU-bound, the measurements describe the efficiency by the overall runtime cost required to compute an entire ranking averaged over ten queries.

The following experiments are based on artificial and real-world datasets. The artificial datasets *ART*, which were used for the efficiency experiments, contain 10 to 1,000 ten-dimensional uncertain objects that are located by a Gaussian distribution in the data space. Each object consists of $m = 10$ observations that are uniformly distributed around the mean positions of the objects with a variance (in the following referred to as *degree of uncertainty* (*UD*)) of 10% of the data space, if not stated otherwise. Figure 11.4 exemplarily depicts the distribution of uncertain objects when varying *UD*. A growing degree of uncertainty leads to an increase of the overlap between the observations.

For the evaluation of the effectiveness of the proposed methods, two real-world datasets were used: O_3 and *NSP*. The O_3 dataset is an environmental dataset consisting of 30

Dataset	**PRQ_MC**	**PRQ_MAC**	**PRQ_EM**	**MP**
O_3	0.51	**0.65**	0.53	0.63
NSP_h	0.36	**0.43**	0.29	0.35
NSP_{frq}	0.62	**0.70**	0.41	0.60

Table 11.3: Avg. precision for probabilistic ranking queries on different real-world datasets.

uncertain objects created from time series, each composing a set of measurements of the ozone concentration in the air measured within one month[1]. Thereby, each observation features a daily ozone concentration curve. The dataset covers observations from the years 2000 to 2004 and is labeled according to the months in a year. *NSP* is a chronobiologic dataset describing the cell activity of Neurospora[2] within sequences of day cycles. This dataset is used to investigate endogenous rhythms. It can be classified w.r.t. two parameters among others: day cycle and fungal type. For the experiments, two subsets of the *NSP* datasets were used: NSP_h and NSP_{frq}. NSP_h is labeled according to the day cycle length. It consists of 36 objects that created three classes of day cycle (16, 18 and 20 hours). The NSP_{frq} dataset consists of 48 objects and is labeled w.r.t. the fungal type ($frq1$, $frq7$ and $frq+$).

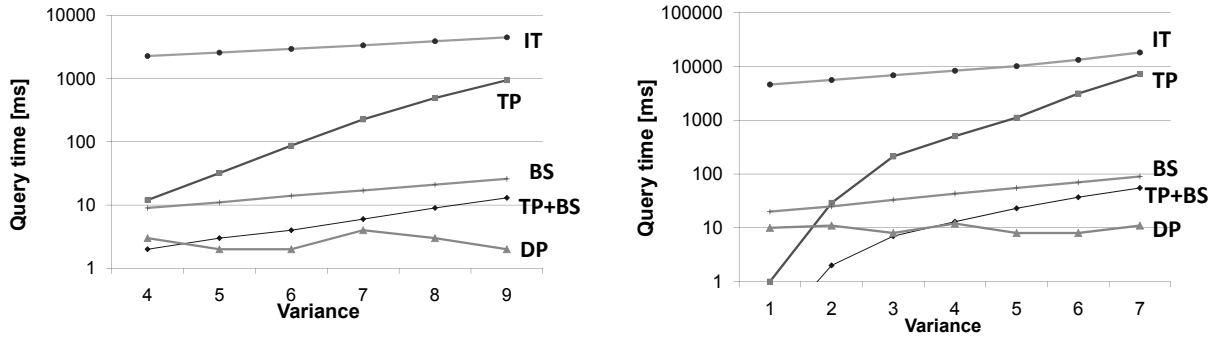
11.5.2 Effectiveness Experiments

The first experiments evaluate the quality of the different probabilistic ranking query semantics (**PRQ_MC**, **PRQ_MAC**, **PRQ_EM**) proposed in Subsection 11.2.2. In order to make a fair evaluation, the results obtained with these approaches were compared with the results of a non-probabilistic ranking (**MP**) which ranks the objects based on the distance between their mean positions. For these experiments, the three real-world datasets O_3, NSP_h and NSP_{frq} were used, each consisting of uncertain objects which are labeled as described above.

In order to evaluate the quality of the semantics, a k-nearest neighbor (k-NN) classification was performed. According to the semantics of a classification [103], objects are divided into positive (P) and negative (N) objects, which denote the number of objects that are returned by a classifier w.r.t. a label and the number of objects that have been discarded, respectively. In the context of document retrieval, a popular measure that rates the overall significance of query results when, for example, retrieving the k most similar documents, is the *precision* [103], which denotes the percentage of relevant objects that have been retrieved, and, thus, serves as a measure that can also reflect the quality of the similarity ranking schemes. Formally, the precision is defined by $\frac{TP}{P}$, which yields values between 0 and 1. Hereby, TP denotes the number of retrieved relevant objects. The average precision over all class labels (cf. dataset description in Subsection 11.5.1)

[1]The O_3 dataset has been provided by the Bavarian State Office for Environmental Protection, Augsburg, Germany (`http://www.lfu.bayern.de/`).

[2]*Neurospora* is the name of a fungal genus containing several distinct species. For further information see *The Neurospora Home Page*: `http://www.fgsc.net/Neurospora/neurospora.html`



(a) Number of observations $m = 10$.

(b) Number of observations $m = 30$.

Figure 11.5: Query processing cost w.r.t. *UD*.

can be observed from Table 11.3. Concluding, **PRQ_MAC** provides a superior result quality to the other approaches including the non-probabilistic ranking approach **MP**. Interestingly, the approach **PRQ_MC**, which has a quite similar definition as the *U-kRanks* query proposed in [192, 214], does not work very well and shows similar quality to **MP**. The approach **PRQ_EM** loses clearly and is even significantly below the non-probabilistic ranking approach **MP**. This observation points out that the postprocessing step, i.e., the way in which the results of the RPD are combined to a definite result, indeed affects the quality of the result.

11.5.3 Efficiency Experiments

The next experiment evaluates the performance of the proposed probabilistic ranking acceleration strategies proposed in Section 11.4 w.r.t. the query processing time. The different proposed strategies were compared with the straightforward solution without any additional strategy. The competing methods are the following:

- **IT**: Iterative fetching of the observations from the distance browsing $\mathcal{B}$ and computation of the probability table PT entries without any acceleration strategy.
- **TP**: Table pruning strategy where the reduced table space was used.
- **BS**: Bisection-based computation of the probability permutations.
- **TP+BS**: Combination of **TP** and **BS**.
- **DP**: Dynamic-programming-based computation of the probability permutations.

Influence of the Degree of Uncertainty

The first experiment compares all strategies (including the straightforward solution) for the computation of the RPD on the artificial datasets with different values of *UD*. The evaluation of the query processing time of the proposed approaches is illustrated in Figure 11.5.

In particular, the differences between the used computation strategies are depicted for two different numbers of observations per object ($m = 10$ and $m = 30$). Here, a database size of 20 uncertain objects in a ten-dimensional vector space was utilized.

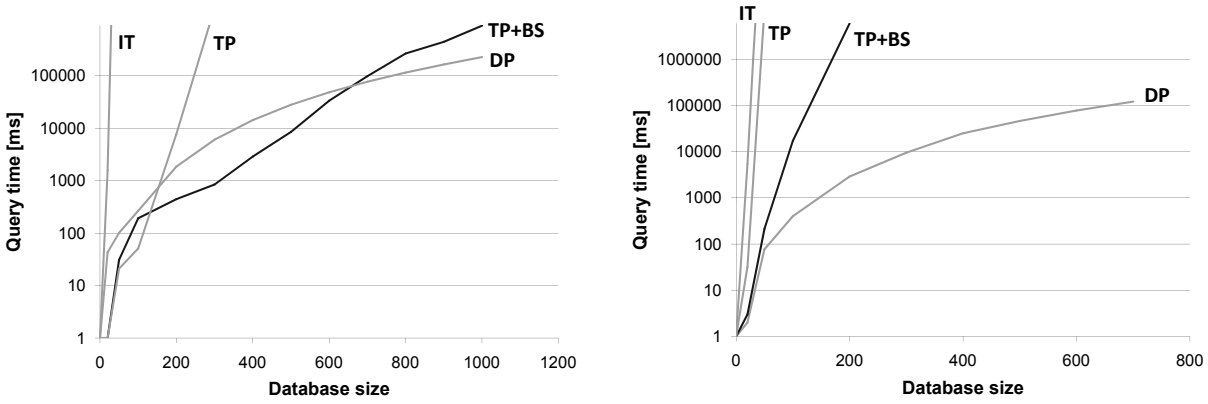
The plain iterative fetching of observations (**IT**) is hardly affected by an increasing *UD* value, as it anyway has to consider all possible worlds for the computations of the probabilistic rank distribution. The table pruning strategy **TP** significantly decreases the required computation time. For a low *UD*, many objects cover only a small range of ranking positions and can, thus, be neglected. An increasing *UD* leads to a higher overlap of the objects and requires more computational effort. For the divide-and-conquer-based computation of **BS**, the query time increases only slightly when increasing *UD*. However, the required runtime is quite high even for a low *UD* value. The runtime of **TP** is much lower for low degrees of uncertainty in comparison with **BS**; here **TP** is likely to prune a high number of objects that are completely processed or not yet seen at all. A combination of the benefits of the **TP** and **BS** strategies results in a quite good performance, but it is outperformed by the **DP** approach. This is due to the independence of the dynamic iterations of the degree of uncertainty, because the iterations require quadratic runtime in any case.

Finally, it can be observed that the behavior with of each approach with an increasing *UD* remains stable for different values of m. However, a higher number of observations per object leads to significantly higher computational requirements of about an order of magnitude for each approach. Thus, these experiments support that the required runtime of computing the RPD is highly dependent on m, so that the need for efficient solutions is obvious.

Scalability

The next experiment evaluates the scalability based on the *ART* datasets of different size. The **BS** approach will be omitted in the following, as the combination **TP+BS** proved to be more effective. Here again, different combinations of strategies were considered. The results are depicted in Figure 11.6 for two different values of *UD*.

Figure 11.6(a) illustrates the results for a low *UD* value. Since, by considering all possible worlds, the simple approach **IT** produces exponential cost, such that experiments for a database size above 30 objects are not applicable. The application of **TP** yields a significant performance gain. Assuming a low *UD* value, the ranges of possible ranking positions of the objects hardly overlap. Furthermore, there are objects that do not have to be considered for all ranking positions, since the minimum and maximum ranking positions of all objects are known (cf. Subsection 11.4.1). It can clearly be observed that the combination **TP+BS** significantly outperforms the case where only **TP** is applied, as the split of the r-sets reduces the number of combinations of higher ranked objects that have to be considered when computing a rank probability for an observation. For small databases where $N < 100$, there is a splitting and merging overhead of the **BS** optimization, which, however, pays off for an increasing database size. For $N < 700$, **TP+BS** even beats the **DP** approach, which is due to the fact that **TP+BS** takes advantage from the combi-



(a) $UD = 0.5$. (b) $UD = 5.0$.

Figure 11.6: Comparison of the scalability of all strategies on the *ART* datasets with different degrees of uncertainty.

nation of two optimizations, whereas the dynamic-programming algorithm **DP** requires cubic runtime complexity anyway (cf. Subsection 11.4.3). However, for higher values of N, **DP** outperforms the other optimizations, as the presence of more objects also leads to the presence of a higher overlap among uncertain objects as well as to an increasing size of the r-sets.

With a high value of UD (cf. Figure 11.6(b)), the behavior of **IT** does not change, as it has to consider all possible worlds anyway, regardless of the distribution of the observations of uncertain objects. Also, **TP** is already not applicable for very small databases because of an increased possible range of ranking positions and an increased overlap among the objects. Even **TP+BS** degenerates soon, despite that the **BS** optimization has a higher effect than the **TP** optimization for high degrees of uncertainty. Finally, as observed before, **DP** is not much affected by the value of UD, and, thus, achieves an improvement of several orders of magnitude in comparison with the other approaches.

11.6 Summary

This chapter introduced a framework that efficiently computes the rank probability distribution (RPD) in order to solve probabilistic similarity ranking queries on spatially uncertain data. In particular, methods were introduced that break down the high computational complexity required to compute, for an object X, the probability that X appears on each ranking position according to the distance to a query object Q. This complexity, in the first approach still exponential in the number of retrieved observations, could be reduced to a polynomial runtime by extending a dynamic-programming technique called *Poisson Binomial Recurrence*. The following chapter will introduce an incremental approach of computing the RPD that enhances the dynamic-programming algorithm and finally achieves an overall runtime complexity which is linear in the number of accessed observations.

Chapter 12

Incremental Probabilistic Similarity Ranking

12.1 Introduction

The step to compute the *Rank Probability Distribution* (*RPD*) that solves the bipartite graph between uncertain objects and ranking positions w.r.t. the distance to a (potentially uncertain) query object represents the main bottleneck of solving the problem of probabilistic ranking. Chapter 11 already adopted a dynamic-programming technique from [214] for the use in spatial data, which can perform this computation in quadratic time and linear space w.r.t. the number of observations required to be accessed until the solution is confirmed. These requirements can finally regarded w.r.t. the database size, as basically, it can be assumed that the total number of observations in the database is linear in the number of database objects. This assumption holds for this chapter. The solution that will be presented in this chapter will further extend the dynamic-programming-based algorithm and reduce the former quadratic time complexity requirements to a linear-time complexity solution. Similarly to Chapter 11, an assumption that will be made here is that the observations can be accessed in increasing distance order to the query observation.

This chapter utilizes the definition of spatially uncertain objects according to Definition 9.2 of Chapter 9. However, the proposed method applies in general to *x-relations* [25] and can be used irrespectively to whether uncertain objects or *x-tuples* are assumed. Thus, it can be used as a module in various semantics that rank the objects or observations according to their rank probabilities.

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

- This chapter will utilize the framework of Chapter 11, which is based on iterative distance browsing and which, thus, efficiently supports probabilistic similarity ranking on spatially uncertain data.

- This chapter will present a theoretically founded approach for computing the RPD, which corresponds to Module 2 of the framework presented in Chapter 11. It will be

proved that the proposed method reduces the computational cost from $O(k \cdot N^2)$, achieved by [214] and Chapter 11, to $O(k \cdot N)$, where N is the size of the database and k denotes the ranking depth; in this chapter, $k < N$ will be assumed. The key idea is to use the ranking probabilities of the previously accessed observation to derive those of the currently accessed observation in $O(k)$ time.

- Similarly to Chapter 11, the objective is to find an unambiguous ranking where each object or observation is uniquely assigned to one rank. Here, any user-defined ranking method (also those suggested in Chapter 11) can be plugged in, as the RPD is required in order to compute unique positions. This will be illustrated for several well-known probabilistic ranking queries that make use of such distributions. In particular, it will be demonstrated that, by using the proposed framework, such queries can be processed in $O(N \cdot log(N) + k \cdot N)$ time[1], as opposed to existing approaches that require $O(k \cdot N^2)$ time.
- Finally, an experimental evaluation will be conducted, using real-world and synthetic data, which demonstrates the applicability of the framework and verifies the theoretical findings.

The rest of this chapter is organized as follows: Section 12.2 will introduce an efficient approach to compute the RPD. The complete algorithm exploiting the framework will be presented in Section 12.3. Section 12.4 will apply the approach to different probabilistic ranking query types, including *U-kRanks* [192, 214], *PT-k* [108] and *Global top-k* [219] (cf. Chapter 10). The efficiency of the proposed approach will be experimentally evaluated in Section 12.5. Section 12.6 will conclude this chapter. The notations used in Chapter 11 will also be used throughout this chapter.

12.2 Efficient Retrieval of the Rank Probabilities

12.2.1 Dynamic Probability Computation

Consider an uncertain object X, defined by m probabilistic observations $X = \{(x_1, P(X = x_1)), \ldots, (x_m, P(X = x_m))\}$ according to the model in Definition 9.2 of Chapter 9. The probability that X is assigned to a given ranking position i ($1 \leq i \leq k$) w.r.t. the distance to a query observation q is equal to the chance that exactly $i-1$ objects $Z \in \mathcal{D} \setminus \{X\}$ are closer to q than the object X. This probability, $P_q(X, i)$, can be computed by aggregating the probabilities $P_q(x, i)$ over all observations $(x, P(X = x))$ of X, as already mentioned in Chapter 11. Formally,

$$P_q(X, i) = \sum_{x \in X} P_q(x, i) \cdot P(X = x). \tag{12.1}$$

[1]The $O(N \cdot log(N))$ factor is due to presorting the observations according to their distances to the query object. If the assumption is made that the observations are already sorted, then the framework can compute the probability distributions for the first k rank positions in $O(k \cdot N)$ time.

Hereby, the probability $P_q(x,i)$ reflects the likelihood that exactly $i-1$ objects $Z \in \mathcal{D}\setminus\{X\}$ are closer to q than the observation x. Contrary to the approach of Chapter 11, which maintains a table to store the probabilities, the more elegant solution is to maintain a list of objects from which observations have been seen so far. This list will be called *Active Object List* (*AOL*) in the following. Using the *AOL*, the *table pruning* of Chapter 11 is implicitly performed, since objects, that have completely been processed or from which no observation has yet been retrieved, do not have to be considered.

The computation of $P_q(X,i)$ is performed in an iterative way, i.e., whenever a new observation x is fetched from the distance browsing $\mathcal{B}$, the probabilities $P_q(x,i)\cdot P(X=x)$ are computed for all ranks $i \in \{1,\ldots,k\}$ and $P_q(X,i)$ is updated accordingly.

The following part will show how to compute the probabilities $P_q(x,i)\cdot P(X=x)$ for all $i \in \{1,\ldots,k\}$ for a given observation $(x, P(X=x))$ of an uncertain object X, which is assumed to be currently fetched from the distance browsing $\mathcal{B}$. For this computation it is required that, for all uncertain objects $Z \in \mathcal{D}\setminus\{X\}$, the probability $P_x(Z)$ that Z is closer to q than the current observation x is known. These probabilities are stored in the *AOL* and can easily be kept updated due to Lemma 11.3 of Chapter 11:

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

In fact, it is only needed to manage in the list the probabilities of those objects for which an observation has already been accessed and for which it is expected to access further observations in the remaining iterations.

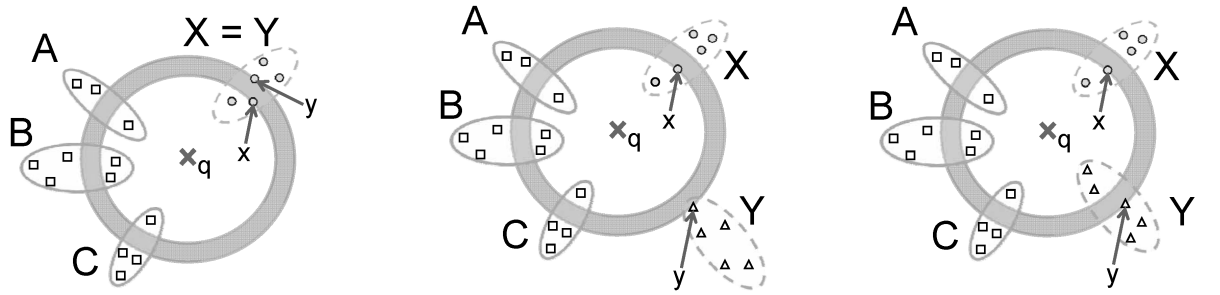
The issue of interest now is how the list *AOL* can be used to efficiently compute the probabilities $P_q(x,i)$. Assume that $(x, P(X=x)) \in X$ is the current observation reported from the distance browsing $\mathcal{B}$. Let $\mathcal{S} = \{Z_1,\ldots,Z_j\}$ be the set of objects which have been seen so far, i.e., for which at least one observation has already been retrieved from $\mathcal{B}$. Furthermore, assume that X has been seen for the first time with the current observation x, but not yet been added to $\mathcal{S}$. According to Lemma 11.4 of Chapter 11, the probability that x appears on ranking position $i+1$ of the first $j+1$ objects seen so far only depends on the event that i out of j objects $Z \in \mathcal{S}$ $(i \le j)$ appear before X, no matter which of these objects satisfies this criterion. Let $P_{i,\mathcal{S},x}$ denote the probability that exactly i objects of $\mathcal{S}$ are closer to q than the observation x. Now, the *Poisson Binomial Recurrence* [147] can be applied:

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

where

$$P_{0,\emptyset,x} = 1 \text{ and } P_{i,\mathcal{S},x} = 0 \text{ if } i<0 \vee i>|\mathcal{S}|. \tag{12.2}$$

For each observation $(x, P(X=x))$ reported from $\mathcal{B}$, it is necessary to apply this recursive function. Specifically, it is needed to compute, for each observation $(x, P(X=x))$, the probabilities $P_{i,\mathcal{S},x}$ for all $i \in \{0,\ldots,\min(k,|\mathcal{S}|)\}$ and for $j = |\mathcal{S}|$ subsets of $\mathcal{S}$. It is further assumed that the ranks $> k$ are neglected. This has a cost factor of $O(k\cdot N)$



(a) Case 1: Previous observation x and current observation y belong to the same object.

(b) Case 2: Observation y is the first returned observation of object Y.

(c) Case 3: Observation y is not the first returned observation of object Y and $X \neq Y$.

Figure 12.1: Cases when updating the probabilities, assuming x was the last processed observation and y is the current one.

per observation retrieved from the distance browsing, leading to a total cost of $O(k \cdot N^2)$. Assuming that k is a small constant and that it is often not required to return a complete ranking, this yields an overall runtime of $O(N^2)$.

The following section will show how to compute each $P_{i,\mathcal{S},x}$ in constant time by utilizing the probabilities computed for the previously accessed observation.

12.2.2 Incremental Probability Computation

Let $(x, P(X = x)) \in X$ and $(y, P(Y = y)) \in Y$ be two observations consecutively returned from the distance browsing. Without loss of generality, let $(x, P(X = x))$ be returned before $(y, P(Y = y))$. The current state assumes that x was the last processed observation, such that $X \in \mathcal{S}$ holds. Each probability $P_{i,\mathcal{S}\setminus\{Y\},y}$ $(i \in \{0, \ldots, \min(k, |\mathcal{S} \setminus \{Y\}|)\})$ 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 backwards. This will turn out to be the main improvement compared to [214], as the new probabilities $P_{i,\mathcal{S}\setminus\{Y\},y}$ are incorporated in the previous results, whereas [214] computes the ranking probabilities from scratch (i.e., all shaded cells of the illustrated matrix in Chapter 11), requiring an update cost of $O(k \cdot N)$.

The following three cases have to be considered, which are illustrated in Figure 12.1. The first two cases are easy to tackle; the third case is the most frequently occurring and challenging one.

- **Case 1:** Both observations belong to the same object, i.e., $X = Y$ (cf. Figure 12.1(a)).
- **Case 2:** Both observations belong to different objects, i.e., $X \neq Y$ and $(y, P(Y = y))$ is the first retrieved observation of object Y (cf. Figure 12.1(b)).
- **Case 3:** Both observations belong to different objects, i.e., $X \neq Y$ and $(y, P(Y = y))$ is *not* the first retrieved observation of object Y (cf. Figure 12.1(c)).

Now, it will be presented how the probabilities $P_{i,\mathcal{S}\setminus\{Y\},y}$ for $i \in \{0,\dots,\min(k,|\mathcal{S}\setminus\{Y\}|)\}$ can be computed in constant time considering the above cases.

In the first case (cf. Figure 12.1(a)), the probabilities $P_x(Z)$ and $P_y(Z)$ of all objects in $Z \in \mathcal{S}\setminus\{X\}$ are equal, because the observations of objects in $\mathcal{S}\setminus\{X\}$ that appear within the distance range of q of y and within the distance range of q and 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. Figure 12.1(b)), it is possible to exploit the fact that $P_{i,\mathcal{S}\setminus\{X\},x}$ does not depend on Y, as y is the first returned observation of Y. At this point, $y \in \mathcal{S}$. Thus, given the probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$, the probability $P_{i,\mathcal{S}\setminus\{Y\},y}$ can easily be computed by incorporating the object X using the recursive Equation (12.2):

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

Since $\mathcal{S}\setminus\{Y,X\} = \mathcal{S}\setminus\{X,Y\}$ and there is no observation of any object in $\mathcal{S}\setminus\{X,Y\}$ which appears within the distance range of q and y but not within the range of q and x (cf. Figure 12.1(b)), similar conditions that held for x can also be assumed for y. Thus, 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 of q and x and, thus, $Y \notin \mathcal{S}\setminus\{X\}$. Now, the above equation can be reformulated:

$$P_{i,\mathcal{S}\setminus\{Y\},y} = P_{i-1,\mathcal{S}\setminus\{X\},x} \cdot P_y(X) + P_{i,\mathcal{S}\setminus\{X\},x} \cdot (1 - P_y(X)). \tag{12.3}$$

All probabilities of the term on the right hand side in Equation (12.3) 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 $i \in \{0,\dots,\min(k,|\mathcal{S}\setminus\{X\}|)\}$.

The third case (cf. Figure 12.1(c)) is the general case which is not as straightforward as the previous two cases and requires special techniques. Again, the assumption is made that the probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$ computed in the previous step for all $i \in \{0,\dots,\min(k,|\mathcal{S}\setminus\{X\}|)\}$ are known. Similarly to Case 2, the probability $P_{i,\mathcal{S}\setminus\{Y\},y}$ can be computed by

$$P_{i,\mathcal{S}\setminus\{Y\},y} = P_{i-1,\mathcal{S}\setminus\{X,Y\},x} \cdot P_y(X) + P_{i,\mathcal{S}\setminus\{X,Y\},x} \cdot (1 - P_y(X)). \tag{12.4}$$

Since the probability $P_y(X)$ is assumed to be known, now the computation of $P_{i,\mathcal{S}\setminus\{X,Y\},x}$ is left for all $i \in \{0,\dots,\min(k,|\mathcal{S}\setminus\{X,Y\}|)\}$ by again exploiting Equation (12.2):

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

which can be resolved to

$$P_{i,\mathcal{S}\setminus\{X,Y\},x} = \frac{P_{i,\mathcal{S}\setminus\{X\},x} - P_{i-1,\mathcal{S}\setminus\{X,Y\},x} \cdot P_x(Y)}{1 - P_x(Y)}. \tag{12.5}$$

Assuming $i = 0$ yields

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

because the probability $P_{-1,\mathcal{S}\setminus\{X,Y\},x} = 0$ by definition (cf. Equation (12.2)). 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 $i \in \{1,\ldots,\min(k,|\mathcal{S}\setminus\{X\}|)\}$ as well as $P_x(Y)$ are available from the previous iteration step, Equation (12.5) can be used to recursively compute $P_{i,\mathcal{S}\setminus\{X,Y\},x}$ for all $i \in \{1,\ldots,\min(k,|\mathcal{S}\setminus\{X,Y\}|)\}$ using the previously computed $P_{i-1,\mathcal{S}\setminus\{X,Y\},x}$. This recursive computation yields all probabilities $P_{i,\mathcal{S}\setminus\{X,Y\},x}$ ($i \in \{0,\ldots,\min(k,|\mathcal{S}\setminus\{X,Y\}|)\}$) which can be used to compute the probabilities $P_{i,\mathcal{S}\setminus\{Y\},y}$ for all $i \in \{0,\ldots,\min(k,|\mathcal{S}\setminus\{X,Y\}|)\}$ according to Equation (12.4).

12.2.3 Runtime Analysis

Building on this case-based analysis for the cost of computing $P_{i,\mathcal{S}\setminus\{X\},x}$ for the currently accessed observation x of an object X, it is now possible to prove that the RPD can be computed at cost $O(k \cdot N)$. The following lemma suggests that the incremental cost per observation access is $O(k)$.

Lemma 12.1 *Let $(x, P(X = x)) \in X$ and $(y, P(Y = y)) \in Y$ be two observations consecutively returned from the distance browsing $\mathcal{B}$. Without loss of generality, the assumption is made that the observation $(x, P(X = x))$ was returned in the last iteration in which the probabilities $P_{i,\mathcal{S}\setminus\{X\},x}$ have been computed for all $i \in \{0,\ldots,\min(k,|\mathcal{S}\setminus\{X\}|)\}$. In the next iteration, in which $(y, P(Y = y))$ is fetched, the probabilities $P_{i,\mathcal{S}\setminus\{Y\},y}$ for all $i \in \{0,\ldots,\min(k,|\mathcal{S}\setminus\{Y\}|)\}$ can be computed in $O(k)$ time and space.*

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 $i \in \{0,\ldots,\min(k,|\mathcal{S}\setminus\{Y\}|)\}$. 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 $i \in \{0,\ldots,\min(k,|\mathcal{S}\setminus\{Y\}|)\}$ can be computed according to Equation (12.3) taking $O(k)$ time. This assumes that the $P_{i,\mathcal{S}\setminus\{X\},x}$ have to be stored for all $i \in \{0,\ldots,\min(k,|\mathcal{S}\setminus\{Y\}|)\}$, requiring at most $O(k)$ space.

In Case 3, it is first needed to compute and store the probabilities $P_{i,\mathcal{S}\setminus\{X,Y\},x}$ for all $i \in \{0,\ldots,\min(k,|\mathcal{S}\setminus\{X,Y\}|)\}$ using the recursive function in Equation (12.5). 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 $i \in \{0,\ldots,\min(k,|\mathcal{S}\setminus\{Y\}|)\}$ according to Equation (12.4) which takes at most $O(k)$ time and space. □

After giving the runtime evaluation of the processing of one single observation, it is now possible to extend the cost model for the whole query process. According to Lemma 12.1, the assumption can be made that each observation can be processed in constant time if k is chosen to be constant. Under the assumption that the total number of observations

Approach	No precomputed $\mathcal{B}$	Precomputed $\mathcal{B}$
Soliman et al. [192]	exponential	exponential
Chapter 11 [45]	exponential	exponential
Yi et al. [214]	$O(k \cdot\ N^2)$	$O(k \cdot\ N^2)$
This chapter [43]	$O(N \cdot\ log(N) + k \cdot\ N)$	$O(k \cdot\ N)$

Table 12.1: Runtime complexity comparison between the probabilistic ranking approaches; N and k denote the database size and the ranking depth, respectively.

in the database is linear in the number of database objects, a runtime complexity would be obtained which is linear in the number of database objects, more exactly $O(k \cdot\ N)$, where k is the specified depth of the ranking. Up to now, the utilized data model assumes that the pre- and postprocessing steps of the proposed framework require at most linear runtime. Since the postprocessing step only includes an aggregation of the results in order to obtain a final ranking output, the linear runtime complexity of this step is guaranteed. Now, the runtime of the initial (certain) observation ranking has to be examined, which is the preprocessing step needed to initialize the distance browsing $\mathcal{B}$. Similarly to the assumptions that hold for the competitors [45, 192, 214], it can also be assumed that the observations are already sorted, which would involve linear runtime cost also for this module. However, for the general case where a distance browsing has to be initialized first, the runtime complexity of this module would increase to $O(N \cdot\ log(N))$. As a consequence, the total runtime cost of the proposed approach (including distance browsing) sums up to $O(N \cdot\ log(N) + k \cdot\ N)$. An overview of the computation cost is given in Table 12.1.

The cost required to solve the object-based rank probability problem is similar to that required to solve the observation-based rank probability problem. The solution based on observations additionally only requires to build the sum over all observation-based rank probabilities, which can be done on-the-fly without additional cost. Furthermore, the cost required to build a final unambiguous ranking (e.g., the rankings proposed in Section 12.4 or those proposed in Chapter 11) from the rank probabilities can be neglected. The final ranking can also be computed on-the-fly by simple aggregations of the corresponding (observation-based) rank probabilities.

Regarding the space complexity of an RPD of size $O(k \cdot\ N)$, a vector of length k has to be stored for each object in the database. In addition, it is required to store the AOL of a size of at most $O(N)$, yielding a total space complexity of $O(k \cdot\ N + N) = O(k \cdot\ N)$. [214] directly combines the probability computations with the output of U-kRanks with a space complexity of $O(N)$. The approach presented this chapter solves the problem of computing the RPD, i.e., the bipartite graph problem introduced in Chapter 9, and can apply the solution to any definite ranking output. Details will be provided in Section 12.4. To compute an RPD according to the current definition, [214] requires $O(k \cdot\ N)$ space as well.

Algorithm 7 Probabilistic Ranking Algorithm: probRanking($\mathcal{B}$, q)

Require: $\mathcal{B}$, q
1: $AOL \leftarrow \emptyset$
2: $result \leftarrow$ Matrix of 0s // size = $N \cdot k$
3: $p\text{-}rank_x \leftarrow [0, \ldots, 0]$ // length k
4: $p\text{-}rank_y \leftarrow [0, \ldots, 0]$ // length k
5: $y \leftarrow \mathcal{B}$.next()
6: updateAOL(y)
7: $p\text{-}rank_x[0] \leftarrow 1$
8: add $p\text{-}rank_x$ to the first line of *result*
9: **while** $\mathcal{B}$ is not empty **and** $\exists p \in p\text{-}rank_x : p > 0$ **do**
10: $x \leftarrow y$
11: $y \leftarrow \mathcal{B}$.next()
12: updateAOL(y)
13: **if** $Y = X$ **then**
14: {**Case 1** (cf. Figure 12.1(a))}
15: $p\text{-}rank_y \leftarrow p\text{-}rank_x$
16: **else if** $Y \notin AOL$ **then**
17: {**Case 2** (cf. Figure 12.1(b))}
18: $P(X) \leftarrow AOL$.getProb(X)
19: $p\text{-}rank_y \leftarrow$ dynamicRound($p\text{-}rank_x$, $P_y(X)$)
20: **else**
21: {**Case 3** ($Y \neq X$, cf. Figure 12.1(c))}
22: $P(X) \leftarrow AOL$.getProb(X)
23: $P(Y) \leftarrow AOL$.getProb(Y)
24: $adjustedProbs \leftarrow$ adjustProbs($p\text{-}rank_x$, $P_x(Y)$)
25: $p\text{-}rank_y \leftarrow$ dynamicRound($adjustedProbs$, $P_y(X)$)
26: **end if**
27: Add $p\text{-}rank_y$ to the next line of *result*
28: $p\text{-}rank_x \leftarrow p\text{-}rank_y$
29: **end while**
30: **return** *result*

12.3 Probabilistic Ranking Algorithm

12.3.1 Algorithm Description

The pseudocode of the probabilistic ranking algorithm is illustrated in Algorithm 7 and provides the implementation details of the previously discussed steps. The algorithm requires a query object q and a distance browsing operator $\mathcal{B}$ that allows to iteratively access the observations sorted in ascending order of their distance to a query object.

First, the AOL is initialized, a data structure that contains one tuple $(X, P(X))$ for

each object X that

- has previously been found in $\mathcal{B}$, i.e., at least one observation of X has been processed
- and has not yet been completely processed, i.e., at least one observation of X has yet to be found,

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

- updateAOL(x): adds the probability $P(X = x)$ of the observation $x \in X$ to $P(X)$, where X is the object that x belongs to.
- getProb(X): returns the aggregated probability of object X ($P(X)$).

For efficient retrieval and update, it is mandatory that the position of a tuple $(X, P(X))$ in the *AOL* 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 by directly storing with each object X the information about the probability $P(X)$, both requiring an additional space cost of $O(N)$. Another structure to keep is *result*, a matrix that contains, for each observation x that has been retrieved from $\mathcal{B}$, and each ranking position i the probability $P_q(x, i)$ that x is located on ranking position i. In order to get an object-based rank probability, observations belonging to the same object can be aggregated, using Equation (12.1).

Additionally, two arrays *p-rank*$_x$ and *p-rank*$_y$ are initialized, 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 $i \in \{0, \ldots, k\}$. $x \in X$ is the observation found in the previous iteration and $y \in Y$ is the observation found in the current iteration (cf. Figure 12.1).

In line 5, the algorithm starts by fetching the first observation, which is closest to the query observation q in the database. A tuple containing the corresponding object as well as the probability of this observation is added to the *AOL*.

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

$$P_{0,\mathcal{S}\setminus\{X\},x} = P_{0,\emptyset,x} = 1 \text{ and } P_{i,\mathcal{S}\setminus\{X\},x} = P_{i,\emptyset,x} = 0$$

for $i \geq 1$ by definition (cf. Equation (12.2)). This simply reflects the fact that the first retrieved observation from $\mathcal{B}$ is always on rank 1. *p-rank*$_y$ is implicitly assigned to *p-rank*$_x$. Then, the first iteration of the main algorithm begins by fetching the next observation from $\mathcal{B}$ (line 11). Now, the three cases explained in Subsection 12.2.2 have to be distinguished.

In the first case (line 13), both the previous and the current observation refer to the same object. As explained in Subsection 12.2.2, there is nothing to do in this case, since $P_{i,\mathcal{S}\setminus\{X\},x}$=$P_{i,\mathcal{S}\setminus\{Y\},y}$ for all $i \in \{0 \ldots, k-1\}$.

In the second case (line 16), the current observation refers to an object that has not been seen yet. As explained in Subsection 12.2.2, only an additional iteration of the dynamic-programming algorithm has to be applied (cf. Equation (12.2)). This dynamic iteration

Algorithm 8 Dynamic Iteration for Observation y: dynamicRound(*oldRanking*, $P_y(X)$)

Require: *oldRanking* (intermediate result without object X)
Require: $P_y(X)$ (probability that object X is closer to q than observation y)
  $newRanking \leftarrow [0, \ldots, 0]$ {length k}
  $newRanking[0] \leftarrow oldRanking[0] \cdot (1 - P_y(X))$
  for $i = 1 \rightarrow k - 1$ **do**
    $newRanking[i] \leftarrow oldRanking[i-1] \cdot P_y(X) + oldRanking[i] \cdot (1 - P_y(X))$
  end for
  return *newRanking* (result including object X)

Algorithm 9 Probability Adjustment: adjustProbs(*oldRanking*, $P_x(Y)$)

Require: *oldRanking* (intermediate result including object Y)
Require: $P_x(Y)$ (prob. that object Y is closer to q than the last retrieved observation x)
  $adjustedProbs \leftarrow [0, \ldots, 0]$ {length k}
  $adjustedProbs[0] \leftarrow \frac{oldRanking[0]}{1-P_x(Y)}$
  for $i = 1 \rightarrow k - 1$ **do**
    $adjustedProbs[i] \leftarrow \frac{oldRanking[i]-adjustedProbs[i-1]\cdot P_x(Y)}{1-P_x(Y)}$
  end for
  return *adjustedProbs* (intermediate result at observation $y \in Y$, excluding object Y from the current result)

dynamicRound is shown in Algorithm 8 and is used here to incorporate the probability that X is closer to q than y into $p\text{-}rank_y$ in a single iteration of the dynamic algorithm.

In the third case (line 20), the current observation 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 Subsection 12.2.2, the influence of previously retrieved $y \in Y$ on $P_{i,\mathcal{S}\setminus\{X\},x}$ has to be filtered out first, and then $P_{i,\mathcal{S}\setminus\{X,Y\},x}$ has to be computed. This is performed by the probability adjustment algorithm *adjustProbs* (cf. Algorithm 9) utilizing the technique explained in Subsection 12.2.2. Using the $P_{i,\mathcal{S}\setminus\{X,Y\},x}$, the algorithm then computes the $P_{i,\mathcal{S}\setminus\{Y\},y}$ performing a single iteration of the dynamic algorithm like in Case 2.

In line 27, the computed ranking for observation y is added to the result. If the application (i.e., the ranking method) requires objects to be ranked instead of observations, then $p\text{-}rank_y$ is used to incrementally update the probabilities of Y for each rank.

The algorithm continues fetching observations from the distance browsing operator $\mathcal{B}$ and repeats this case analysis until either no more samples are left in $\mathcal{B}$ or until an observation is found with a probability of 0 for each of the first k positions. In the latter case, there exist k objects that are closer to q with a probability of 1, i.e. for which all observations have been retrieved, and the computation can be stopped, because the same k objects must be closer to q than all further observations in the database that have not yet been retrieved by the distance browsing $\mathcal{B}$.

12.4 Probabilistic Ranking Approaches

The method proposed in Subsection 12.2.2 efficiently computes, for each uncertain observation x and each ranking position $i \in \{1, \ldots, k\}$ the probability that x has the ith rank. However, most applications require a unique ranking, i.e., each object (or observation) is uniquely assigned to exactly one rank. Various top-k query approaches have been proposed generating unambiguous rankings from probabilistic data which are called *probabilistic ranking queries*. The question at issue is how the framework proposed in this and in the previous chapter can be exploited in order to significantly accelerate probabilistic ranking queries. This section will show that the framework supports and significantly boost the performance of the state-of-the-art probabilistic ranking queries. Specifically, this is demonstrated by applying state-of-the-art ranking approaches, including *U-kRanks*, *PT-k* and *Global top-k*.

The following ranking approaches are based on the x-relation model [25]. As mentioned in Chapter 9, the x-relation model conceptionally corresponds to the uncertainty model used here, where an observation corresponds to a tuple and an uncertain object correspond to an x-tuple. In the following, the terms *object* and *observation* will again be used.

12.4.1 U-*k*Ranks

The *U-kRanks* approach [192] reports the most likely observation for each rank i, i.e., the observation that is most likely to appear on rank i over all possible worlds. This is essentially the same definition as proposed in *PRank* in [155]. The approach proposed in [192] has exponential runtime. The runtime has been reduced to $O(N^2 \cdot k)$ time in [214]. Using the proposed framework, the problem of U-kRanks can be solved in $O(N \cdot log(N) + k \cdot N)$ time requiring the same space complexity as follows.

First, the framework is used to create the RPD in $O(N \cdot log(N) + k \cdot N)$ as explained in the previous section. Then, for each rank i ($1 \leq i \leq k$), the observation $\arg\max_x(p\text{-}rank_q(x, i))$ that has the highest probability of appearing on rank i can be found in $O(k \cdot N)$. This is performed by finding for each rank i the observation which has the highest probability to be assigned to rank i. Obviously, in this problem definition, a single observation x may appear on more than one ranking position, or it may not appear in the result at all (cf. also the example in Chapter 11). For example, in Figure 12.2, observation a is ranked on both ranks 1 and 2, while observation b is ranked nowhere. Therefore, alternatively, the ranking semantics *PRQ_MC* and *PRQ_MAC* from Chapter 11 can be considered here.

The total runtime for U-kRanks has, thus, been reduced from $O(k \cdot N^2)$ to $O(N \cdot log(N) + k \cdot N)$, which is $O(N \cdot log(N))$ if k is assumed to be constant. *PRQ_MC* and also *PRQ_MAC*, which has to regard all prior ranks $1 \leq j < i$ in addition, can be solved with the same computational requirements.

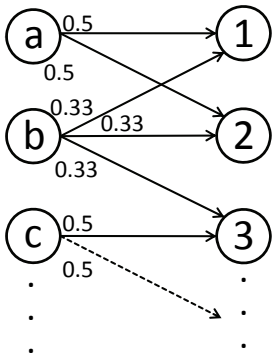


Figure 12.2: Small example extract of a rank probability distribution (RPD) as produced by the proposed framework.

12.4.2 PT-*k*

Similarly to the definition of *PRQ_MAC* in Chapter 11, the *Probabilistic Threshold Top-k* (*PT-k*) query problem [108] fixes the drawback of the previous definition by aggregating the probabilities of an observation x appearing on rank k or higher. Given a user-specified probability threshold p, PT-k returns all observations that have a probability of at least p of being on rank k or higher. In this definition, the number of results is not limited by k, but depends on the threshold parameter p. The model of PT-k consists of a set of observations and a set of generation rules that define mutual exclusiveness of observations. Each observation occurs in one and only one generation rule. This model conceptionally corresponds to the x-relation model (with disjoint x-tuples). PT-k computes all result observations in $O(k \cdot\ N)$ time, while also assuming that the observations are already presorted. Thus, this yields a total runtime of $O(N \cdot\ log(N) + k \cdot\ N)$. The framework can be used to solve the PT-k problem in the following way.

The RPD is created in $O(k \cdot\ N)$ as explained in the previous section. For each observation x, the probability that x appears at position k or higher is computed (in $O(k \cdot\ N)$). Formally, all observations $x \in \mathcal{D}$ are returned for which the condition

$$\{x \in \mathcal{D} : \sum_{i=1}^{k} P_q(x, i) > p\}$$

holds. As seen in Figure 12.2, this probability can simply be computed by aggregating all probabilities of an observation to be ranked at k or higher. For example, for $k = 2$ and $p = 0.5$, the obtained results are a and b. For $p = 0.1$, further observations may be in the result, because there must be further observations (from observations that are left out here for simplicity) with a probability greater than 0 to ranks 1 and 2, since the probability of the respective edges associated with these ranks does not sum up to 1 yet.

The proposed framework is only able to match, not to beat the runtime of PT-k. However, using the proposed approach, it is possible to additionally return the ranking order, instead of just the top-k set.

12.4.3 Global Top-k

Global top-k [219] is very similar to PT-k. It ranks the observations by their top-k probability, and then takes the top-k of these. This approach has a runtime of $O(k \cdot N^2)$. The advantage here is that, unlike in PT-k, the number of results is fixed, and there is no user-specified threshold parameter. Here, the ranking order information that has been acquired in the PT-k using the proposed framework to solve Global top-k in $O(N \cdot log(N) + k \cdot N)$ time, can be exploited.

The framework is used to create the RPD in $O(N \cdot log(N) + k \cdot N)$ as explained in the previous section. For each observation x, the probability that x appears at position k or higher is computed (in $O(k \cdot N)$) like in PT-k. Then, the k observations with the highest probability are returned in $O(k \cdot log(k))$.

12.5 Experimental Evaluation

12.5.1 Datasets and Experimental Setup

Extensive experiments were performed to evaluate the performance of the proposed probabilistic ranking approach proposed in this chapter. The parameters that were evaluated are the database size N (the number of uncertain objects), the ranking depth k and the *degree of uncertainty* (UD) as defined below. In the following, the ranking framework proposed in this chapter is briefly denoted by **PSR**.

The probabilistic ranking was applied to a scientific semi-real-world dataset *SCI* and several artificial datasets *ART_X* of varying size and degree of uncertainty. All datasets are based on the discrete uncertainty model according to Definition 9.2 in Chapter 9.

The *SCI* dataset is a set of 1,600 objects, which was synthetically created based on a data set comprising 1,600 environmental time series[2]. In the original time series data set, each object consists of 48 ten-dimensional environmental sensor measurements taken on one single day, one per 30 minutes. The ten measured attributes were 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. An uncertain object X was then created based on one single time series as follows by incorporating a real as well as a synthetic component. Addressing the real component, each sensor measurement can be considered as an observation in the feature space which is spanned by the ten dimensions enumerated above. The dimensions were normalized within the interval [0,1] to give each attribute the same weight. Thus, a time series is translated to a ten-dimensional spatial object with 48 alternative observations $x_i, i \in \{1, \ldots, 48\}$. Finally, addressing the synthetic component, each $x_i \in X$ has to possess a likelihood to represent X: $P(X = x_i)$. Here, the probability for each x_i was set to $\frac{1}{48}$, summing up in an overall probability of 1; thus, the dataset complies with the uncertain data model of Definition 9.2 of Chapter 9. It is important to note that the method of

[2]The environmental time series have been provided by the Bavarian State Office for Environmental Protection, Augsburg, Germany (`http://www.lfu.bayern.de/`).

creating the dataset does not imply a specific degree of uncertainty due to the availability of the attributes values. The *SCI* dataset was used to evaluate the scalability and the ranking depth.

The *ART_1* dataset was used for the scalability experiments and consists of 1,000,000 objects. Here, each uncertain object is represented by a set of 20 three-dimensional observations that are uniformly distributed within a three-dimensional hyperrectangle. The degree of uncertainty of this object then corresponds to the size (i.e., the side length) of this hyperrectangle. All rectangles are uniformly distributed within a $10 \times 10 \times 10$ feature space. For the evaluation of the performance w.r.t. the ranking depth and the degree of uncertainty, two collections of datasets, *ART_2* and *ART_3*, were applied. Each dataset of the collections is composed of 10,000 objects with 20 observations each and differs in the degree of uncertainty of the corresponding objects. In *ART_2*, the observations of an object are also uniformly distributed within a three-dimensional hyperrectangle. In *ART_3*, the observations of an object follow a three-dimensional Gaussian distribution. The datasets of *ART_3* vary in the degree of uncertainty as well. For this dataset, the degree of uncertainty simply denotes the standard deviation of the Gaussian distribution of the objects.

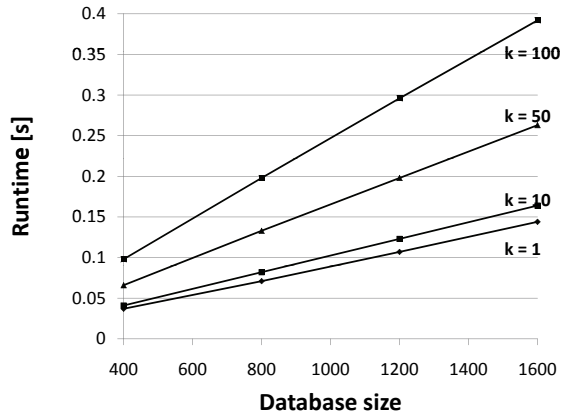
The degree of uncertainty is interesting in the performance evaluation, since it is expected to have a significant influence on the runtime. The reason is that a higher degree of uncertainty obviously leads to a higher overlap between the objects which influences the size of the active object list AOL (cf. Section 12.3) 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 expected to lead to a higher runtime. This characteristic will be experimentally evaluated in Subsection 12.5.3.

12.5.2 Scalability

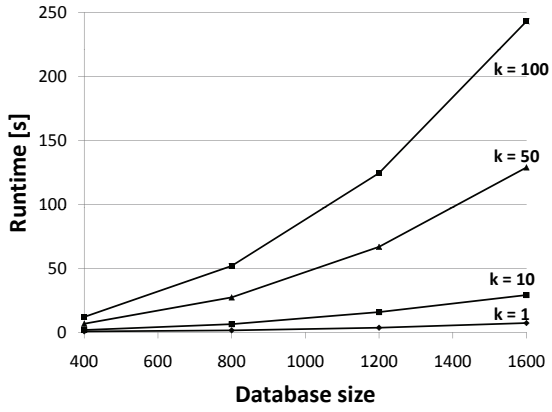
This section gives an overview of the experiments regarding the scalability of **PSR**. The obtained results are compared to the rank probability computation based on dynamic programming as proposed by Yi et al. in [214]. This method, in the following denoted by **YLKS**, has been the best prior approach for solving the U-kRanks (cf. Table 12.1). For a fair comparison, the **PSR** framework was used to compute the same (observation-based) rank probability problem as described in Section 12.2. As mentioned in Subsection 12.2.3, the cost required to solve the object-based rank probability problem is similar to that required to solve the observation-based rank probability problem. Furthermore, the cost required to build a final unambiguous ranking (e.g., the rankings proposed in Section 12.4) from the rank probabilities can be neglected, because this ranking can also be computed on-the-fly by simple aggregations of the corresponding (observation-based) rank probabilities.

For the sorting of the distances of the observations to the query point, a tuned quicksort adapted from [26] was used. This algorithm offers $O(N \cdot log(N))$ performance on many datasets that cause other quicksort algorithms to degrade to quadratic runtime.

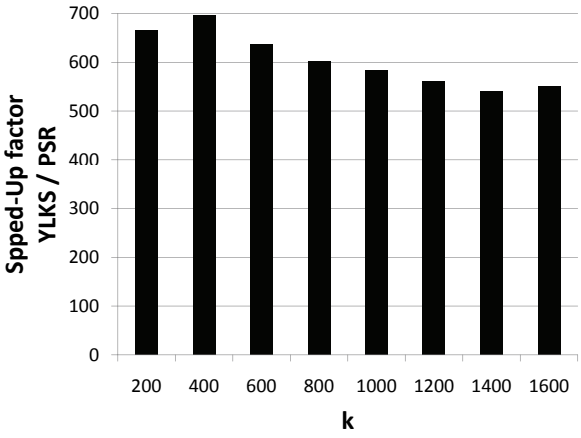
The results of the first scalability tests on the real-world dataset *SCI* are depicted in Figures 12.3(a) and 12.3(b). It can be observed that the required runtime for computing



(a) **PSR**.



(b) **YLKS**.

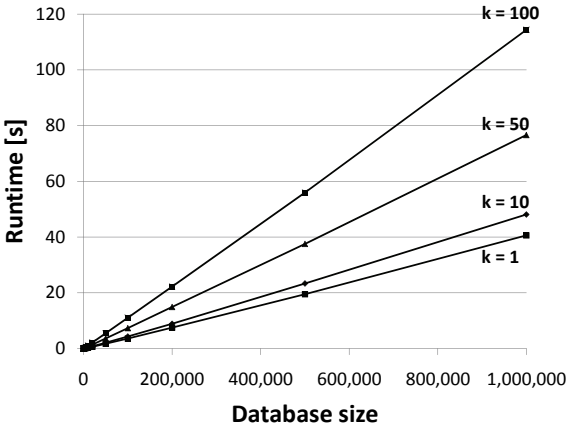


(c) Speed-up gain w.r.t. k.

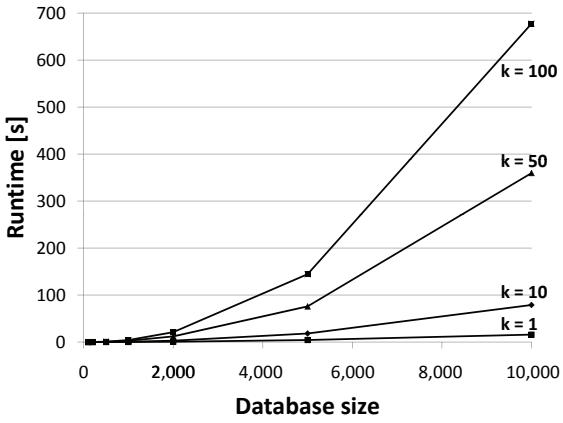
Figure 12.3: Scalability evaluated on *SCI* for different values of k.

the probabilistic ranking using the **PSR** framework increases linearly in the database size, whereas **YLKS** has a runtime that is quadratic in the database size with the same parameter settings. It can also be observed that this effect persists for different settings of k. The effect of the $O(N \cdot log(N))$ sorting of the distances of the observations is insignificant on this relatively small dataset. The direct speed-up of the rank probability computation using **PSR** in compared to **YLKS** is depicted in Figure 12.3(c). It shows, for different values of k, the speed-up factor, which is defined as the ratio $\frac{runtime(\mathbf{YLKS})}{runtime(\mathbf{PSR})}$ describing the performance gain of **PSR** w.r.t. **YLKS**. It can be observed that, for a constant number of objects in the database ($N = 1,600$), the ranking depth k has no impact on the speed-up factor. This can be explained by the observation that both approaches scale linearly in k.

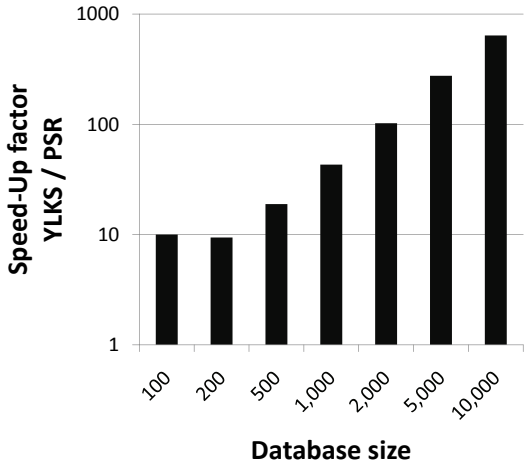
The next experiment evaluates the scalability of the database size based on the *ART_1* dataset. The results of this experiment are depicted in Figures 12.4(a) and 12.4(b). The former shows that the approach proposed in this chapter performs ranking queries in a



(a) **PSR**.



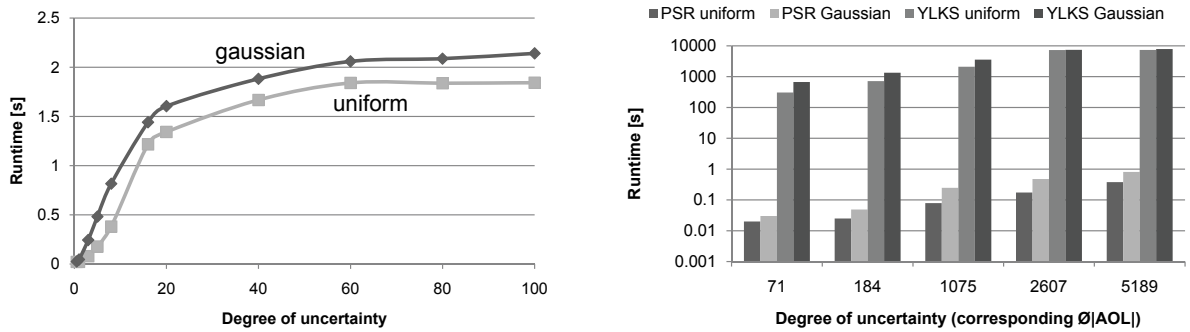
(b) **YLKS**.



(c) Speed-up gain w.r.t. N ($k = 100$).

Figure 12.4: Scalability evaluated on *ART_1* for different values of k.

reasonable time of less than 120 seconds, even for very large database containing 1,000,000 and more objects, each having 20 observations (thus having a total of 20,000,000 observations). An almost perfect linear scale-up can be seen despite of the $O(N \cdot log(N))$ cost for sorting the database. This is due to the very efficient quicksort implementation in [26] that the experiments have shown to require only slightly worse than linear time. Furthermore, it can be observed that, due to its quadratic scaling, the **YLKS** algorithm is already inapplicable for relatively small databases of size 5,000 or more. The direct speed-up of the rank probability computation using **PSR** in comparison to **YLKS** for a varying database size is depicted in Figure 12.4(c). Here, it can be observed that the speed-up of **PSR** in comparison to **YLKS** increases linearly with the size of the database, which is consistent with the runtime analysis in Subsection 12.2.3.



(a) Evaluation of **PSR** by an increasing degree of uncertainty.

(b) **YLKS** vs. **PSR** in a logarithmic scale w.r.t. different $\varnothing(|AOL|)$ values.

Figure 12.5: Runtime w.r.t. the degree of uncertainty on *ART_2* (uniform) and *ART_3* (Gaussian).

12.5.3 Influence of the Degree of Uncertainty

The next experiment varies the degree of uncertainty (cf. Subsection 12.5.1) on the datasets *ART_2* and *ART_3*. In the following experiments, the ranking depth is set to a fixed value of $k = 100$. As previously discussed, a varying *UD* leads to an increase of the overlap between the observations of the objects and thus, objects will remain in the *AOL* for a longer time. The influence of the *UD* depends on the probabilistic ranking algorithm. This statement is underlined by the experiments shown in Figure 12.5. It can be seen in Figure 12.5(a) that **PSR** scales superlinear in the *UD* at first, until a maximum value is reached. This maximum value is reached when the *UD* becomes so large that the observations of an object cover the whole vector space. In this case, objects remain in the *AOL* until almost the whole database is processed in most cases due to the increased overlap of observations. In this case of extremely high uncertainty, almost no objects can be pruned for a ranking position, thus slowing down the algorithm by several orders of magnitude. It is also worth noting that, in the used 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 Gaussian distribution with standard deviation σ in each dimension is generally larger than a hyperrectangle with a side length of σ in each dimension. A runtime comparison of **YLKS** and **PSR** w.r.t. the average *AOL* size is depicted in Figure 12.5(b) for both the uniformly and the Gaussian distributed datasets. The *UD* has a similar influence on both **YLKS** and **PSR**.

12.5.4 Influence of the Ranking Depth

The influence of the ranking depth k on the runtime performance of the probabilistic ranking method **PSR** is studied in the next experiment. As illustrated in Figure 12.6, where the experiments were performed using both the *SCI* and the *ART_2* dataset, the influence of an increasing k yields a linear effect on the runtime of **PSR**, but does not depend on

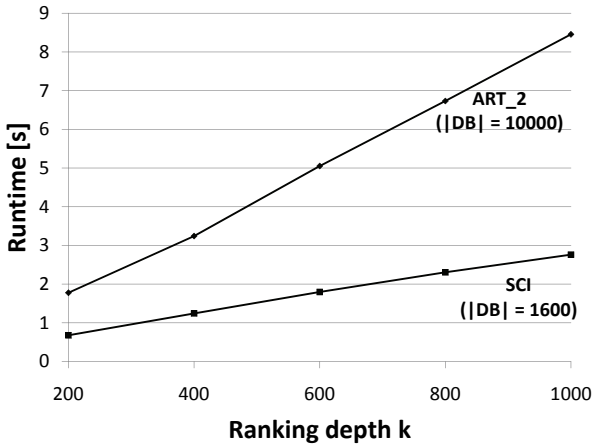


Figure 12.6: Runtime using **PSR** on *SCI* and *ART_2*.

the type of the dataset. This effect can be explained by the fact that each iteration of Case 2 or Case 3 of the incremental probability computation (cf. Subsection 12.2.2) requires a probability computation for each ranking position $i \in \{0, \ldots, k\}$. The overall runtime requirements on *ART_2* is higher that on *SCI* due to the different database sizes, which could already be observed in Subsection 12.5.2.

12.5.5 Conclusions

The experiments presented in this section show that the theoretical analysis of the proposed approach, which was given in Subsection 12.2.3, can be confirmed empirically on both artificial and real-world data. The performance studies showed that the proposed framework computing the rank probabilities indeed reduces the quadratic runtime complexity of state-of-the-art approaches to linear complexity. The cost required to presort the observations are neglected in the settings due to the tuned quicksort. It could be shown that the proposed approach scales very well even for large databases. The speed-up gain of the proposed approach w.r.t. the rank depth k has shown to be constant, which proves that both approaches scale linearly in k. Furthermore, it could be observed that the proposed approach is applicable for databases with a high degree of uncertainty (i.e., the variance of the observation distribution).

12.6 Summary

The approach proposed in this chapter achieved a significant improvement of the runtime complexity of computing probabilistic ranking queries via incremental processing, overall yielding a linear complexity under specific assumptions. The concepts proposed in this chapter were theoretically and empirically proved to be superior to all existing approaches, as the *Poisson Binomial Recurrence* was improved by a significant case study.

Chapter 13

Continuous Probabilistic Inverse Ranking on Uncertain Streams

13.1 Introduction

The two previous chapters focused on probabilistic ranking queries; they studied semantics of ranking outputs and finally provided efficient solutions to compute the rank probability distribution w.r.t. the distance to a query object, i.e., for each uncertain object $X \in \mathcal{D}$ and each ranking position i ($1 \leq i \leq k$) the probability of an object to occur on rank i, for the first k ranks. The semantics of similarity ranking is, finally, to return an ordered list of "best" objects w.r.t. a particular criterion, where, for spatial data, the distance to a query object is the most common one.

This chapter will focus on the *probabilistic inverse ranking* (PIR) query. While the PIR problem has been tackled for static data [149, 158], this chapter will propose an extension for uncertain streaming data, i.e., when the data changes with elapsing time. Given a stream of uncertain objects, a user-defined score function f_{score} that ranks the objects and a user-defined (uncertain) query object Q, a PIR query monitors the possible ranks of this object, i.e., it computes all possible ranks of Q associated with a probability, which corresponds to the rank probability distribution restricted to Q. Apart from considering only static data, the semantics of a PIR query according to the definition in this chapter semantically differs from [158], where the output of a PIR query consists of all possible ranks for a (certain) query object q, for which q has a probability higher than a given threshold. Furthermore, the approach of computing the expected inverse ranks [149] also differs from a semantic point of view, bearing some significant disadvantages. For example, an object that has a very high chance to be on rank 1, may indeed have an expected rank far from rank 1, and may not be in the result using expected ranks. Thus, no conclusion can be made about the actual rank probabilities if the expected rank is used, since the expected rank is an aggregation that drops important information.

The PIR query is important for many real-world applications including financial data analysis, sensor data monitoring and multi-criteria decision making where one might be

(Chances; Risk)			
Conf.	Analyst I (50%)	Analyst II (30%)	Analyst III (20%)
Stock I	(10; 6)	(12; 8)	(10; 9)
Stock II	(5; 4)	(4; 4)	(6; 5)
Stock III	(4; 1)	(5; 2)	(5; 1)

Table 13.1: Chances and risk predictions by three analysts for three stocks.

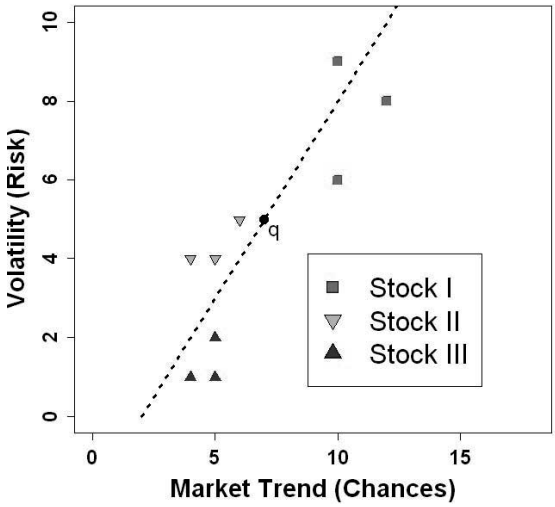


Figure 13.1: Stock prediction chart.

interested in the identification of the rank (significance) of a particular object among peers.

Example 13.1 *Consider the exemplary application illustrated in Table 13.1. A financial decision support system monitors diverse prognostic attributes of a set of three stocks, e.g., predicted market trend (chances) and volatility (risk), which are used to rate the profitability of the stocks w.r.t. a given score function. As it can be observed, the chance and risk estimations are not unique among different analysts, and each analyst is given a different confidence level. Figure 13.1 shows graphically the three stocks with their respective analyst predictions and the (certain) query object q which consists of only one observation. Here, it is assumed to be given a score function defined as* $f_{score} = (Chances - Risk)$*. The dotted line in Figure 13.1 denotes all observations* x *where* $f_{score}(x) = f_{score}(q)$*, i.e., all points that have the same score as* q*. Any observation located to the right of this line has a higher score than* q*, while any observation to the left has a lower score. Therefore, it can safely be assumed that Stock II has a lower score than* q*, while Stock III certainly has a higher score than* q*. However, the relative ranking of Stock I w.r.t.* q *is uncertain. While two of three analysts (with a total confidence of* 80%*) would rank Stock I higher than* q*, the third analyst would rank it lower. Thus, the PIR query for* q *returns that* q *is on rank 2 with a probability of* 20%*, on rank 3 with a probability of* 80% *and definitely not on rank 1 or 4. This result can be used to answer questions like "Given a score function, what is the likelihood that a query stock* q *is one of the top-3 best stocks?". The problem studied in this chapter is how to efficiently update these likelihoods when the analysts release new estimations on a ticker stream.*

The rest of this chapter is organized as follows: Section 13.2 will formally define the problem of probabilistic inverse ranking on data streams. The approach for solving the problem

efficiently will be described in Section 13.3. Section 13.4 will generalize the problem by additionally considering uncertain queries. The efficiency of the proposed approach will be experimentally evaluated in Section 13.5. Finally, Section 13.6 will conclude this chapter.

13.2 Problem Definition

Similarly to the previous chapters, the solution that will be provided in this chapter adopts the uncertain object model of Definition 9.2 of Chapter 9, where each uncertain object X is assigned to m alternative observations and corresponds to exactly one *x-tuple* T, which includes m alternative tuples t. It is assumed that $\sum_{t \in T} P(t) = 1$. For the problem of inverse ranking, this assumption means no loss of generality, since existential uncertainty can be modeled by simply adding to T an additional observation with a probability $1 - \sum_{t \in T} P(t)$ and a score of $-\infty$ (that is a distance of ∞ to the query). For example, the m observations of an uncertain object are derived from m sources of information (sensors). In the stock example (cf. Example 13.1), the sources correspond to the assessments of the analysts. The most frequent notations throughout this chapter are slightly different from the previous chapters and, thus, summarized in Table 13.2 on the following page.

The management of continuous measurements within a "dynamic" uncertain database, i.e., where new observations arrive subsequently and need to be processed, leads to the utilization of uncertain stream models. A *probabilistic stream* is defined analogously to [113].

Definition 13.1 (Probabilistic Stream) *A* probabilistic stream *is a data stream* $S = [x_0, \ldots, x_t, \ldots]$ *in which each observation* x_t *encodes a random variable reported at time* t *from the stream, corresponding to an object update. In particular, each observation* x_t *has the form* (X, L)*, where* X *is an object ID and* L *is a location sequence of length* $|L|$*. Each element* $l \in L$ *contains a location* $\in \mathbb{R}^d$ *and a probability* $P(l) \in [0, 1]$*. In addition, the assumption is made that* $\sum_{l \in L} P(l) = 1$*, i.e., it is assumed that the object does not have an existential uncertainty, i.e., that object* X *is existentially certain. This implies the definition of an appropriate time-fading function for the probabilities* $P(l)$*.*

An uncertain database instantiated from a probabilistic stream is defined as follows.

Definition 13.2 (Probabilistic Stream Database) *A* probabilistic stream database *is an uncertain database connected to at least one probabilistic stream. Each stream observation* $x_t = (X, L)$ *at time* t *denotes an update of the uncertain object* $X \in \mathcal{D}$*, where* X *may also be a new object. Therefore, at time* t*, the* x-relation *describing object* X *is replaced by the new location distribution* L *coming from the stream.*

This probabilistic stream database model is very general and can be easily adapted to simulate popular stream models: the *sliding window model* of size w can be simulated by imposing the following constraint to the probabilistic stream: for any two stream observations $x_t = (X, L_t)$ and $x'_t = (X, L_{t'}), t < t'$ of the same object X, it holds that, if there is no other stream observation between time t and time t' concerning the same object X, $L_{t'}$ is derived from L_t by

Notation	Description
$\mathcal{D}$	an uncertain database
N	the cardinality of $\mathcal{D}$
k	the ranking depth that determines the number of ranking positions of the inverse ranking query result
S	a probabilistic stream
w	the window size, i.e., the number of currently regarded recent observations
q	a query observation in respect to which a probabilistic inverse ranking is computed
X, Y, Z	uncertain stream objects, each corresponding to a finite set of alternative observations
$P_q^t(X)$	the probability that object X has a higher score than q at time t
$P^t(q,i)$	the result of the inverse ranking at time t: the probability that q is on rank i at time t
$P_{i,j,q}^t$	the probability that, out of j processed objects, exactly i objects have a higher score than q at time t
P_i^t	the result of the PBR at time t: the probability that i objects have a higher score than q at time t, if all objects X for which $P_q^t(X)=1$ are ignored
$\hat{P}_i^t$	the adjusted result of the PBR at time t: identical to P_i^t except that the effect of the object that changes its position at time $t+1$ is removed from the calculation
C^t	the number of objects X at time t for which $P_q^t(X)=1$

Table 13.2: Table of notations used in this chapter.

- adding exactly one new observation to L_t, and
- removing the oldest observation of L_t if $|L_t| > w$.

The probabilities $P(l), l \in L_t$ are often set to $P(l) = \frac{1}{|L_t|}$, but other distributions can be used. In particular, more recently observed observations can be given a higher probability to obtain the *weighted sliding window model*. Additionally, the infinite sliding window model is obtained by setting $w = \infty$. In this chapter, the stream model is left abstract, as the proposed solutions are applicable for any model.

Now, as the data model is available, is the probabilistic inverse ranking query is defined as follows.

Definition 13.3 (Probabilistic Inverse Ranking Query) *Given an uncertain database $\mathcal{D}$ of size N, a query object Q and a score function*

$$f_{score} : \mathcal{D} \to \mathbb{R}_o^+.$$

Assuming that only the first k ranks are of interest, a probabilistic inverse ranking query $PIR(Q)$ *returns for each* $i \in \{1, \ldots, k\}$ *the probability* $P^t(Q, i)$ *that* q *is on rank* i *w.r.t. the score function* f_{score}*, i.e., the probability that there exist exactly* $i - 1$ *objects* $X \in \mathcal{D}$ *such that* $f_{score}(X) > f_{score}(Q)$ *at time* t*.*

Given a set of N uncertain objects and a probabilistic stream S as defined above, the problem is to compute and update, for a given query object Q and a given score function f_{score}, the result of $PIR(Q)$ at each time t, i.e., after each object update. The challenge is to ensure that this can be done correctly in terms of the *Possible Worlds Semantics* [145] (cf. Chapter 9), and highly efficiently to allow online processing of the probabilistic stream S. Since the number of possible worlds at a time t is exponential in the number N of uncertain stream objects at time t, these two challenges are conflicting. The following section will propose an approach to compute $PIR(q)$, i.e., the probabilistic inverse ranking for a single observation $q \in Q$, in $O(k \cdot N)$ from scratch, and to update it in $O(k)$ when a new update is fetched from the stream. In addition, Section 13.4 will show how the result of $PIR(Q)$ can be efficiently updated if the query object Q consists of more than one observation and, thus, is itself a stream object that changes frequently.

13.3 Probabilistic Inverse Ranking (PIR)

13.3.1 The PIR Framework

Consider an uncertain stream database $\mathcal{D}$ of size N, a query observation q, a score function f_{score} and a positive integer k. The proposed algorithm basically consists of two modules:

- **Module 1**: The *initial computation* of the PIR that computes, for each rank $i \in \{1, \ldots, k\}$, the probability $P^t(q, i)$ that q is ranked on position i at the initial time t, when the query is issued. Subsection 13.3.2 will show how this can be performed in $O(k \cdot N)$ time.

- **Module 2**: The *incremental stream processing* that updates $PIR(q)$ at time $t + 1$, given the PIR at time t. Therefore, the probabilities $P^{t+1}(q, i)$ that q is ranked on position i at time $t + 1$ have to be computed given the $P^t(q, i)$, $i \in \{1, \ldots, k\}$. In Subsection 13.3.3, it will be shown how this update can be done in $O(k)$ time.

13.3.2 Initial Computation

For each object $X \in \mathcal{D}$, let $P_q^t(X)$ be the probability that X has a higher rank than q at time t, i.e., $P_q^t(X) = P(f_{score}(X) > f_{score}(q))$. These probabilities can be computed in a single database scan. The $P_q^t(X)$ can be processed successively by means of the *Poisson Binomial Recurrence* (*PBR*) [147], as proposed for probabilistic ranking in the previous chapters. Let $P_{i,j,q}^t$ be the probability that, out of the j objects processed so far, exactly i objects have a higher score than q. This probability depends only on two events:

- $i-1$ out of the first $j-1$ processed objects have a higher score than q *and* X has a higher score than q.
- i out of the first $j-1$ processed objects have a higher score than q *and* X does *not* have a higher score than q.

This observation and the assumption of independence between stream objects can be used to formulate the following PBR:

$$P^t_{i,j,q} = P^t_{i-1,j-1,q} \cdot P^t_q(X) + P^t_{i,j-1,q} \cdot (1 - P^t_q(X)) \tag{13.1}$$
$$\text{with } P^t_{0,0,q} = 1 \text{ and } P^t_{i,j,q} = 0 \text{ if } i < 0 \vee i > j.$$

When the last object of the database is processed, i.e., $j = N$, then $P^t_{i,j,q} = P^t_{i,N,q} \overset{Definition}{=} P^t(q, i+1)$.[1] Computing the $P^t(q, i+1)$ for $0 \leq i < k$ yields the probabilistic inverse ranking. In each iteration, the computation of any $P^t_{i,j,q}$ can be omitted where $i \geq k$, since any ranks greater than k are not relevant, and thus, the cases where at least k objects have a higher score than q are not of interest. In total, for each $0 \leq i < k$ and each $1 \leq j \leq N$, $P^t_{i,j,q}$ has to be computed resulting in an $O(k \cdot N)$ time complexity.

Equation (13.1) is only required for objects X for which $0 < P^t_q(X) < 1$. Objects X for which $P^t_q(X) = 0$ can safely be ignored in the initial computation, since they have no effect on the $P^t(q, i)$. For objects X for which $P^t_q(X) = 1$, a counter C^t is used that denotes the number of these objects. Thus, when X is encountered in the initial computation, the PBR can be avoided and C^t is incremented. This optimization will be referred to as *0-1-optimization* in the experimental evaluation. The probabilities obtained from the PBR by ignoring objects for which $P^t_q(X) = 1$ are denoted as P^t_i, $0 \leq i < k$.

The probabilistic inverse ranking can be obtained from the P^t_i $(0 \leq i < k)$ and from C^t as follows:

$$P^t(q, i+1) = \begin{cases} P^t_{i-C^t} \text{ , for } C^t \leq i \leq C^t + k \\ 0 \text{ , otherwise} \end{cases} \tag{13.2}$$

Example 13.2 *Given a database containing four objects $X_1, \ldots, X_4$ and an inverse ranking query with query observation q and $k = 2$, assume that $P^t_q(X_1) = 0.1$, $P^t_q(X_2) = 0$, $P^t_q(X_3) = 0.6$ and $P^t_q(X_4) = 1$. To compute the initial inverse ranking, the first object to process is X_1, using Equation* (13.1)*:*

$$P^t_{0,1,q} = P^t_{-1,0,q} \cdot P^t_q(X_1) + P^t_{0,0,q} \cdot (1 - P^t_q(X_1)) = 0 \cdot 0.1 + 1 \cdot 0.9 = 0.9,$$
$$P^t_{1,1,q} = P^t_{0,0,q} \cdot P^t_q(X_1) + P^t_{1,0,q} \cdot (1 - P^t_q(X_1)) = 1 \cdot 0.1 + 0 \cdot 0.9 = 0.1.$$

Next, X_2 is processed, but notice that $P^t_q(X_2) = 0$, so X_2 can be skipped. Then, object X_3 requires an additional iteration of Equation (13.1)*:*

$$P^t_{0,2,q} = P^t_{-1,1,q} \cdot P^t_q(X_3) + P^t_{0,1,q} \cdot (1 - P^t_q(X_3)) = 0 \cdot 0.6 + 0.9 \cdot 0.4 = 0.36.$$

[1] The event that i objects have a higher score than q corresponds to the event that q is on rank $i+1$.

$$P^t_{1,2,q} = P^t_{0,1,q} \cdot P^t_q(X_3) + P^t_{1,1,q} \cdot (1 - P^t_q(X_3)) = 0.9 \cdot 0.6 + 0.1 \cdot 0.4 = 0.58.$$

$P^t_{2,2,q}$ does not need to be computed, since $k = 2$. The next object to process is X_4. Since $P^t_q(X_4) = 1$, only C^t has to be incremented to 1. At this point, the computation is finished. The obtained results are

$$P^t_0 = 0.36 \text{ and } P^t_1 = 0.58.$$

To get the final inverse ranking at time t, it is possible to use Equation (13.2) *to obtain*

$$P^t(q, 1) = P^t_{0-1} = P^t_{-1} = 0 \text{ and}$$

$$P^t(q, 2) = P^t_{1-1} = P^t_0 = 0.36.$$

13.3.3 Incremental Stream Processing

A naïve solution would apply the PBR (cf. Equation (13.1)) whenever a new observation of object X is fetched from the stream. However, the expensive update which is linear in the size of the database would make online stream processing impractical for large databases. The following part shows how $P^{t+1}(q, i)$ can be updated for $1 \leq i \leq k$ in constant time using the results of the previous update iteration.

Without loss of generality, let X be the object for which a new position information is returned by the stream at time $t + 1$. $P^t_q(X)$ ($P^{t+1}_q(X)$) denotes the old (new) probability that X has a higher score than q.

The update algorithm uses two phases:

- **Phase 1:** Removal of the effect of the old value distribution of the uncertain object X, that is, removal of the effect of the probability $P^t_q(X)$ from the result $P^t_i, 0 \leq i < k$. This yields an intermediate result $\hat{P}^{t+1}_i, 0 \leq i < k$.

- **Phase 2:** Incorporation of the new value distribution of the uncertain object X, that is, including the probability $P^{t+1}_q(X)$ in the intermediate result $\hat{P}^{t+1}_i, 0 \leq i < k$, obtained in Phase 1.

Phase 1: Removal of $P^t_q(X)$

The following cases w.r.t. $P^t_q(X)$ have to be considered:

- **Case 1:** $P^t_q(X) = 0$. This case occurs if X is a new object or if it is certain that X has a lower score than q at time t. Thus, nothing has to be done to remove the effect of $P^t_q(X)$: $\hat{P}^{t+1}_i = P^t_i$.

- **Case 2:** $P^t_q(X) = 1$, i.e., it is certain that X has a higher score than q at time t. In this case, it is just needed to decrement C^t by one to remove the effect of $P^t_q(X)$. Thus, $\hat{P}^{t+1}_i = P^t_i$ and $C^{t+1} = C^t - 1$.

- **Case 3:** $0 < P_q^t(X) < 1$, i.e., it is uncertain whether X has a higher score than q at time t. In order to remove the effect of $P_q^t(X)$ on all P_i^t $(0 \leq i < k)$, the iteration that most recently applied the PBR (cf. Equation (13.1)) has to be considered, which was performed at time $t' \leq t-1$ and used to obtain P_i^t, $0 \leq i < k$. Let Y be the object that was incorporated in this iteration:

$$P_i^t = P_{i-1}^{t'} \cdot P_q^t(Y) + P_i^{t'} \cdot (1 - P_q^t(Y)),$$

where $P_i^{t'}$ describes the probability that i objects have a score higher than q at time t', if (in addition to all objects Z for which $P_q^t(Z) = 1$) Y is ignored. Now it can be observed that the probabilities P_i^t $(0 \leq i < k)$ are not affected by the order in which the objects are processed within the recursion. In particular, the probabilities P_i^t do not change if the objects are processed in an order that processes X last. Thus, the obtained probability is

$$P_i^t = \hat{P}_{i-1}^t \cdot P_q^t(X) + \hat{P}_i^t \cdot (1 - P_q^t(X)).$$

This can be resolved to

$$\hat{P}_i^t = \frac{P_i^t - \hat{P}_{i-1}^t \cdot P_q^t(X)}{1 - P_q^t(X)}. \tag{13.3}$$

Setting $i = 0$ yields

$$\hat{P}_0^t = \frac{P_0^t}{1 - P_q^t(X)}, \tag{13.4}$$

because the probability $\hat{P}_{-1}^t$ that exactly -1 objects have a higher score than q is 0 by definition (cf. Equation (13.1)). Since the probabilities P_i^t for $0 \leq i < k$ are known from the previous stream processing iteration, $\hat{P}_0^t$ can be easily computed using Equation (13.4). Now it is possible to inductively compute $\hat{P}_{i+1}^t$ by using $\hat{P}_i^t$ for any i and exploiting Equation (13.3).

Phase 2: Incorporation of $P_q^{t+1}(X)$

In Phase 2, the same cases have to be considered:

- **Case 1:** $P_q^{t+1}(X) = 0$, i.e., object X has no influence on the result at time $t+1$. Nothing has to be done. Thus, $P_i^{t+1} = \hat{P}_i^{t+1}$.
- **Case 2:** $P_q^{t+1}(X) = 1$, i.e., it is certain that object X has a higher score than q. Thus, $C^{t+1} = C^t + 1$ and $P_i^{t+1} = \hat{P}_i^{t+1}$.
- **Case 3:** $0 < P_q^{t+1}(X) < 1$, i.e., the new probability for X to be ranked higher than q, i.e. $P_q^{t+1}(X)$, can be incorporated to compute the new probabilistic inverse ranking by an additional iteration of the PBR:

$$P_i^{t+1} = \hat{P}_{i-1}^{t+1} \cdot P_q^{t+1}(X) + \hat{P}_i^{t+1} \cdot (1 - P_q^{t+1}(X)).$$

Example 13.3 *Reconsider Example 13.2, where time t yielded $C^t = 1$, $P_0^t = 0.36$ and $P_1^t = 0.58$. Now, assume that at time $t+1$ object X_3 changes its probability from 0.6 to 0.2, i.e., $P_q^t(X_3) = 0.6$ and $P_q^{t+1}(X_3) = 0.2$. Phase 1 starts using Case 3. The use of Equation* (13.4) *yields*

$$\hat{P}_0^t = \frac{P_0^t}{1 - P_q^t(X_3)} = \frac{0.36}{0.4} = 0.9.$$

Going further, Equation (13.3) *yields*

$$\hat{P}_1^t = \frac{P_1^t - \hat{P}_0^t \cdot P_q^t(X_3)}{1 - P_q^t(X_3)} = \frac{0.58 - 0.9 \cdot 0.6}{0.4} = 0.1.$$

This completes Phase 1. In Phase 2, Case 3 is chosen, which yields

$$P_0^{t+1} = \hat{P}_{-1}^t \cdot P_q^{t+1}(X_3) + \hat{P}_0^t \cdot (1 - P_q^{t+1}(X_3)) = 0 \cdot 0.2 + 0.9 \cdot 0.8 = 0.72 \text{ and}$$

$$P_1^{t+1} = \hat{P}_0^t \cdot P_q^{t+1}(X_3) + \hat{P}_1^t \cdot (1 - P_q^{t+1}(X_3)) = 0.9 \cdot 0.2 + 0.1 \cdot 0.8 = 0.26.$$

This completes the update step (C^t remains unchanged, i.e., $C^{t+1} = C^t$). The result is obtained analogously to Example 13.2 using Equation (13.2)*:*

$$P^{t+1}(q,1) = P_{0-1}^{t+1} = P_{-1}^{t+1} = 0 \text{ and}$$

$$P^{t+1}(q,2) = P_{1-1}^{t+1} = P_0^{t+1} = 0.72.$$

Now, at time $t+2$, object X_4 is assumed to change its probability from 1 to 0: in Phase 1, Case 2 is used and C_t is decremented from 1 to 0 to obtain $C^{t+1} = 0$. In Phase 2, Case 1 is used and nothing has to be done. The obtained probabilities are

$$P_0^{t+2} = \hat{P}_0^{t+1} = P_0^{t+1} = 0.72 \text{ and}$$

$$P_1^{t+2} = \hat{P}_1^{t+1} = P_1^{t+1} = 0.26.$$

The result after using Equation (13.2) *is*

$$P^{t+2}(q,1) = P_{0-0}^{t+2} = P_0^{t+2} = 0.72 \text{ and}$$

$$P^{t+2}(q,2) = P_{1-0}^{t+2} = P_1^{t+2} = 0.26.$$

Example 13.3 shows why it is important to maintain k probability values at each point of time: even though some of the k probabilities may not be required to obtain the result, they may be required to obtain the result at a later time.

Regarding the computational complexity, the following holds for both Phase 1 and Phase 2: Case 1 and Case 2 have a cost of $O(1)$, since either nothing has to be done or only C^t has to be incremented or decremented. Case 3 has a total cost of $O(k)$ leading to a total runtime of $O(k)$ in the update step.

13.4 Uncertain Query

The previous section assumed that the query q is fixed, i.e., it consists of only one observation with a certain position in $\mathbb{R}^d$. This section will consider the case in which the query is also given as an uncertain stream object, consisting of several observations that are reported over time. Similarly to the database objects, it is assumed that the query object Q is represented by a set of m alternative observations $q \in Q$ at time t. The probabilistic inverse ranking query $PIR(Q)$ w.r.t. an uncertain query object Q can be computed by aggregating the PIR query results w.r.t. each observation q of Q, as it is done with the computation of an object-based probabilistic ranking in Chapter 11. Formally,

$$P^t(Q,i) = \sum_{q \in Q} P^t(q,i) \cdot P(Q=q)$$

for all $q \in Q$, where $P(Q=q)$ denotes the probability that Q is located at observation q and $P^t(q,i)$ is the probability that observation q is on rank i. $P^t(q,i)$ can be computed and updated as proposed in Section 13.3.

In the current scenario, the stream may return new position information of Q as well. Then, the probabilities of all objects being ranked before Q may change. Consequently, the inverse ranking result usually needs to be recomputed from scratch, using the technique shown in Subsection 13.3.2. However, in most applications, the position of an object only changes slightly. Therefore, the probability of other objects to have a higher score than Q normally does not change for most objects. This property is exploited as follows.

Let Q be the query object with alternative observations $q \in Q$ at time t and let $f^t_{score\_min}(Q)$ and $f^t_{score\_max}(Q)$ denote the minimum and maximum among all possible scores derived from the observations of Q at time t. The following part assumes that a new query observation is reported from the stream at time $t+1$:

Lemma 13.1 *If $f^t_{score\_min}(Q) \leq f^{t+1}_{score\_min}(Q)$, then it holds that, for any object X with $P^t_Q(X) = 0$, $P^{t+1}_Q(X) = 0$, assuming X has not been updated at time $t+1$.*

Proof.

$$\textit{Assumption: } f^t_{score\_min}(Q) \leq f^{t+1}_{score\_min}(Q) \tag{13.5}$$

$$\textit{Assumption: } \forall x \in X : f^t_{score}(x) = f^{t+1}_{score}(x) \tag{13.6}$$

$$\textit{Assumption: } P^t_Q(X) = 0 \tag{13.7}$$

$$(\textit{Equation (13.7)}) \Leftrightarrow \forall q \in Q, \forall x \in X : f^t_{score}(q) > f^t_{score}(x)$$

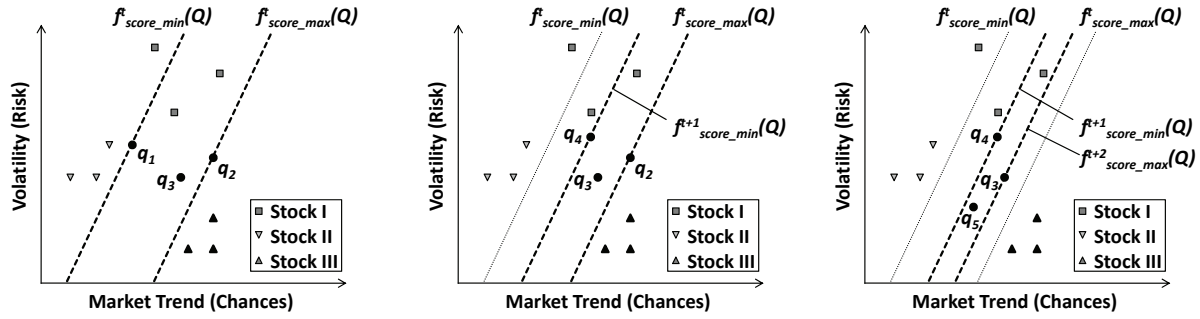
$$\Leftrightarrow \forall x \in X : f^t_{score\_min}(Q) > f^t_{score}(x) \tag{13.8}$$

$$\textit{Definition: } \forall q \in Q, \forall x \in X : f^{t+1}_{score}(q) \geq f^{t+1}_{score\_min}(Q)$$

$$\overset{Equation\ (13.5)}{\geq} f^t_{score\_min}(Q) \overset{Equation\ (13.8)}{\geq} f^t_{score}(x) \overset{Equation\ (13.6)}{=} f^{t+1}_{score}(x)$$

$$\Rightarrow \forall q \in Q, \forall x \in X : f^{t+1}_{score}(q) \geq f^{t+1}_{score}(x) \Leftrightarrow P^{t+1}_Q(X) = 0$$

□



(a) State at time t. (b) State at time $t+1$. (c) State at time $t+2$.

Figure 13.2: Changes of $f_{score\_min}(Q)$ and $f_{score\_max}(Q)$.

Lemma 13.2 *If $f^t_{score\_max}(Q) \geq f^{t+1}_{score\_max}(Q)$, then for any object X with $P^t_Q(X) = 1$ it holds that $P^{t+1}_Q(X) = 1$.*

Proof. *Analogous to Lemma 13.1.* □

The following example for both cases is illustrated in Figure 13.2.

Example 13.4 *Consider a slight modification of the stock Example (cf. Example 13.1). In Figure 13.2(a), the current state of the observations at time t is visualized. Here, it is certain that Stock II has a lower score than Q, whereas Stock III certainly has a higher score than Q, as the set of observations of both objects do not overlap with the score range of Q. However, the ranking position of Stock I w.r.t. Q is uncertain, as the observations overlap with the score range of Q. Now, at time $t+1$, a new observation q_4 of Q is fetched from the stream (cf. Figure 13.2(b)). Assuming $w = 3$, the oldest observation q_1 is removed, such that $f^{t+1}_{score\_min}(Q) > f^t_{score\_min}(Q)$. In is obvious that $P^{t+1}_Q(Stock\ II) = P^t_Q(Stock\ II) = 0$, since*

$$f^{t+1}_{score\_max}(Stock\ II) = f^t_{score\_max}(Stock\ II) < f^t_{score\_min}(Q) < f^{t+1}_{score\_min}(Q).$$

Thus, the probabilities of Q w.r.t. Stock II do not have to be updated. However, an update w.r.t. Stock I is needed, since

$$0 < P^{t+1}_Q(Stock\ I) < 1 \text{ and } 0 < P^t_Q(Stock\ I) < 1.$$

Another update of Q at time $t+2$ decreases its maximum score, such that $f^{t+2}_{score\_max}(Q) < f^{t+1}_{score\_max}(Q)$ (cf. Figure 13.2(c)). $P^{t+2}_Q(Stock\ III) = P^{t+1}_Q(Stock\ III) = 1$, since

$$f^{t+2}_{score\_min}(Stock\ III) = f^{t+1}_{score\_min}(Stock\ III) > f^{t+1}_{score\_max}(Q) > f^{t+2}_{score\_max}(Q).$$

Thus, the probabilities of Q w.r.t. Stock III do not have to be updated. Here again, an update w.r.t. Stock I is performed, since

$$0 < P^{t+2}_Q(Stock\ I) < 1 \text{ and } 0 < P^{t+1}_Q(Stock\ I) < 1.$$

The above lemmata allow to reduce the number of objects that have to be considered for the recomputation of the inverse ranking at time $t+1$. Especially, if $f^t_{score\_min}(Q) \leq f^{t+1}_{score\_min}(Q) \wedge f^t_{score\_max}(Q) \geq f^{t+1}_{score\_min}(Q)$, then it is only needed to compute $P^{t+1}_q(X)$ for those objects $X \in \mathcal{D}$ for which $P^t_Q(X) \notin \{0,1\}$. For the remaining objects $Z \in \mathcal{D} \setminus \{X\}$, it is necessary to update $P^t_Q(Z)$ and the inverse ranking probabilities considering the cases outlined in Subsection 13.3.3. The effectiveness of this optimization scheme highly depends on the *degree of uncertainty* of the objects. The experiments in Section 13.5 will show that the number of objects that can be pruned from the computation of the inverse ranking can be very large.

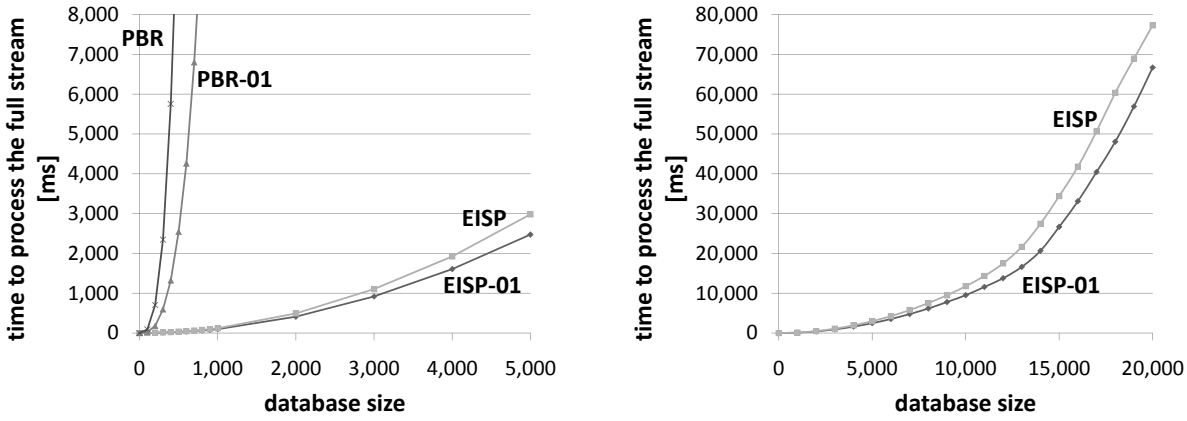
A very drastic change of the position of the query object may, in the worst case, cause all probabilities $P^t_Q(X), X \in \mathcal{D}$ to change. The incremental computation of Section 13.3 requires two steps: the removal of the effect of $P^t_Q(X)$ and the incorporation of $P^{t+1}_Q(X)$ for any object $X \in \mathcal{D}$ that changed its probability of having a higher score than Q. In contrast, a computation from scratch requires only one computation for each $X \in \mathcal{D}$: the incorporation of $P^{t+1}_Q(X)$. Therefore, it is wise to switch to a full recomputation of the PIR if more than $\frac{N}{2}$ objects change their probability.

13.5 Experimental Evaluation

13.5.1 Datasets and Experimental Setup

The experiments that will be presented in the following Subsections 13.5.2 to 13.5.4 used a synthetic dataset modeling a data stream with observations of two-dimensional objects. The location of an object X at time t is modeled by w observations of a Gaussian distributed random variable $\mathcal{X}$ maintained in an array called *sample buffer*. For each $X \in \mathcal{D}$, the mean $E(\mathcal{X})$ follows a uniform $[-10, 10]$-distribution in each dimension. The probabilistic stream S contains, for each $X \in \mathcal{D}$, exactly $m = 10$ observations, which are randomly shuffled into the stream. Once a new observation of an object X is reported by the stream, it is stored in the sample buffer of X by replacing the least recently inserted one. Three parameters were tuned in order to evaluate the performance of the incremental PIR method described in Section 13.3: the database size N (default $N = 10,000$, Subsection 13.5.2), the degree of uncertainty of the objects, which is, in this chapter, reflected by the standard deviation σ of uncertain observations belonging to the same object (default $\sigma = 5$, Subsection 13.5.3), and the sample buffer size w (Subsection 13.5.4). For the scalability experiments, w was set to 3. The evaluation of σ was performed with $w = m = 10$. An additional experiment evaluates the influence of an uncertain query object on the performance of the incremental PIR method (Subsection 13.5.5). Finally, Subsection 13.5.6 will examine the scalability issues on a real-world dataset.

The proposed approach will be denoted by **EISP** (Efficient Inverse Stream Processing). As a comparison partner serves the implementation of an algorithm based on the PBR (abbreviated by **PBR**) as proposed by [214] that uses Equation (13.1) at each point of time where the stream provides a new observation. An additional evaluation examines the



(a) **PBR** vs. **EISP**.

(b) **EISP** vs. **EISP-01**.

Figure 13.3: Scalability of the PIR approaches (full processing).

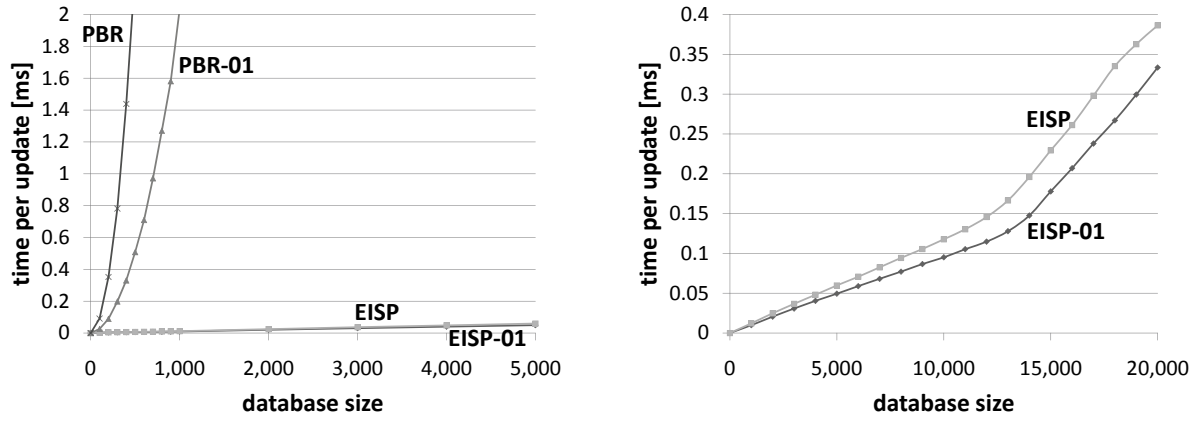
effect of the strategy proposed in Section 13.3 to avoid the computations w.r.t. all objects X with a probability $P_q^t(X) \in \{0, 1\}$ of having a higher score than the query observation q (*0-1-optimization*). **EISP-01** and **PBR-01** will denote the versions of **EISP** and **PBR**, respectively, that use the *0-1-optimization*.

As the existing PIR solutions provided in [149, 158] are only designed for static data and moreover semantically differ from the solution provided in this chapter (cf. Section 13.1), they have not been considered as comparison partners for the experimental evaluation.

13.5.2 Scalability

The first experiment will evaluate the scalability of **EISP**, **PBR**, **EISP-01** and **PBR-01** w.r.t. the database size N. k was chosen to be equal to N, because if k is chosen to be constant and N is scaled up, the number of objects that certainly have a higher score than q will eventually reach k. In this case, the *0-1-optimization* will immediately notice that q cannot possibly be at one of the first k positions and will prune the computation. Then, **EISP-01** and **PBR-01** will have no further update cost. The results are illustrated in Figures 13.3 and 13.4.

Figure 13.3 illustrates the total time required to process the whole stream, i.e., all $m \cdot N$ object updates. It can be observed that all four algorithms show a superlinear time complexity to process the whole stream. Using *0-1-optimization* leads to an improvement in the runtime. As the number of uncertain objects (i.e., the objects in the database for which it is uncertain whether they have a higher score than q and, thus, cannot be removed by the *0-1-optimization*) increases as well as the number of certain objects, a linear speed-up gain is achieved using the *0-1-optimization*. These observations can be explained by the runtime requirements of **PBR** and **PBR-01** of $O(N^3)$ and that of **EISP** and **EISP-01** of $O(N^2)$ to process the whole stream.



(a) **PBR** vs. **EISP**.

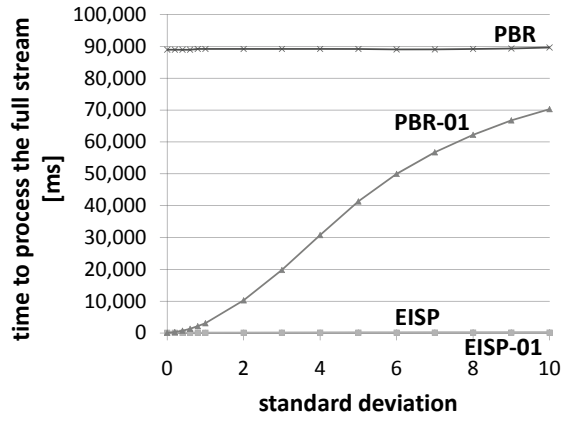
(b) **EISP** vs. **EISP-01**.

Figure 13.4: Scalability of the PIR approaches (single update).

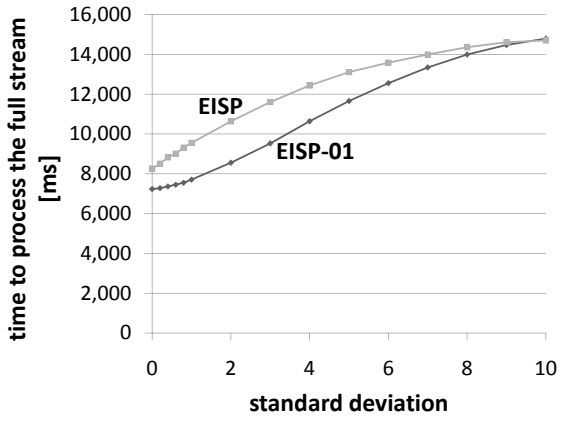
A more detailed evaluation of the update cost in each iteration is illustrated in Figure 13.4. Here, the average time required for an update is shown. The update cost of both **PBR** and **PBR-01** grows fast with N. This is explained by the quadratic cost of $O(N^2)$ of the PBR at each update step (recall that $k = N$ was chosen). On the other hand, the update cost of $O(N)$ of **EISP** is linear to the number of database objects in this experiment (due to $k = N$). Here, the *0-1-optimization* has a high influence on **PBR**, but a smaller effect on **EISP**, especially for $N \leq 5,000$. The effect of the *0-1-optimization* may seem low for **EISP**, but, in the experiments, the total time required for an update was measured; this includes the time required to fetch a new location from the stream, compute its score, and recompute the total probability that the respective object has a higher score than q. This overhead is naturally required for any approach.

13.5.3 Influence of the Degree of Uncertainty

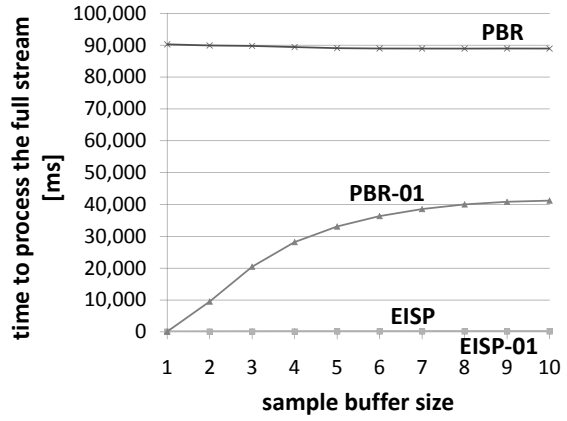
The next experiment will examine the effect of the degree of uncertainty (i.e., the standard deviation σ) on the distribution of the observations. Here, the total time required to process the whole stream was examined. The results are depicted in Figures 13.5(a) and 13.5(b). As **PBR** has to process all objects in each iteration of the inverse ranking, there is no influence of σ when this method is used (cf. Figure 13.5(a)). The *0-1-optimization* reduces the runtime complexity working with low standard deviations, as, in this case, many objects do not overlap with the score function and can therefore be neglected in each iteration. However, with an increasing value of σ, the cost of **PBR-01** approaches that of **PBR**, as the uncertainty ranges are spread over a greater range of the data space. **EISP** and **EISP-01** outperform the other methods by several orders of magnitude. Figure 13.5(b) shows that, for a small value of σ, there is a significant effect of the *0-1-optimization*. This becomes evident considering that the time overhead required to process the stream is more



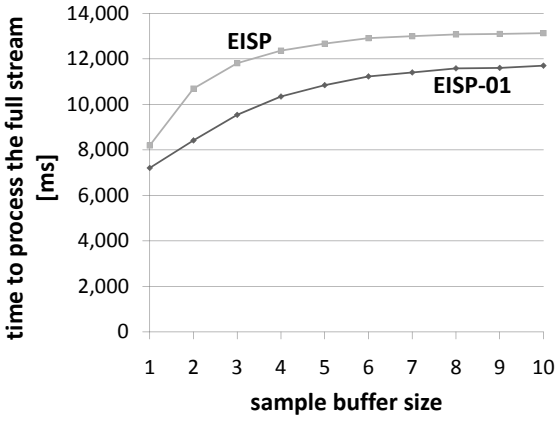
(a) Eval. of σ, **PBR** vs. **EISP**, $N = 1,000$.



(b) Eval. of σ, **EISP** vs. **EISP-01**, $N = 10,000$.



(c) Eval. of w, **PBR** vs. **EISP**, $N = 1,000$.



(d) Eval. of w, **EISP** vs. **EISP-01**, $N = 10,000$.

Figure 13.5: Runtime w.r.t. the standard deviation σ and the sample buffer size w.

than 7,000 ms in this experiment. The reason is that, for $\sigma = 0$, there exists no uncertainty, and, thus, all objects always have a probability of either 0 or 1 of having a higher score than q. Thus, Cases and 2 (cf. Section 13.3) are used in each update step and the PBR is never required. For $\sigma \geq 10$, most objects X have a probability $0 < P_q^t(X) < 1$ of having a higher score than q. Thus, Case 3 is used in each iteration and C^t (the number of objects for which $P_q^t(X) = 1$) approaches 0.

13.5.4 Influence of the Sample Buffer Size

Next, the total stream processing time was evaluated w.r.t. the sample buffer size w. Figures 13.5(c) and 13.5(d) illustrates that w has an impact on all inverse ranking methods. Again, using **PBR**, the number of considered observations only influences the required runtime if the *0-1-optimization* is applied (cf. Figure 13.5(c)). If w increases, the probability that an object X has both observations with a higher and smaller score than q increases,

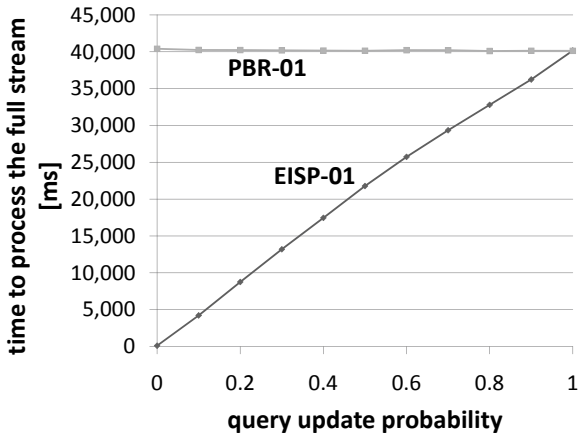


Figure 13.6: Runtime w.r.t. the probability of updating the query object ($N = 1,000$).

i.e., it is uncertain whether $f_{score}(q) > f_{score}(X)$. Figure 13.5(d) shows that, even for $w = 10$, a relatively high performance gain is obtained using the *0-1-optimization*, since the observations remain in the extent of their probabilistic distribution. Thus, for many objects X, $f_{score}(q) > f_{score}(X)$ can be decided even for a large w.

13.5.5 Uncertain Query

Finally, this subsection evaluates the case that the query q is given as an uncertain stream object, now denoted by Q. As described in Section 13.4, the whole inverse ranking may have to be recomputed by the **PBR** method if a position update of Q occurs. For this case, the performance of the adapted **EISP** method is tested.

For each time stamp t, a probability value for Q of being updated is varied. The versions of **PBR** and **EISP** that use the *0-1-optimization* are compared in Figure 13.6. A value of 0 corresponds to the case that Q is certain, whereas a value of 1 assumes an update of Q in each iteration and, thus, forces **EISP-01** to always recompute the current inverse ranking. It can be observed that the runtime required for processing the whole stream when using **EISP-01** increases linearly with a growing probability of the query object of being uncertain. This effect is due to the fact that the number of updates of Q and, thus, the number of complete recomputations have to be performed according to the chosen probability value. As **PBR-01** does not depend on the uncertainty of Q, because it recomputes the inverse ranking in each iteration anyway, its curve defines an upper asymptote to the curve of **EISP-01**.

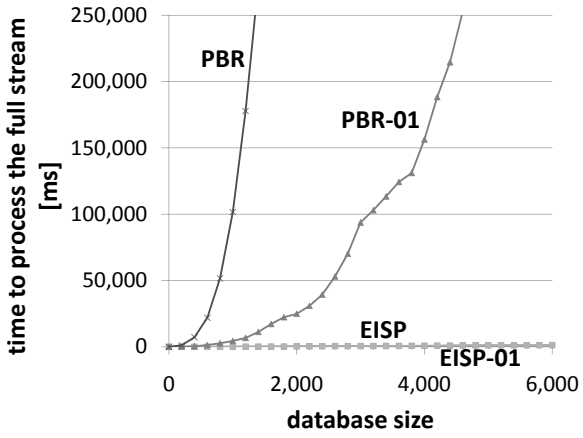


Figure 13.7: Scalability of the PIR approaches regarding full processing on the *IIP* dataset.

13.5.6 Scalability Evaluation on Real-World Data

IIP Dataset

The first experimental evaluation of the scalability on real-world data utilized the International Ice Patrol (*IIP*) Iceberg Sightings Dataset[2]. This dataset contains information about iceberg activity in the North Atlantic from 2001 to 2009. The latitude and longitude values of sighted icebergs serve as two-dimensional positions of up to 6,216 probabilistic objects, where each iceberg has been sighted at different positions. The stream consists of up to ten observations for each iceberg which are ordered chronologically. Here again, w is set to 3. Figure 13.7 indicates that the results obtained on real-world data are similar to those on synthetic data. For the *IIP* dataset, the *0-1-optimization* is very effective, since the position of an iceberg has a very small degree of uncertainty. Many icebergs even appear to hold their position over time.

NBA Dataset

The next set of experiments used the *NBA* dataset[3], containing information about North American basketball players. Each of the 3,738 records in this dataset corresponds to the performance of one player in one season. In particular, each record contains a total of 17 dimensions representing the number of games played, the number of points scored, and other statistics from one given season between the years 1946 and 2006. For the experiments, players are modeled by uncertain stream objects, using a sliding window model of size $w = 3$, that is, a player is described by his performance in the last three years. The probabilistic stream contains all records of the dataset. For simplicity, the used score function f_{score} is simply the sum of all (normalized) attributes. In this scenario, the

[2]The *IIP* dataset is available at the National Snow and Ice Data Center (NSIDC) web site (`http://nsidc.org/data/g00807.html`).

[3]The *NBA* dataset was derived from `http://www.databasebasketball.com`.

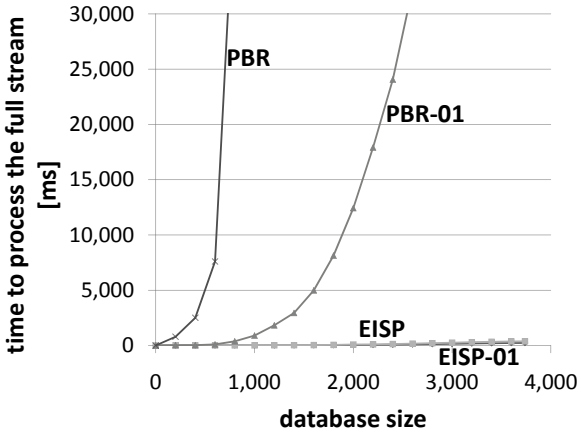


Figure 13.8: Scalability of the PIR approaches regarding full processing on the *NBA* dataset.

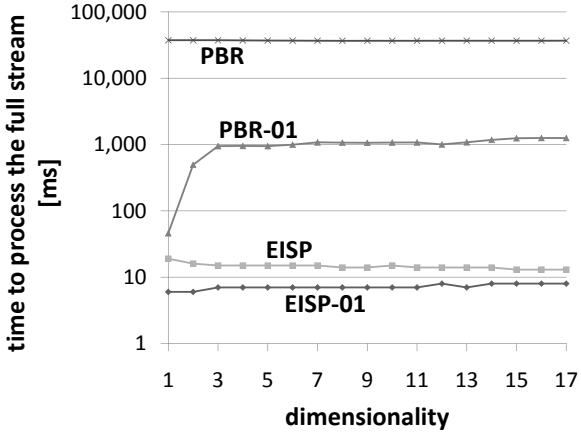


Figure 13.9: Scalability of the PIR approaches w.r.t. the data dimensionality regarding full processing on the *NBA* dataset.

semantics of a PIR query is to compute, for any given time, the rank of player Q w.r.t. all NBA players.

First, the scalability of the PIR algorithm was evaluated using all 17 dimensions. It can be observed from Figure 13.8 that the scalability is very similar to the *IIP* dataset, despite of the increased dimensionality. This is further evaluated in Figure 13.9, where the number of dimensions is scaled. For the approaches that do not utilize the *0-1-optimization*, the runtime appears to be constant in the number of dimensions. This can be explained by the fact that the dimensionality only affects the computation of the score of an object. The use of the sum of all dimensions leads to the theoretical expectation that the algorithm should scale linearly in the number of dimensions, but the impact of this linear computation can be neglected. It can also be observed that, for **PBR-01**, the runtime increases for low dimensionality and then becomes constant for higher dimensionality. This can be explained by the uncertainty of the individual dimensions: the first dimension represents the number of games played by a player, which is a variable with a rather low deviation for each player. Even if a player has a very volatile performance, the number of games he played may be about the same. Therefore, the one-dimensional dataset has a rather low uncertainty, and thus, a lower runtime (cf. Subsection 13.5.3). However, a player playing bad in the first games may be replaced, and, thus, not play the full time, which is covered by the second dimension that aggregates the number of minutes played in a year and has a higher deviation. The third dimension has the highest uncertainty, as it describes the number of points scored by a player in a year. After the third dimension, adding further dimensions does not significantly increase the total deviation of the sum (i.e., the score) of a player. In summary, increasing the dimensionality has no significant effect on the runtime, but may increase the uncertainty of the object, thus, indirectly increasing the runtime.

13.6 Summary

This chapter presented a general solution to efficiently answering continuous inverse ranking queries on uncertain streams, extending the incremental approach of updating rank probability distributions presented in Chapter 12. State-of-the-art approaches solving the probabilistic inverse ranking query problem for static data have not been applicable for stream data due to the originally quadratic complexity of the *Poisson Binomial Recurrence*. This chapter showed theoretically and experimentally that a linear update cost can be achieved and, thus, the approach is applicable for stream databases.

Chapter 14

Hot Item Detection in Uncertain Data

14.1 Introduction

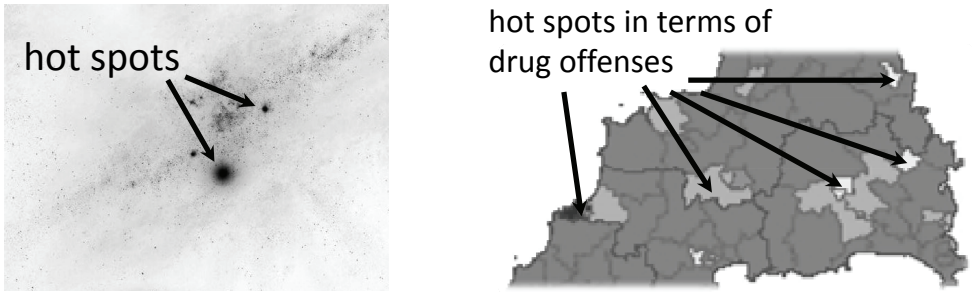
Beyond the relevance of similarity ranking in probabilistic databases, where efficient solutions were given in Chapters 11 and 12, also data mining tasks are faced with the presence of uncertainty. An important task is to rate the significance of uncertain objects. Chapter 13 tackled the problem of probabilistic inverse ranking, where the ranking position w.r.t. a given score function indicated the significance of a particular (uncertain) object among peers. This chapter will focus on a different semantics to rate the significance of objects. According to this semantics, an object is considered to be important if it shows characteristics that are similar to these of a sufficiently high population of other objects in the database.

The detection of objects which build dense regions with other objects within a vector space is a foundation of several density-based data mining techniques, in particular density-based clustering [90, 184], outlier detection and other density-based mining applications [61, 143, 194]. A (certain) object x for which exists a sufficiently large population of other objects in a database $\mathcal{D}$ that are similar to x is called a *hot item*. Intuitively, an item that shares its attributes with many other items could be potentially of interest, as its shows a typical occurrence of items in the database.

Application areas where the detection of hot items is potentially important exemplarily include scientific applications, e.g., astrophysics (cf. Figure 14.1(a)), biomedical, sociological and economic applications. In particular, the following applications give a good motivation for the efficient detection of hot items:

- Detection of "hot" research topics: Given a large database of actual research papers and articles, the task of this application is to identify those research articles addressing problems that might be relevant for a research community. A paper might be relevant if there exist enough other papers which address a similar problem.

- Online shopping advertising: Online shopping advertising often profits from software



(a) Astrological hot items in terms of interesting constellations.

(b) Hot item detection for crime defense applications.

Figure 14.1: Applications for hot item detection.

tools that extract items containing a high number of bids from online auction and shopping websites, e.g., the *Hot Item Finder*[1] for *eBay*[2]. One can imagine that a product which is quite similar to a lot of other products that already have a high number of bids is a potential candidate for also becoming a good selling product. The detection of such products could be very valuable for online shopping advertising.

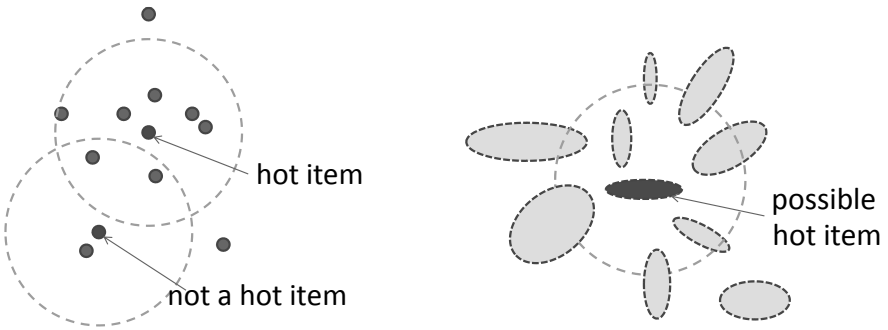
- Pre-detection of criminal activities: After a soccer game, one might be interested in the detection of larger groups of hooligans that should be accompanied by guards in order to avoid criminal excesses. If we assume that the locations of all hooligans are monitored, then it would be interesting which of these individuals have a lot of other hooligans in their immediate vicinity. Another example is the detection of outstanding crime, e.g., cases of drug abuse in areas with high population of drug offences as depicted in Figure 14.1(b)[3].

The applications mentioned above require special methods supporting the efficient search in modern databases that have to cope with uncertain or imprecise data. This chapter will propose the first approach addressing the retrieval of hot items in uncertain domains.

A hot item x has the property that the number of other items (objects) which are in the proximity of x, i.e., which are similar to x, exceed a given minimum population value. This chapter will give a general definition of hot items by relaxing the similarity predicate between the objects.

Definition 14.1 (Hot Item) *Given a database $\mathcal{D}$ with objects and a minimum population threshold minItems. Furthermore, given a score function $f_{score} : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}_0^+$, which is defined on pairs of objects in $\mathcal{D}$, and a similarity predicate $\phi_\varepsilon : \mathbb{R}_0^+ \rightarrow \{true, false\}$, where $\phi_\varepsilon \in \{<\varepsilon, \leq\varepsilon, =\varepsilon, \geq\varepsilon, >\varepsilon\}$ and $\varepsilon \in \mathbb{R}_0^+$ is a given scalar. An object $x \in \mathcal{D}$ is called* hot item*, iff there exist at least minItems objects $y \in \mathcal{D} \setminus \{x\}$ which satisfy the predicate ϕ_ε,*

[1] `http://www.hotitemfinder.com`
[2] `http://www.ebay.com`
[3] Source: `https://www.amethyst.gov.uk/crime_map/crimedrugs.htm`



(a) Hot items in certain data. (b) Hot items in uncertain data.

Figure 14.2: Examples of hot items.

formally

$$|\{y \in \mathcal{D} \setminus \{x\} : \phi_\varepsilon(f_{score}(x,y)) = true\}| \geq minItems \Leftrightarrow x \text{ is a } \textbf{\textit{hot item}}.$$

The value of the score function f_{score} reflects the degree of similarity of two objects w.r.t. the predicate ϕ_ε, where a small value indicates a high similarity, whereas a high value indicates high dissimilarity. In particular, two objects x and y are considered to be equal if $f_{score}(x,y) = 0$. For spatial data, this corresponds to the semantics of the distance between x and y.

In the case of uncertain objects, an exact score cannot be determined, in particular if the score relates to the object attributes which are assumed to be uncertain (cf. Figure 14.2). Consequently, uncertain objects lead to uncertain scores, which in turn lead to uncertain predicate results. Thus, the result of the predicate ϕ_ε is no longer binary and, instead, yields a probability value. This probabilistic predicate result can be estimated. Based on this estimation, it is possible to compute, for each probabilistic object X of an uncertain database, a probability value which reflects the likelihood that X is a hot item or not.

In the context of this chapter, hot items can be abstracted to objects that satisfy a given similarity predicate together with a reasonably large set of other items. If the *equality* predicate is assumed, i.e., $\phi_\varepsilon\left(f_{score}(x,y)\right) :=$ "$f_{score}(x,y) = 0$", then a hot item x satisfies the *frequent item* property, as x is equal to many other items and, thus, occurs frequently in the database.

The detection of hot items can be efficiently supported by a similarity join query used in a preprocessing step, in particular the distance-range self-join. Approaches for an efficient join on uncertain data are proposed in [138]. The main advantage of this approach is that discrete positions in space can efficiently be indexed using traditional spatial access methods, thus allowing to reduce the computational complexity of complex query types. The approach that will be proposed in this chapter exploits the similarity join approach proposed in [138]. However, the cost of the probabilistic detection of hot items is originally highly CPU-bound, which will be demonstrated in the experimental evaluation. The advantage of an I/O-cost-efficient approach for the preprocessing step only becomes no-

ticeable when applying the methods in a way that the CPU cost less outbalance the overall query cost.

The remainder of this chapter is organized as follows. Section 14.2 will formally introduce the problem of probabilistic identification of hot items in uncertain databases. The solution for the efficient computation of hot item probabilities can be found in Section 14.3. A performance evaluation of the proposed approach will be given in Section 14.4. Section 14.5 will conclude this chapter.

14.2 Problem Definition

14.2.1 Probabilistic Score

The identification whether an object is a hot item or not requires to know the neighborhood of the object according to a given (similarity) score function. Assuming that the object attributes that the score function relates to are uncertain, then the score result is uncertain, too. Therefore, a probabilistic score function is required which is defined as follows: let $P_{\phi_\varepsilon} : \mathcal{D} \times \mathcal{D} \rightarrow [0,1]$ be a probabilistic function defined on a pair of objects that returns the likelihood that a given score w.r.t. both objects satisfies a given predicate ϕ_ε. For example, consider two spatially uncertain objects X and Y according to Definition 9.2 in Chapter 9 that comply with the *x-relation model* [25]. The definition of the proximity of an object is assumed to be given by a spatial distance range, where ε denotes the distance parameter. Now, defining the score function f_{score} as the distance $dist(X,Y)$ between X and Y and using the predicate $\phi_\varepsilon\,(dist(X,Y)) :=$ "$dist(X,Y) \leq \varepsilon$", then $P_{\phi_\varepsilon}(X,Y)$ denotes the probability that Y is within the ε-range of X and vice versa.

14.2.2 Probabilistic Hot Items

Based on the definitions given above, it is possible to determine hot items in uncertain data in a probabilistic way. However, the problem of dependencies among the uncertain attributes has to be solved. Though the assumption is made that the attributes of uncertain objects are independent of each other, it is important to respect the mutual exclusiveness of the values of an uncertain object attribute. For this reason, there is the need of a definition of probabilistic hot items based on a conditional probability.

Definition 14.2 (Conditional Probabilistic Hot Item) *Given a database $\mathcal{D}$ with uncertain objects and a minimum population threshold minItems. Furthermore, a predicate $\phi_\varepsilon : \mathbb{R}_0^+ \rightarrow \{true, false\}$ is assumed, which is defined on a probabilistic score function, where $\phi_\varepsilon \in \{<\varepsilon, \leq\varepsilon, =\varepsilon, \geq\varepsilon, >\varepsilon\}$ and $\varepsilon \in \mathbb{R}_0^+$ is a given scalar. Assuming the uncertain object model of Definition 9.2 in Chapter 9, the probability that X is a hot item can be computed as follows:*

$$P\,(X \text{ is a hot item} \mid X = x) =$$

$$P\,(|\{Y \in \mathcal{D} \setminus \{X\} : \phi_\varepsilon(f_{score}(X,Y)) = true\}| \geq minItems) =$$

$$\sum_{\substack{S_{minItems} \subseteq \mathcal{D} \setminus \{X\} \\ |S_{minItems}| \geq minItems}} \left(\prod_{Y \in S_{minItems}} P_{\phi_\varepsilon}(X,Y) \cdot \prod_{Y \in \mathcal{D} \setminus (S_{minItems} \cup \{X\})} (1 - P_{\phi_\varepsilon}(X,Y)) \right),$$

where $S_{minItems}$ contains at least *minItems* objects $Y \in \mathcal{D} \setminus \{X\}$ which satisfy the query predicate ϕ_ε.

The above definition gives rise to the following general definition of probabilistic hot items which depends on the used uncertainty model. The probability $P(X \text{ is a hot item})$ of an object X being an (unconditionally) probabilistic hot item can be computed by aggregating the conditional hot item probabilities over all possible observations x of X multiplied with the probability that object X is represented by x, i.e.,

$$\sum_{x \in X} P(X = x) \cdot P\left(|\{Y \in \mathcal{D} \setminus \{X\} : \phi_\varepsilon(f_{score}(x,Y)) = true\}| \geq minItems\right).$$

14.3 Hot Item Detection Algorithm

14.3.1 Initialization

Let $\mathcal{D}$ be a database with uncertain objects. Each object $X \in \mathcal{D}$ is examined w.r.t. the hot item property. This computation can be split into the preprocessing step, which finds candidates that match the predicate ϕ_ε, and the query step, which detects the hot items.

14.3.2 Preprocessing Step

First, for each object $Y \in \mathcal{D} \setminus \{X\}$, it is required to compute the probability that Y satisfies a given predicate $\phi_\varepsilon \in \{< \varepsilon, \leq \varepsilon, = \varepsilon, \geq \varepsilon, > \varepsilon\}$ w.r.t. object X, i.e., it is necessary to compute $P_{\phi_\varepsilon}(X,Y)$. Obviously, only those objects $Y \in \mathcal{D}' \subseteq \mathcal{D}$, for which the predicate ϕ_ε is satisfied with a probability greater than 0, i.e., $P_{\phi_\varepsilon}(X,Y) > 0$, have to be taken into account in order to compute the probability $P(X \text{ is a hot item})$. Depending on the used predicate ϕ_ε, ϕ_ε is usually selective, i.e., only a small number of N' objects $\in \mathcal{D}' \subseteq \mathcal{D}$ satisfy the predicate $\phi_\varepsilon(X,Y)$ with a probability greater than 0. A quick search of those objects which have to be taken into account can be efficiently supported by means of an index structure, e.g., the R^*-tree [23]. In particular for the predicate $\phi_\varepsilon :=$ " $\leq \varepsilon$", the index-supported ε-range join [63] can be used to enhance the search as proposed in [53]. Here, approximate representations like the minimum bounding rectangle (MBR) of an uncertain object are very appropriate to be used as index key for a filter step following the multi-step query processing paradigm. A solution for the ε-range join on uncertain data is proposed in [138], which can be used as a preprocessing step for the proposed algorithm for the detection of hot items.

14.3.3 Query Step

In the following, an approach will be introduced which efficiently computes the probability that an object $X \in \mathcal{D}$ is a hot item. The proposed algorithm has quadratic runtime or even needs linear time if *minItems* is assumed to be constant. The key idea of the proposed approach is based on the following property. Given a set of j predicates $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_j\}$ for which the probability $P(\varphi_i)$ that the predicate $\varphi_i \in \Phi$ is true is known, respectively. Now, the probability $P_{p,\Phi}$ that at least p predicates of Φ are true has to be computed.

Lemma 14.1 *If it is assumed that the predicate φ_i is true, then $P_{p,\Phi}$ is equal to the probability that at least $p-1$ predicates of $\Phi \setminus \{\varphi_i\}$ are true. Otherwise, $P_{p,\Phi}$ is equal to the probability that at least p predicates of $\Phi \setminus \{\varphi_i\}$ are true.*

The above lemma leads to the following recursion that allows to compute $P_{p,\Phi}$ by extending the *Poisson Binomial Recurrence* [147], as already done in the contexts of probabilistic ranking in Chapters 11 and 12 and of probabilistic inverse ranking in Chapter 13:

$$P_{p,\Phi} = P_{p-1,\Phi\setminus\{\varphi_i\}} \cdot P(\varphi_i) + P_{p,\Phi\setminus\{\varphi_i\}} \cdot (1 - P(\varphi_i)), \tag{14.1}$$

where

$$P_{0,\emptyset} = 1 \text{ and } P_{p,\Phi} = 0 \text{ if } p < 0 \vee p > |\Phi|.$$

Here, this technique is generalized for arbitrary probabilistic predicates. The solution presented in this chapter extends this method to compute the probability that an uncertain object $X \in \mathcal{D}$ is a hot item. Given an uncertain object $X \in \mathcal{D}$, the value for *minItems* and the set $\mathcal{D}' \subseteq \mathcal{D}$ of objects for which the probability that the predicate $P_{\phi_\varepsilon}(f_{score}(X,Y))$ $(Y \in \mathcal{D}')$ is true is greater than 0, i.e.,

$$\forall Y \in \mathcal{D}' : P_{\phi_\varepsilon}(f_{score}(X,Y)) > 0.$$

The probability $P(X \text{ is a hot item})$ is equal to the probability $P_{minItems,\mathcal{D}',X}$ that, for at least *minItems* objects $Y \in \mathcal{D}'$, the predicates $\phi_\varepsilon(f_{score}(X,Y))$ are true. With Lemma 14.1 and the dynamic-programming technique described in Equation (14.1), it is possible to compute the probability $P_{minItems,\mathcal{D}',X}$ efficiently by

$$P_{minItems,\mathcal{D}',X} = \begin{cases} P_{minItems-1,\mathcal{D}'\setminus\{Y\},X} \cdot P_{\phi_\varepsilon}(f_{score}(X,Y)) + \\ P_{minItems,\mathcal{D}'\setminus\{Y\},X} \cdot (1 - P_{\phi_\varepsilon}(f_{score}(X,Y))) & \text{if } minItems > 0 \\ 1 & \text{if } minItems = 0. \end{cases}$$

14.4 Experimental Evaluation

14.4.1 Datasets and Experimental Setup

This section will present the results of an experimental evaluation of the proposed methods w.r.t. efficiency on artificial and real-world datasets. In the artificial *ART* dataset, each

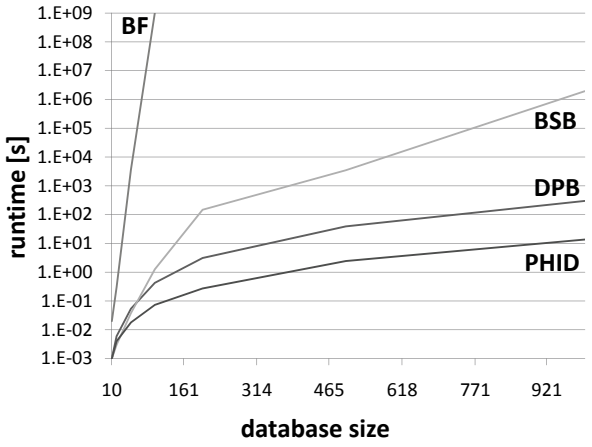
object is represented by a set of positions sampled from an individual five-dimensional hyperrectangle R with a given size. The observations are uniformly distributed within the rectangles. The rectangles are arbitrarily distributed within the object space. Each of the 1,500 objects of the two semi-real-world datasets *SCI1* and *SCI2* consists of a set of $m = 10$ observations taken from environmental time series, where each observation is described by several attributes that correspond to different environmental sensor measurements of one single day[4]. The attribute set of *SCI1* describes temperature, humidity and CO concentration, whereas *SCI2* has a larger set of attributes (temperature, humidity, speed and direction of wind as well as concentrations of CO, SO_2, NO, NO_2 and O_3). Similarly to Chapter 12, the probability for each observation x_i of an uncertain object X was set to $\frac{1}{10}$, summing up in an overall probability of 1.

The following experiments compare two variants of the approach proposed in this chapter, denoted by **DPB** and **PHID**. In contrast to **PHID**, **DPB** applies dynamic programming on the complete database, i.e., $\mathcal{D}' = \mathcal{D}$ and, thus, does not require the preprocessing step. The performance of **PHID** and **DPB** is compared to that of the brute-force solution (**BF**) that simply applies the formulas given in Subsection 14.2.2. Furthermore, they are compared to the bisection-based method (**BSB**) which was adapted to the method proposed in Chapter 11. This method is able to significantly reduce the computational cost than the brute-force method, but is still exponential. The proposed algorithm concentrates on the evaluation of the CPU cost only. The reason is that the **PHID** approach is clearly CPU-bound. The only I/O bottleneck is the initial computation of the likelihood that X is in the ε-range of y, for each object $X \in \mathcal{D}$ and each observation $y \in Y$, where $Y \in \mathcal{D} \setminus \{X\}$. This requires a distance-range self-join of the database, which can be performed by a nested-block-loop join that requires $O(N^2)$ page faults in the worst case. In contrast, the CPU time for the **PHID** approach is cubic: each call of the dynamic-programming algorithm requires $O(N'^2)$ time and has to be performed once for each observation in $\mathcal{D}'$. This yields computational cost of $O(N'^3 \cdot m)$, where m is the number of observations per object, but as $m \ll N'$ can be assumed and, therefore, m can be regarded to be constant, an overall computational cost of $O(N'^3)$ is required.

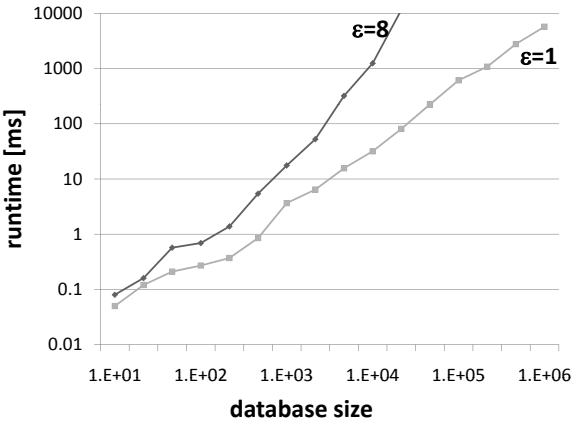
14.4.2 Scalability Experiments

The first experiments relate to the scalability of the proposed approaches. The results depicted in Figure 14.3 demonstrate how the runtime of the competing techniques is influenced by the database size. Figure 14.3(a) shows that, though the bisection-based approach has exponential runtime, it outperforms the brute-force approach by several orders of magnitude. However, the dynamic-programming-based approaches **DPB** and **PHID** are significantly more efficient than their competitors **BF** and **BSB**, since the latter have exponential runtime. Furthermore, the preprocessing step of **PHID** obviously pays off. The performance can be further improved by an order of magnitude when applying the

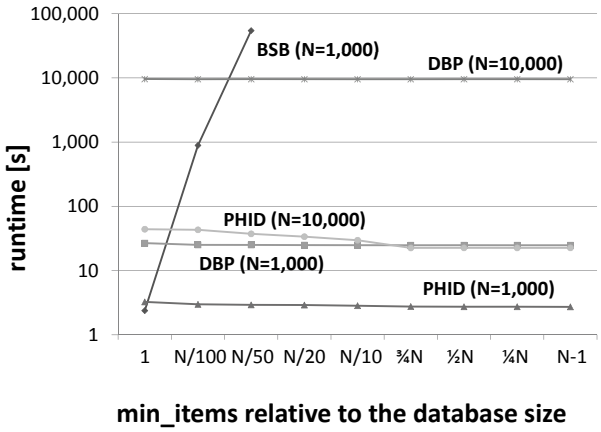
[4]The environmental time series have been provided by the Bavarian State Office for Environmental Protection, Augsburg, Germany (`http://www.lfu.bayern.de/`).



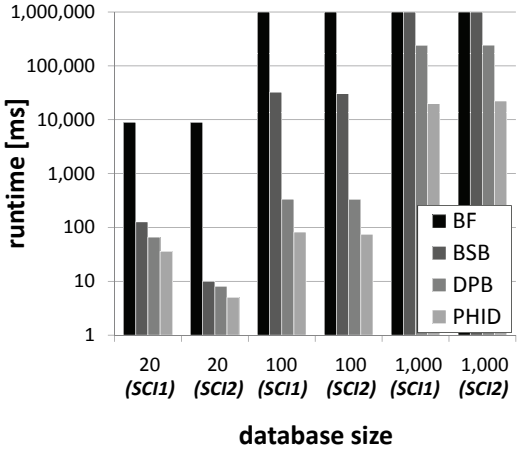
(a) Evaluation of competing techniques.



(b) Scalability experiments with **PHID** and different values for ε.



(c) Influence of *minItems*.



(d) Experiments on real-world data.

Figure 14.3: Performance experiments.

dynamic-programming technique only on N' objects Y where the probability of the predicate $P_{\phi_\varepsilon}(f_{score}(X,Y))$ is not 0, such that the query processing step reduces from $O(N^3)$ to $O(N'^3)$ with $N' \ll N$. The next experiment shows the scalability of **PHID** for different values of ε. For a higher value of ε implying a lower selectivity, there are significantly more candidates w.r.t. the predicate ϕ_ε, resulting in higher computational requirements. Here, the average time required to compute the hot item probability for an object was measured. The results shown in Figure 14.3(b) demonstrate that **PHID** scales well, even for very large databases. Figure 14.3(c) demonstrates the performance w.r.t. the *minItems* value for different database sizes. Contrary to **DPB** and **PHID**, the **BSB** method is very affected by the *minItems* value due to the expensive probability computation. The slight increase of the **DPB** and **PHID** performances can be explained by the reduced number of hot items with increasing *minItems* value. Finally, the performance is evaluated based

on the real-world datasets *SCI1* and *SCI2* (cf. Figure 14.3(d)). Unlike the exponential algorithms, **DPB** and **PHID** perform a full hot item scan of the database in reasonable time, even for a relatively large database size.

14.5 Summary

This chapter proposed an efficient approach for the detection of probabilistic *hot items*, i.e., uncertain objects X for which there exists a sufficiently high population of other objects which are similar to X. In particular, the proposed approach computes, for each object X in an uncertain database, the probability that X is a hot item. Therefore, methods were proposed that break down the high computational complexity required to compute this probability. Theoretical and experimental proofs showed that the proposed approach can efficiently solve the problem (in a cubic worst-case time complexity in the number of objects that satisfy the query predicate), while the competing techniques have exponential runtime.

Chapter 15

Probabilistic Frequent Itemset Mining in Uncertain Databases

15.1 Introduction

15.1.1 Uncertainty in the Context of Frequent Itemset Mining

Beyond the detection of hot items, association rule analysis is one of the most important fields in data mining. It is commonly applied to market-basket databases for the analysis of consumer purchasing behavior. Such databases consist of a set of transactions, each containing the items a customer purchased. The most important and computationally intensive step in the mining process is the extraction of *frequent itemsets* – sets of items that occur in at least *minSup* transactions. It is generally assumed that the items occurring in a transaction are known for certain. However, this is not always the case – as already outlined in Part I –, due to several reasons:

- In many applications, the data is inherently noisy, such as data collected by sensors or in satellite images.
- In privacy protection applications, artificial noise can be added deliberately [210]. Finding patterns despite this noise is a challenging problem.
- By aggregating transactions by customer, it is possible to mine patterns across customers instead of transactions. This produces estimated purchase probabilities per item per customer rather than certain items per transaction.

In such applications, the information captured in transactions is uncertain, since the existence of an item is associated with a likelihood measure or existential probability. Given an uncertain transaction database, it is not obvious how to identify whether an item or itemset is frequent because it cannot be generally said for certain whether an itemset appears in a transaction. In a traditional (certain) transaction database, the solution is to simply perform a database scan and count the transactions that include the itemset. This does not work in an uncertain transaction database.

Customer	Item	Prob.
A	Game	1.0
A	Music	0.2
B	Video	0.4
B	Music	0.7

ID	Transaction
t_A	(Game, 1.0); (Music, 0.2)
t_B	(Video, 0.4); (Music, 0.7)

Table 15.1: Example application of an uncertain transaction database.

World	TransactionDB	Prob.
W_1	{Game}; {}	0.144
W_2	{Game, Music}; {}	0.036
W_3	{Game}; {Video}	0.096
W_4	{Game, Music}; {Video}	0.024
W_5	{Game}; {Music}	0.336
W_6	{Game, Music}; {Music}	0.084
W_7	{Game}; {Video, Music}	0.224
W_8	{Game, Music}; {Video, Music}	0.056

Table 15.2: Corresponding possible worlds.

Dealing with such databases is a difficult, but interesting problem. While a naïve approach might transform uncertain items into certain ones by thresholding the probabilities, this loses useful information and leads to inaccuracies. Existing approaches in the literature are based on expected support, first introduced in [80]. Chui et. al. [79, 80] take the uncertainty of items into account by computing the expected support of itemsets. There, itemsets are considered to be frequent if the expected support exceeds *minSup*. Effectively, this approach returns an estimate of whether an object is frequent or not with no indication of how good this estimate is. Since uncertain transaction databases yield uncertainty w.r.t. the support of an itemset, the probability distribution of the support and, thus, information about the confidence of the support of an itemset is very important. This information, while present in the database, is lost using the expected support approach.

Example 15.1 *Consider a department store. To maximize sales, customers can be analyzed to find sets of items that are all purchased by a large group of customers. This information can be used for advertising directed to this group. For example, by providing special offers that include all of these items along with new products, the store can encourage new purchases. Table 15.1 shows such information. Here, Customer A purchases games every time he visits the store and music (CDs) 20% of the time. Customer B buys music in 70% of her visits and videos (DVDs) in 40% of them. The store uses a database that represents each customer as a single* uncertain transaction*, also shown in Table 15.1.*

The following subsection will introduce the uncertain data model that will be assumed in this chapter. Then, the problem definition will be given. First, an overview of the frequently used notations in this chapter is valuable. These are listed in Table 15.3.

Notation	Description
$\mathcal{W}$	the set of all possible worlds
W	a possible world instance $W \in \mathcal{W}$
$\mathcal{T}$	an uncertain transaction database
N	the cardinality of $\mathcal{T}$
t	a transaction $t \in \mathcal{T}$
$\mathcal{T}_j$	a subset of $\mathcal{T}$, limited to j transactions
$\mathcal{I}$	the set of all items
x	an item $x \in \mathcal{I}$
X	an itemset $X \subseteq \mathcal{I}$
$P(x \in t)$	the probability that item x occurs in transaction t
$\mathcal{S}(X, W)$	the support of X in world W
$P_i(X)$	the probability that the support of X is i
$P_{\geq i}(X)$	the probability that the support of X is *at least* i
$P_{i,j}(X)$	the probability that i of the first j transactions contain X
$P_{\geq i,j}(X)$	the probability that *at least* i of the first j transactions contain X

Table 15.3: Table of notations used in this chapter.

15.1.2 Uncertain Data Model

The uncertain data model used in this chapter is based on the *Possible Worlds Semantics* [145] (cf. Definition 9.1 in Chapter 9) with existential *uncertain items*. Uncertain items and uncertain transactions can be defined as follows.

Definition 15.1 (Uncertain Item) *An* uncertain item *is an item $x \in \mathcal{I}$ whose presence in a transaction $t \in \mathcal{T}$ is defined by an* existential probability $P(x \in t) \in (0, 1)$. *A* certain item *is an item where $P(x \in t) \in \{0, 1\}$. $\mathcal{I}$ is the set of all possible items.*

Definition 15.2 (Uncertain Transaction Database) *An* uncertain transaction *t is a transaction that contains uncertain items. A transaction database $\mathcal{T}$ containing $|\mathcal{T}| = N$ uncertain transactions is called an* uncertain transaction database.

An uncertain transaction t is represented in an uncertain transaction database by the items $x \in \mathcal{I}$ associated with an existential probability value $P(x \in t) \in (0, 1]$[1]. An example of an uncertain transaction database is listed in Tables 15.1 and 15.2, which illustrate the store example of Example 15.1. To interpret an uncertain transaction database, the possible worlds semantics is applied. An uncertain transaction database generates *possible worlds*, where each world is defined by a fixed set of (certain) transactions. A possible world is instantiated by generating each transaction $t \in \mathcal{T}$ according to the occurrence probabilities $P(x \in t)$. Consequently, each probability $0 < P(x \in t) < 1$ derives two possible worlds *per transaction*: one possible world in which x exists in t, and one possible world where x does

[1] If an item x has an existential probability of 0, it does not appear in the transaction.

not exist in t. Thus, the number of possible worlds of a database increases exponentially in both the number of transactions and the number of uncertain items contained in it.

Each possible world W is associated with a probability that this world exists, $P(W)$. Table 15.2 shows all possible worlds derived from Table 15.1. Returning to Example 15.1, in world W_6, both customers bought music, Customer B decided against a new video and Customer A bought a new game.

It is assumed that uncertain transactions are mutually independent. Thus, in the current scenario, the decision by Customer A has no influence on Customer B. This assumption is reasonable in real-world applications. Additionally, independence between items both within the same transaction as well as in different transactions is often assumed in the literature [6, 79, 80]. This can be justified by the assumption that the items are observed independently. In this case, the probability of a world W is given by

$$P(W) = \prod_{t \in \mathcal{T}} \left(\prod_{x \in t} P(x \in t) \cdot \prod_{x \notin t} (1 - P(x \in t)) \right).$$

This assumption does not imply that the underlying instantiations of an uncertain transaction database will result in uncorrelated items, since the set of items having nonzero probability in a transaction may be correlated. In Example 15.1, the probability of world W_5 in Table 15.2 is $P(Game \in t_A) \cdot (1 - P(Music \in t_A)) \cdot P(Music \in t_B) \cdot (1 - P(Video \in t_B)) = 1.0 \cdot 0.8 \cdot 0.7 \cdot 0.6 = 0.336$.

In the general case, the occurrence of items may be dependent. For example, the decision to purchase a new music video DVD may mean that a customer is unlikely to purchase a music CD by the same artist. Alternatively, some items must be bought together. If these conditional probabilities are known, they can be used in the proposed methods. For example, the probability that both a video and music are purchased by customer B is $P(\{Video, Music\} \in t_B) = P(Video \in t_B) \cdot P(Music \in t_B \mid Video \in t_B)$.

15.1.3 Problem Definition

An itemset is a *frequent itemset* if it occurs in at least *minSup* transactions, where *minSup* is a user-specified parameter. In uncertain transaction databases, however, the support of an itemset is uncertain; it is defined by a discrete *support probability distribution function* (*SPDF*). Therefore, each itemset has a *frequentness probability*[2] – the probability that it is frequent. This chapter focuses on the problem of efficiently computing this SPDF (which will be defined in Section 15.2.2) and extracting all *probabilistic frequent itemsets*.

Definition 15.3 (Probabilistic Frequent Itemset (PFI)) *A* Probabilistic Frequent Itemset (PFI) *is an itemset with a* frequentness probability *of at least* τ.

The parameter τ is the user-specified minimum confidence in the frequentness of an itemset.

[2]Frequentness is the rarely used word describing the property of being frequent.

It is now possible to specify the *Probabilistic Frequent Itemset Mining* (*PFIM*) *problem* as follows. Given an uncertain transaction database $\mathcal{T}$, a minimum support scalar *minSup* and a frequentness probability threshold τ, the objective is to find all probabilistic frequent itemsets.

15.1.4 Contributions and Outline

This chapter makes the following contributions:

- A probabilistic framework will be proposed for frequent itemset mining in databases containing uncertain transactions, based on the possible worlds model (cf. Definition 9.1 in Chapter 9).
- A dynamic computation method will be presented for computing the probability that an itemset is frequent, as well as the entire SPDF of the support of an itemset, in $O(N)$ time, assuming that *minSup* is a constant. Without this technique, it would run in exponential time in the number of transactions. Using the approach that will be proposed in this chapter, the algorithm has the same time complexity as methods based on the expected support [79, 80, 150]. However, the proposed approach yields much more effectiveness, since it provides confidences for frequent itemsets.
- An algorithm will be proposed to mine all itemsets that are frequent with a probability of at least τ. Furthermore, an additional algorithm will be proposed that incrementally outputs the uncertain itemsets in the order of their frequentness probability. This ensures that itemsets with the highest probability of being frequent are output first. This has two additional advantages. First, it makes the approach free of the parameter τ. Secondly, it solves the top-k itemsets problem in uncertain databases.

The remainder of this chapter is organized as follows. Section 15.2 will present the proposed probabilistic support framework. Section 15.3 will show how to compute the frequentness probability in $O(N)$ time. Section 15.4 will present a probabilistic frequent itemset mining algorithm. Section 15.5 will present the proposed incremental algorithm. The experiments will be presented in Section 15.6. Finally, Section 15.7 will conclude this chapter.

15.2 Probabilistic Frequent Itemsets

15.2.1 Expected Support

Previous work addressing the problem of frequent itemset mining in uncertain databases was based on the expected support [79, 80, 150], which is defined as follows.

Definition 15.4 (Expected Support) *Given an uncertain transaction database $\mathcal{T}$, the* expected support $E(X)$ *of an itemset X is defined as* $E(X) = \sum_{t \in \mathcal{T}} P(X \subseteq t)$.

ID	Transaction
t_1	(A, 0.8); (B, 0.2); (D, 0.5); (F, 1.0)
t_2	(B, 0.1); (C, 0.7); (D, 1.0); (E, 1.0), (G, 0.1)
t_3	(A, 0.5); (D, 0.2); (F, 0.5); (G, 1.0)
t_4	(D, 0.8); (E, 0.2); (G, 0.9)
t_5	(C, 1.0); (D, 0.5); (F, 0.8); (G, 1.0)
t_6	(A, 1.0); (B, 0.2); (C, 0.1)

Table 15.4: Example of a larger uncertain transaction database.

Considering an itemset frequent if its expected support is above *minSup* has a major drawback. Uncertain transaction databases naturally involve uncertainty concerning the support of an itemset. Considering this is important when evaluating whether an itemset is frequent or not. However, this information is forfeited when using the expected support approach. In the example shown in Table 15.4, the expected support of the itemset $\{D\}$ is $E(\{D\}) = 3.0$. The fact that $\{D\}$ occurs for certain in one transaction, namely in t_2, and that there is at least one possible world where $\{D\}$ occurs in five transactions are totally ignored when using the expected support in order to evaluate the frequency of an itemset. Indeed, suppose $minSup = 3$; is it appropriate to call $\{D\}$ frequent? And if so, how certain can we even be that $\{D\}$ is frequent? By comparison, consider itemset $\{G\}$. This also has an expected support of 3.0, but its presence or absence in the transactions is more certain. It turns out that the probability that $\{D\}$ is frequent is 0.7 (cf. Subsection 15.2.3), and the probability that $\{G\}$ is frequent is 0.91. While both have the same expected support, we can be quite confident that $\{G\}$ is frequent, in contrast to $\{D\}$. An expected-support-based technique does not differentiate between the two.

The confidence with which an itemset is frequent is very important for interpreting uncertain itemsets. Therefore, concepts are required that allow to evaluate the uncertain data in a probabilistic way. This section formally introduces the concept of probabilistic frequent itemsets.

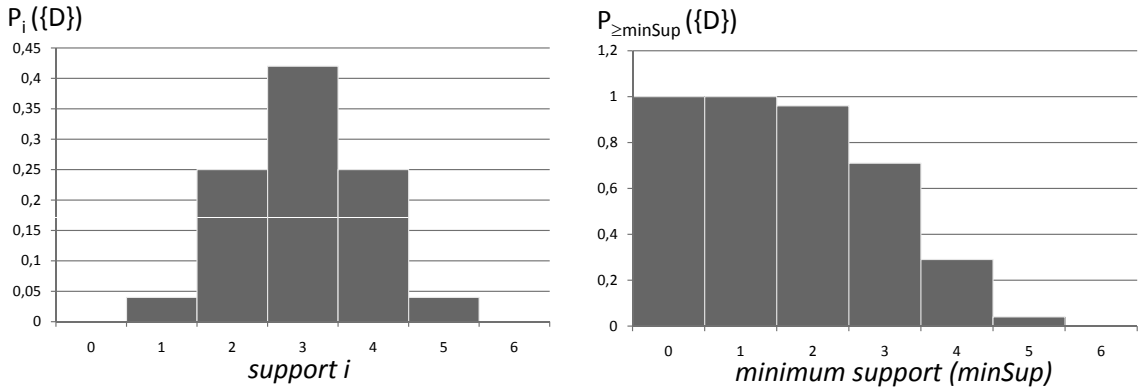
15.2.2 Probabilistic Support

In uncertain transaction databases, the support of an item or itemset cannot be represented by a unique value, but must rather be represented by a discrete SPDF.

Definition 15.5 (Support Probability) *Given an uncertain transaction database $\mathcal{T}$ and the set $\mathcal{W}$ of possible worlds (instantiations) of $\mathcal{T}$, the* support probability $P_i(X)$ *of an itemset X is the probability that X has the support i. Formally,*

$$P_i(X) = \sum_{W \in \mathcal{W}} P(W) \cdot I_{\mathcal{S}(X,W)=i},$$

where $\mathcal{S}(X, W)$ is the support of itemset X in world W and I_z is an indicator variable that is 1 if $z = true$ and 0 otherwise.



(a) SPDF of $\{D\}$. (b) Frequentness probabilities of $\{D\}$.

Figure 15.1: Probabilistic support of itemset $\{D\}$ in the uncertain database of Table 15.4.

Intuitively, $P_i(X)$ denotes the probability that the support of X is exactly i . The support probabilities associated with an itemset X for different support values form the SPDF of the support of X.

Definition 15.6 (Support Probability Distribution Function (SPDF)) *The* probabilistic support *of an itemset X in an uncertain transaction database $\mathcal{T}$ is defined by the support probabilities of X ($P_i(X)$) for all possible support values $i \in \{0, \ldots, N\}$. This probability distribution is called* Support Probability Distribution Fuction *(*SPDF*). The following statement holds: $\sum_{0 \leq i \leq N} P_i(X) = 1.0$.*

Returning to the example of Table 15.4, Figure 15.1(a) shows the SPDF of itemset $\{D\}$. The number of possible worlds $|\mathcal{W}|$ that need to be considered for the computation of $P_i(X)$ is extremely large. In fact, there are $O(2^{N \cdot |\mathcal{I}|})$ possible worlds, where $|\mathcal{I}|$ denotes the total number of items. The following Lemma shows how to compute $P_i(X)$ without materializing all possible worlds.

Lemma 15.1 *For an uncertain transaction database $\mathcal{T}$ with mutually independent transactions and any $0 \leq i \leq N$, the support probability $P_i(X)$ can be computed by*

$$P_i(X) = \sum_{\mathcal{T}' \subseteq T, |\mathcal{T}'| = i} \left(\prod_{t \in \mathcal{T}'} P(X \subseteq t) \cdot \prod_{t \in \mathcal{T} - \mathcal{T}'} (1 - P(X \subseteq t)) \right), \tag{15.1}$$

where the transaction subset $\mathcal{T}' \subseteq \mathcal{T}$ contains exactly i transactions.

Proof. *The transaction subset $\mathcal{T}' \subseteq \mathcal{T}$ contains i transactions. The probability of a world W where all transactions in $\mathcal{T}'$ contain X and the remaining $|\mathcal{T} - \mathcal{T}'|$ transactions do not contain X is $P(W) = \prod_{t \in \mathcal{T}'} P(X \subseteq t) \cdot \prod_{t \in \mathcal{T} - \mathcal{T}'} (1 - P(X \subseteq t))$. The sum of the probabilities according to all possible worlds satisfying the above conditions corresponds to the equation given in Definition 15.5.* □

15.2.3 Frequentness Probability

The definition of the probabilistic support now allows to tackle the actual problem definition to compute the probability that an itemset is frequent, i.e., the probability that an itemset occurs in at least *minSup* transactions.

Definition 15.7 (Frequentness Probability) *Let $\mathcal{T}$ be an uncertain transaction database and X be an itemset. $P_{\geq i}(X)$ denotes the probability that the support of X is* at least *i, i.e., $P_{\geq i}(X) = \sum_{k=i}^{N} P_k(X)$. For a given minimum support $minSup \in \{0, \ldots, N\}$, the probability $P_{\geq minSup}(X)$, which is called the* frequentness probability *of X, denotes the probability that the support of X is at least $minSup$.*

Figure 15.1(b) shows the frequentness probabilities of $\{D\}$ for all possible *minSup* values in the database of Table 15.4. For example, the probability that $\{D\}$ is frequent when $minSup = 3$ is approximately 0.7, while its frequentness probability when $minSup = 4$ is approximately 0.3.

The intuition behind $P_{\geq minSup}(X)$ is to have a confidence to rate an itemset as frequent. With this policy, the frequentness of an itemset becomes subjective and the decision about which candidates shall be reported to the user depends on the application. Hence, the minimum frequentness probability τ is used as a user-defined parameter. Some applications may need a low τ, while in other applications only highly confident results shall be reported (high τ).

In the possible worlds model, it is known that $P_{\geq i}(X) = \sum_{W \in \mathcal{W}, \mathcal{S}(X,W) \geq i} P(W)$. This can be computed according to Equation (15.1) by

$$P_{\geq i}(X) = \sum_{\mathcal{T}' \subseteq \mathcal{T}, |\mathcal{T}'| \geq i} \left(\prod_{t \in \mathcal{T}'} P(X \subseteq t) \cdot \prod_{t \in \mathcal{T} - \mathcal{T}'} (1 - P(X \subseteq t)) \right). \tag{15.2}$$

Hence, the frequentness probability can be computed by enumerating all possible worlds satisfying the *minSup* condition through the direct application of Equation (15.2). However, this naïve approach is very inefficient. It is possible to speed this up significantly. Typically $minSup \ll N$ and the number of worlds with support i is at most $\binom{N}{i}$. Hence, enumeration of all worlds W in which the support of X is greater than *minSup* is much more expensive than enumerating those where the support is less than *minSup*. Using the following easily verified Corollary, the frequentness probability can be computed exponentially in $minSup \ll N$.

Corollary 15.1

$$P_{\geq i}(X) = 1 - \sum_{\mathcal{T}' \subseteq \mathcal{T}, |\mathcal{T}'| < i} \left(\prod_{t \in \mathcal{T}'} P(X \subseteq t) \cdot \prod_{t \in \mathcal{T} - \mathcal{T}'} (1 - P(X \subseteq t)) \right)$$

Despite this improvement, the complexity of the above approach, called **Basic** in the experiments, is still exponential w.r.t. the number of transactions. The bisection-based approach that was proposed in Chapter 11 could achieve a reduction of this complexity, but this was shown to have still exponential runtime requirements. Therefore, Section 15.3 will describe how this can be reduced to linear time.

15.3 Efficient Computation of Probabilistic Frequent Itemsets

15.3.1 Efficient Computation of Probabilistic Support

This section will present a dynamic-programming approach which avoids the enumeration of possible worlds in computing the frequentness probability and the SPDF. Later, in Subsection 15.3.2, probabilistic filter and pruning strategies will be presented, which further improve the runtime of the method. The key to the approach is to consider it in terms of subproblems. First, appropriate definitions are needed:

Definition 15.8 (Dynamic Probability Computation) *The probability that i of j transactions contain itemset X is*

$$P_{i,j}(X) = \sum_{\mathcal{T}' \subseteq T_j, |\mathcal{T}'|=i} \left(\prod_{t \in \mathcal{T}'} P(X \subseteq t) \cdot \prod_{t \in \mathcal{T}_j - \mathcal{T}'} (1 - P(X \subseteq t)) \right),$$

where $\mathcal{T}_j = \{t_1, \ldots, t_j\} \subseteq \mathcal{T}$ is the set of the first j transactions. Similarly, the probability that at least *i of j transactions contain itemset X is*

$$P_{\geq i,j}(X) = \sum_{\mathcal{T}' \subseteq T_j, |\mathcal{T}'| \geq i} \left(\prod_{t \in \mathcal{T}'} P(X \subseteq t) \cdot \prod_{t \in \mathcal{T}_j - \mathcal{T}'} (1 - P(X \subseteq t)) \right).$$

It holds that $P_{\geq i,N}(X) = P_{\geq i}(X)$, which denotes the probability that at least i transactions in the entire database contain X. The key idea now is to split the problem of computing $P_{\geq i,N}(X)$ into smaller problems $P_{\geq i,j}(X)$, $j < N$. This can be achieved as follows. Given a set of j transactions $\mathcal{T}_j = \{t_1, \ldots, t_j\} \subseteq \mathcal{T}$; if transaction t_j is assumed to contain itemset X, then $P_{\geq i,j}(X)$ is equal to the probability that at least $i-1$ transactions of $\mathcal{T}_j \setminus \{t_j\}$ contain X. Otherwise, $P_{\geq i,j}(X)$ is equal to the probability that at least i transactions of $\mathcal{T}_j \setminus \{t_j\}$ contain X. By splitting the problem in this way, the recursion in Lemma 15.2, which provides information about these probabilities, can be used to compute $P_{\geq i,j}(X)$ by means of the paradigm of dynamic programming. This scheme, known as *Poisson Binomial Recurrence* [147], has already been used in the context of probabilistic ranking in Chapters 11 and 12, for solving the problem of probabilistic inverse ranking in Chapter 13 and for the context of detecting hot items in Chapter 14. Here, the dynamic-programming technique is extended for the efficient computation of frequent itemsets in a probabilistic way.

Lemma 15.2

$$P_{\geq i,j}(X) = P_{\geq i-1,j-1}(X) \cdot P(X \subseteq t_j) + P_{\geq i,j-1}(X) \cdot (1 - P_j(X \subseteq t_j)),$$

where

$$P_{\geq 0,j} = 1 \ \forall j \in \{0, \ldots, N\} \text{ and } P_{\geq i,j} = 0 \quad \text{if } i < 0 \vee i > j.$$

Proof.

$$P_{\geq i,j}(X) = \sum_{k=i}^{j} P_{k,j}(X) \overset{[214]}{=} \sum_{k=i}^{j} P_{k-1,j-1}(X) \cdot P(X \subseteq t_j) + \sum_{k=i}^{j} P_{k,j-1}(X) \cdot (1 - P(X \subseteq t_j))$$

$$\overset{[P_{\geq i,j}=0 \ \forall i>j]}{=} P(X \subseteq t_j) \cdot \sum_{k=i}^{j} P_{k-1,j-1}(X) + (1 - P(X \subseteq t_j)) \cdot \sum_{k=i}^{j-1} P_{k,j-1}(X)$$

$$= P(X \subseteq t_j) \cdot \sum_{k=i-1}^{j-1} P_{k,j-1}(X) + (1 - P(X \subseteq t_j)) \cdot \sum_{k=i}^{j-1} P_{k,j-1}(X)$$

$$= P(X \subseteq t_j) \cdot P_{\geq i-1,j-1}(X) + (1 - P(X \subseteq t_j)) \cdot P_{\geq i,j-1}(X).$$

□

Using this dynamic-programming scheme, the probability that at least *minSup* transactions contain itemset X can be obtained by computing the cells depicted in Figure 15.2. In the matrix, each cell relates to a probability $P_{\geq i,j}$, with j (the number of transactions) marked on the x-axis, and i (the support) marked on the y-axis. According to Lemma 15.2, in order to compute a $P_{\geq i,j}$, the probabilities $P_{\geq i-1,j-1}$ and $P_{\geq i,j-1}$ are required, that is, the cell to the left and the cell to the lower left of $P_{\geq i,j}$. Knowing that $P_{\geq 0,0} = 1$ and $P_{\geq 1,0} = 0$ by definition, the first task is to compute $P_{\geq 1,1}$. The probability $P_{\geq 1,j}$ can then be computed by using the previously computed $P_{\geq 1,j-1}$ for all j. $P_{\geq 1,j}$ can, in turn, be used to compute $P_{\geq 2,j}$. This iteration continues until i reaches *minSup*, so that finally $P_{\geq minSup,N}$ – the frequentness probability (cf. Definition 15.7) – is obtained.

In each line (i.e., for each i) of the matrix in Figure 15.2, j only runs up to $N - minSup + i$. Larger values of j are not required for the computation of $P_{\geq minSup,N}$.

Lemma 15.3 *The computation of the frequentness probability $P_{\geq minSup}$ requires at most $O(N \cdot \ minSup) \overset{[minSup \ll N]}{=} O(N)$ time and at most $O(N)$ space.*

Proof. *Using the dynamic computation scheme as shown in Figure 15.2, the number of computations is bounded by the size of the depicted matrix. The matrix contains $N \cdot minSup$ cells. Each cell requires an iteration of the dynamic computation (cf. Lemma 15.2) which is performed in $O(1)$ time. Here, a matrix is used for illustration purpose only. The computation of each probability $P_{\geq i,j}(X)$ only requires information stored in the current line and the previous line to access the probabilities $P_{\geq i-1,j-1}(X)$ and $P_{\geq i,j-1}(X)$. Therefore,*

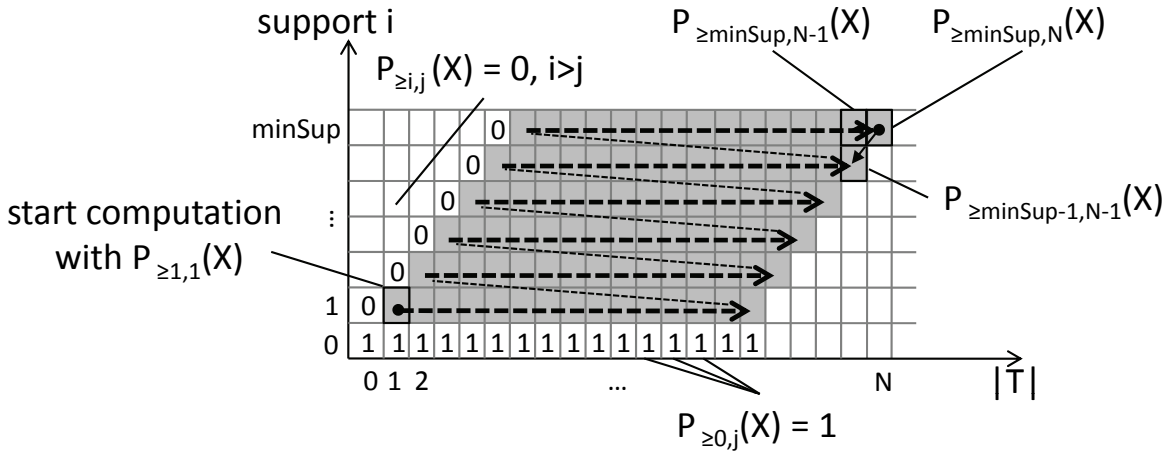


Figure 15.2: Dynamic computation scheme.

only these two lines (of length N) need to be preserved requiring $O(N)$ space. Additionally, the probabilities $P(X \subseteq t_j)$ have to be stored, resulting in a total of $O(N)$ space. □

If an itemset is certain in some transactions, computation time can be saved. If a transaction $t_j \in \mathcal{T}$ contains itemset X with a probability of 0, i.e., $P(X \subseteq t_j) = 0$, transaction t_j can be ignored for the dynamic computation because $P_{\geq i,j}(X) = P_{\geq i,j-1}(X)$ holds (cf. Lemma 15.2). If $|\mathcal{T}'|$ is less than *minSup*, then X can be pruned since, by definition, $P_{\geq minSup,\mathcal{T}'} = 0$ if $minSup > \mathcal{T}'$. The dynamic computation scheme can also omit transactions t_j where the item has a probability of 1, because $P_{\geq i,j}(X) = P_{\geq i-1,j-1}(X)$ due to $P(X \subseteq t_j) = 1$. Thus, if a transaction t_j contains X with a probability of 1, then t_j (i.e., the corresponding column) can be omitted if *minSup* is reduced by one, to compensate the missing transaction. The dynamic-programming scheme therefore only has to consider uncertain items. This trick is called *0-1-optimization* in the following.

15.3.2 Probabilistic Filter Strategies

Monotonicity Criteria

To further reduce the computational cost, this section will introduce probabilistic filter strategies. These reduce the number of probability computations in the dynamic algorithm. The probabilistic filter strategies exploit the following monotonicity criteria.

Lemma 15.4 *If the minimum support i is increased, then the frequentness probability of an itemset decreases, i.e.,*

$$P_{\geq i,j}(X) \geq P_{\geq i+1,j}(X).$$

Proof. $P_{\geq i+1,j}(X) \overset{\text{Definition 15.7}}{=} P_{\geq i,j}(X) - P_{i,j}(X) \leq P_{\geq i,j}(X)$ □

Intuitively, this result is obvious since the predicate "the support is at least i" implies "the support is at least $i + 1$".

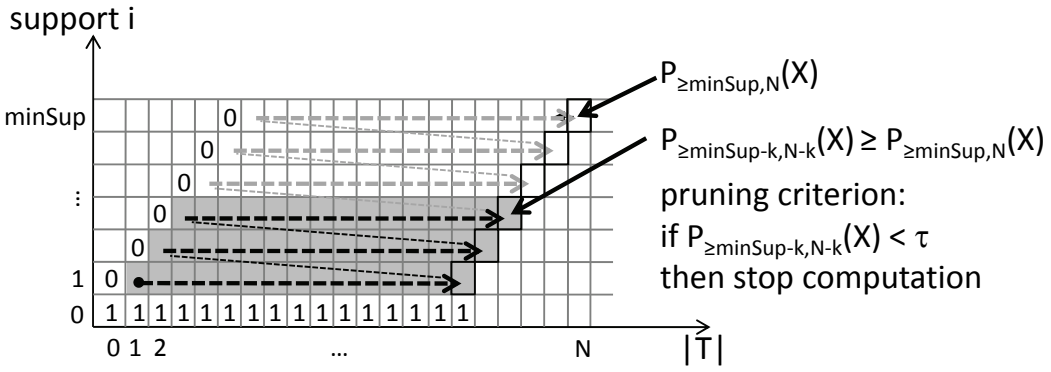


Figure 15.3: Visualization of the pruning criterion.

Lemma 15.5 *The next criterion says that an extension of the uncertain transaction database leads to an increase of the frequentness probability of an itemset, i.e.,*

$$P_{\geq i,j}(X) \leq P_{\geq i,j+1}(X).$$

Proof. $P_{\geq i,j+1}(X) \overset{\text{Lemma 15.2}}{=} P_{\geq i-1,j}(X) \cdot P(X \subseteq t_{j+1}) + P_{\geq i,j}(X) \cdot (1 - P(X \subseteq t_{j+1}))$
$\overset{\text{Lemma 15.4}}{\geq} P_{\geq i,j}(X) \cdot P(X \subseteq t_{j+1}) + P_{\geq i,j}(X) \cdot (1 - P(X \subseteq t_{j+1})) = P_{\geq i,j}(X)$ □

The intuition behind this lemma is that one more transaction can increase the support of an itemset. Putting these results together yields the following lemma.

Lemma 15.6

$$P_{\geq i,j}(X) \geq P_{\geq i+1,j+1}(X)$$

Proof. $P_{\geq i+1,j+1}(X) \overset{\text{Lemma 15.2}}{=} P_{\geq i,j}(X) \cdot P(X \subseteq t_{j+1}) + P_{\geq i+1,j}(X)(1 - P(X \subseteq t_{j+1}))$
$\overset{\text{Lemma15.4}}{\leq} P_{\geq i,j}(X) \cdot P(X \subseteq t_{j+1}) + P_{\geq i,j}(X)(1 - P(X \subseteq t_{j+1})) = P_{\geq i,j}(X)$ □

Now, the following part describes how these monotonicity criteria can be exploited to prune the dynamic computation.

Pruning Criterion

Lemma 15.6 can be used to quickly identify non-frequent itemsets. Figure 15.3 shows the dynamic-programming scheme for an itemset X to compute $P_{\geq minSup,N}(X)$. Lemma 15.6 states that the probabilities $P_{\geq minSup-k,N-k}(X), k \in \{1, \ldots, minSup\}$ (highlighted in Figure 15.3), are conservative upper bounds of $P_{\geq minSup,N}(X)$. Thus, if any of the probabilities $P_{\geq minSup-k,N-k}(X), k \in \{1, \ldots, minSup\}$ is lower than the user-specified parameter τ, then X can be pruned.

15.4 Probabilistic Frequent Itemset Mining (PFIM)

Now, the techniques required to efficiently identify whether a given itemset X is an uncertain frequent itemset are given. This section will show how to find all uncertain frequent itemsets in an uncertain transaction database. Traditional frequent itemset mining is based on support pruning by exploiting the anti-monotonic property of support: $\mathcal{S}(X) \leq \mathcal{S}(Y)$ where $\mathcal{S}(X)$ is the support of X and $Y \subseteq X$. However, as stated in Subsection 15.2.2, support in uncertain transaction databases is defined by an SPDF and itemsets are mined according to their frequentness probability. It turns out that the frequentness probability is anti-monotonic.

Lemma 15.7 $\forall Y \subseteq X : P_{\geq minSup}(X) \leq P_{\geq minSup}(Y)$. *In other words, all subsets of an uncertain frequent itemset are also uncertain frequent itemsets.*

Proof.

$$P_{\geq i}(X) = \sum_{W \in \mathcal{W}} P(W) \cdot I_{\mathcal{S}(X,W) \geq i},$$

since the probability is defined over all possible worlds. Here, I_z is an indicator variable that is 1 if $z = true$ and 0 otherwise. In other words, $P_{\geq i}(X)$ is the relative number of worlds in which $\mathcal{S}(X) \geq i$ holds, where each occurrence is weighted by the probability of the world occurring. Since world W corresponds to a normal transaction database with no uncertainty, $\mathcal{S}(X,W) \leq \mathcal{S}(Y,W)$ $\forall Y \subseteq X$ due to the anti-monotonicity of support. Therefore,

$$I_{\mathcal{S}(X,W) \geq i} \leq I_{\mathcal{S}(Y,W) \geq i} \; \forall W \in \mathcal{W}, \forall Y \subseteq X$$

and, thus,

$$P_{\geq i}(X) \leq P_{\geq i}(Y), \forall Y \subseteq X.$$

□

The contrapositive of Lemma 15.7 can be used to prune the search space for uncertain frequent itemsets. That is, if an itemset Y is not an uncertain frequent itemset, i.e., $P_{\geq minSup}(Y) < \tau$, then all itemsets $X \supseteq Y$ cannot be uncertain frequent itemsets either.

The first proposed algorithm is based on a "marriage" of traditional frequent itemset mining methods and the proposed uncertain itemset identification algorithms presented in Section 15.3. In particular, a probabilistic frequent itemset mining approach will be proposed that is based on the Apriori algorithm [10]. Like Apriori, the proposed method iteratively generates the uncertain frequent itemsets using a bottom-up strategy. Each iteration is performed in two steps: a join step for generating new candidates, and a pruning step for computing the frequentness probabilities and extracting the uncertain frequent itemsets from the candidates. The pruned candidates are, in turn, used to generate candidates in the next iteration. Lemma 15.7 will be exploited in the join step to limit the number of generated candidates and in the pruning step to remove itemsets that need not be expanded.

Algorithm 10 Incremental Algorithm

```
1: result ← []
2: AIQ ← new PriorityQueue() {initialize}
3: for all z ∈ I do
4:    AIQ.add([z, P_≥minSup(z)])
5: end for
6: while further results required do
7:    X ← AIQ.getNext()
8:    result.add(X) {add the next uncertain frequent itemset to result}
9:    for all z ∈ I \ {X} do
10:      AIQ.add([X ∪ {z},P_≥minSup(X ∪ {z})])
11:   end for
12: end while
13: return result
```

A drawback of Apriori is that it returns uncertain frequent itemsets in ascending order of their size. Therefore, the following section will propose an incremental algorithm that utilizes a priority queue w.r.t. the frequentness probability.

15.5 Incremental PFIM (I-PFIM)

15.5.1 Query Formulation

The probabilistic frequent itemset mining approach that will be presented in this section allows the user to control the confidence of the results using τ. However, since the number of results depends on τ, it may prove difficult for a user to correctly specify this parameter without additional domain knowledge. Therefore, this section shows how to efficiently solve the following problems, which do not require the specification of τ:

- *Top-k uncertain frequent itemsets query:* return the k itemsets that have the highest frequentness probability, where k is specified by the user.
- *Incremental ranking query:* successively return the itemsets with the highest frequentness probability, one at a time.

15.5.2 The PFIM Algorithm

In the incremental algorithm (cf. Algorithm 10), an *Active Itemsets Queue* (*AIQ*) is kept that is initialized with all one-item sets (line 4). The *AIQ* is sorted by frequentness probability in descending order. Without loss of generality, itemsets are represented in lexicographical order to avoid generating them more than once. In each iteration of the algorithm, i.e., with each call of the function getNext() on the *AIQ*, the first itemset X

in the queue is removed (line 7). X is the next most uncertain frequent itemset, because all other itemsets in the AIQ have a lower frequentness probability due to the order on AIQ, and all supersets of X (which have not yet been generated) cannot have a higher frequentness probability due to Lemma 15.7. After X is added to the result set, it is refined in a candidate generation step (line 9). This step creates all supersets of X obtained by adding single items z to the end of X, in such a way that the lexicographical order of $X \cup \{z\}$ is maintained. These are then added to the AIQ after their respective frequentness probabilities are computed (cf. Section 15.3). The user can continue calling getNext() until he or she has all required results. During each call of getNext(), the size of the AIQ increases by at most $|\mathcal{I}|$. The maximum size of the AIQ is $2^{|\mathcal{I}|}$, which is no worse than the space required to sort the output of a non-incremental algorithm.

15.5.3 Top-k Probabilistic Frequent Itemsets Query

However, in many applications, relatively few top uncertain frequent itemsets are required. For instance, the store in Example 15.1 may want to know the *top-100*. Top-k highest frequentness probability queries can be efficiently computed by using Algorithm 10 and constraining the length of the AIQ to $k - h$, where h is the number of highest frequentness probability items already returned. Any itemsets that "fall off" the end can safely be ignored. The rationale behind this approach is that, for an itemset X at position p in the AIQ, $p - 1$ itemsets with a higher frequentness than X exist in the AIQ by construction. Additionally, any of the h itemsets that have already been returned must have a higher frequentness probability. Consequently, the top-k algorithm constrains the size of the initial AIQ to k and reduces its size by one each time a result is reported. The algorithm terminates once the size of the AIQ reaches 0.

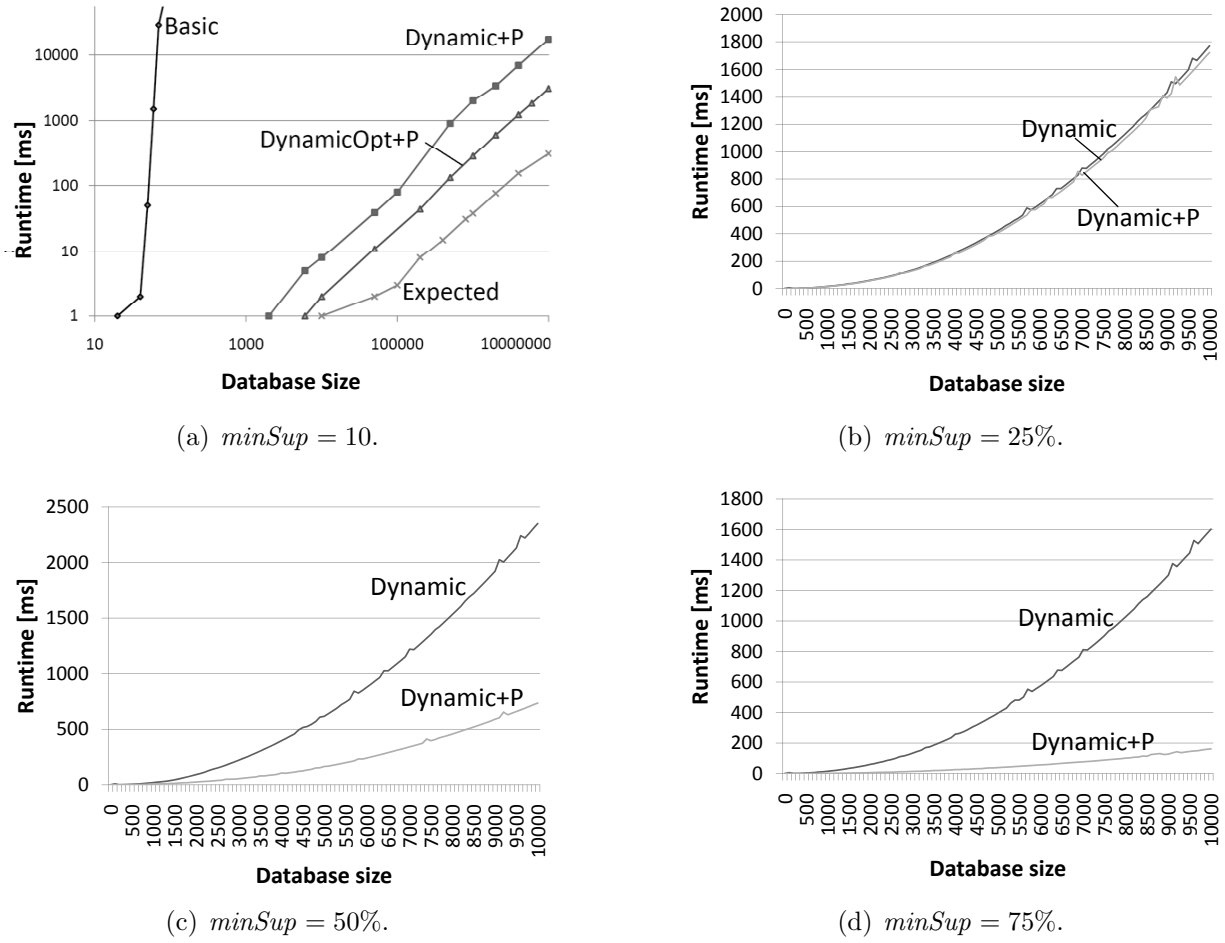
15.6 Experimental Evaluation

15.6.1 Overview

This section will present an evaluation of the proposed algorithm. Subsection 15.6.2 will give efficiency results obtained using the different methods of computing the probabilistic support. Then, Subsection 15.6.3 will discuss the performance and utility of the proposed PFIM algorithms. In all experiments, the runtime was measured in milliseconds (ms).

15.6.2 Evaluation of the Frequentness Probability Computations

The proposed frequentness probability computation methods were evaluated on several artificial datasets with varying database sizes N and densities. The *density* of an item denotes the expected portion of transactions in which an item may be present (i.e., where its existence probability is in $(0, 1]$). The probabilities themselves were drawn from a uniform distribution. The density is directly related to the *degree of uncertainty* of an item. If not



(a) $minSup = 10$.

(b) $minSup = 25\%$.

(c) $minSup = 50\%$.

(d) $minSup = 75\%$.

Figure 15.4: Runtime evaluation w.r.t. N.

stated otherwise, the settings used a database consisting of 10,000 to 10,000,000 uncertain transactions with 20 items and a density of 0.5. The frequentness probability threshold τ was set to 0.9. The following notations are used for the proposed frequentness probability computation algorithms:

- **Basic**: basic probability computation (cf. Subsection 15.2.3)
- **Dynamic**: dynamic probability computation (cf. Subsection 15.3.1)
- **Dynamic+P**: dynamic probability computation with pruning (cf. Subsection 15.3.2)
- **DynamicOpt**: dynamic probability computation utilizing 0-1-optimization (cf. Subsection 15.3.1)
- **DynamicOpt+P**: 0-1-optimized dynamic probability computation method with pruning

Scalability

Figure 15.4 shows the scalability of the probability computation approaches when the number of transactions N is varied. The runtime of the **Basic** approach increases exponentially in *minSup* as explained in Subsection 15.2.3, and is therefore not applicable for a $N > 50$ as can be seen in Figure 15.4(a). The approaches **Dynamic+P** and **DynamicOpt+P** scale linearly as expected when using a constant *minSup* value. The U-Apriori approach which considers the expected support [80] was also evaluated in the first experiment, denoted as **Expected**. This approach achieves a constant performance gain compared to **Dynamic+P** and **DynamicOpt+P**, but shows the same runtime complexity. Nevertheless, the trade-off between computational cost and result quality is not satisfying (cf. Subsection 15.2.1).

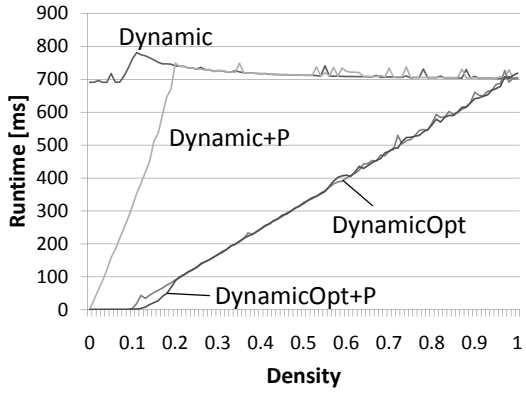
The *0-1-optimization* has an impact on the runtime whenever there is some certainty in the database. The performance gain of the proposed pruning strategies depends on the used *minSup* value. In Figures 15.4(b) to 15.4(d), the scalability of **Dynamic** and **Dynamic+P** is shown for different *minSup* values expressed as percentages of N. It is notable that the time complexity of $O(N \cdot \ minSup)$ becomes $O(N^2)$ if *minSup* is chosen relative to the database size N. Also, it can be observed that the higher *minSup*, the higher the difference between **Dynamic** and **Dynamic+P**; a higher *minSup* causes the frequentness probability to fall overall, thus, allowing earlier pruning.

Effect of the Density

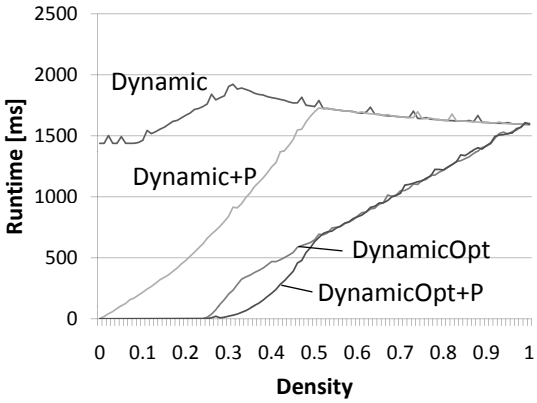
Next, the effectiveness of the proposed pruning strategy is evaluated w.r.t. the density. *minSup* is important here too, so results are reported for different values in Figure 15.5. The *0-1-optimization* works well as long as the density of the data is below *minSup*, as in this case, no item is considered to be frequent. With increasing density, the items are present in more transactions. Thus, the effectiveness of the *0-1-optimization* decreases. The pruning works well for datasets with low density and has no effect on the runtime for higher densities. The reason is straightforward; the higher the density, the higher the probability that a given itemset is frequent and, thus, cannot be pruned. Regarding the effect of *minSup*; a larger *minSup* value decreases the probability that itemsets are frequent and therefore increases the number of computations that can be pruned. The break-even point between pruning and non-pruning in the experiments is when the density is approximately twice the *minSup* value, since, due to the method of creating the datasets, this corresponds to the expected support. At this value, all itemsets are expected to be frequent and cannot be pruned. Overall, with reasonable parameter settings, the proposed pruning strategies achieve a significant speed-up for the identification of uncertain frequent itemsets.

Effect of *minSup*

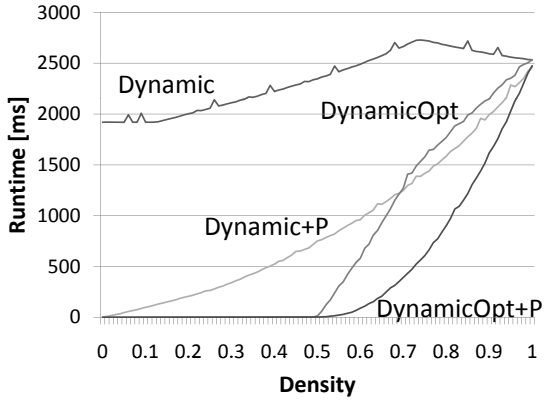
Figure 15.6 shows the influence of *minSup* on the runtime when using different densities, assuming $N = 10,000$. The runtime of **Dynamic** directly correlates with the size of the



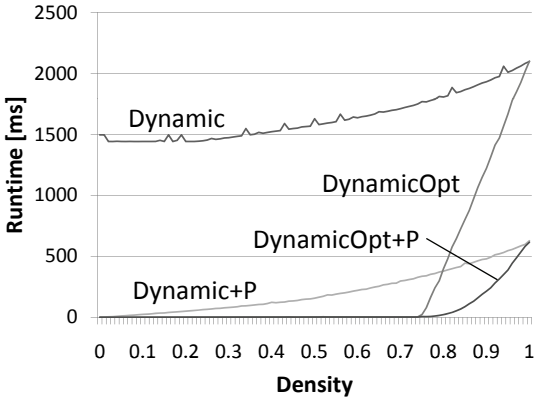
(a) $minSup = 0.1$.



(b) $minSup = 0.25$.

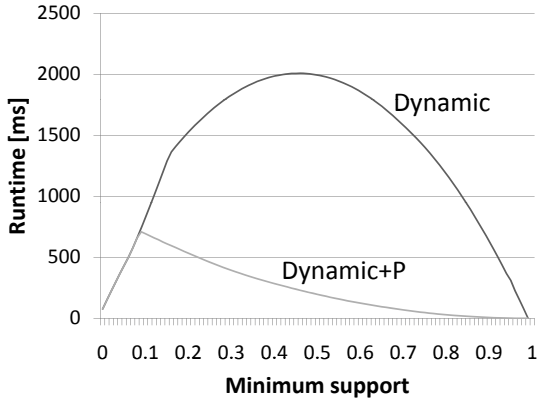


(c) $minSup = 0.5$.

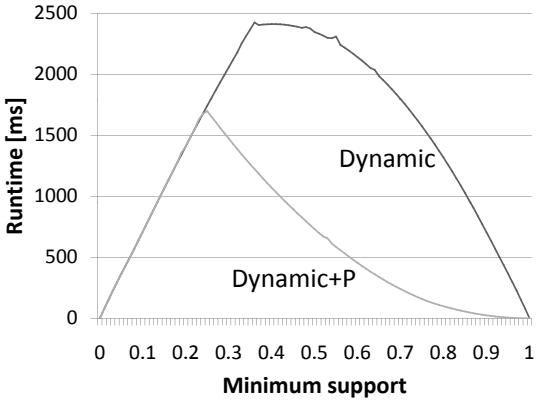


(d) $minSup = 0.75$.

Figure 15.5: Runtime evaluation w.r.t. the density.

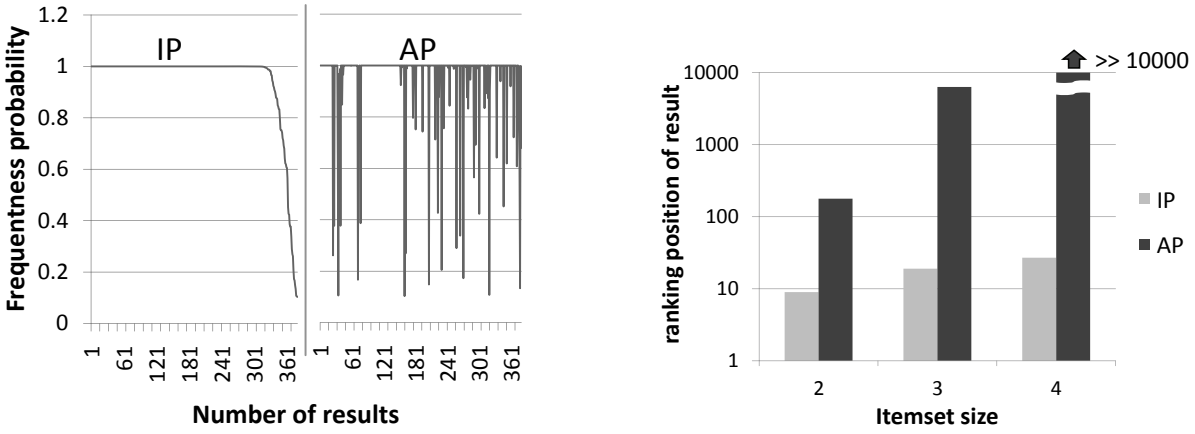


(a) Density = 0.2.



(b) Density = 0.5.

Figure 15.6: Runtime evaluation w.r.t. *minSup*.



(a) Output: **AP** vs. **IP**.

(b) Effectiveness of ranking queries.

Figure 15.7: Effectiveness of **AP** vs. **IP**.

dynamic computation matrix (cf. Figure 15.2). A low *minSup* value leads to few matrix rows which need to be computed, while a high *minSup* value leads to a slim row width. The total number of matrix cells to be computed is $minSup \cdot (N - minSup + 1)$, with a maximum at $minSup = \frac{N+1}{2}$. As long as the *minSup* value is below the expected support value, the approach with pruning shows similar characteristics; in this case, almost all item(set)s are expected to be frequent. However, the speed-up due to the pruning rapidly increases for *minSup* above this break-even point.

15.6.3 Evaluation of the PFIM Algorithms

Experiments for the probabilistic frequent itemset mining algorithms were run on a subset of the real-world dataset *accidents* [3], denoted by ACC. It consists of 340,184 transactions and 572 items whose occurrences in transactions were randomized; with a probability of 0.5, each item appearing for certain in a transaction was assigned a value drawn from a uniform distribution in $(0, 1]$. Here, **AP** is used to denote the Apriori-based and **IP** for the incremental algorithm (cf. Sections 15.4 and 15.5).

Top-k queries were performed on the first 10,000 transactions of ACC using a $minSup = 500$ and $\tau = 0.1$. Figure 15.7(a) shows the result of **IP**. The frequentness probability of the resulting itemsets is monotonically decreasing. In contrast, **AP** returns uncertain frequent itemsets in the classic way; in ascending order of their size, i.e., all itemsets of size 1 are returned first, etc. While both approaches return probabilistic frequent itemsets, **AP** returns an arbitrary frequentness probability order, while **IP** returns the most relevant itemsets first.

Next, ranking queries were performed on the first 100,000 itemsets (Figure 15.7(b)). In this experiment, the aim was to find the h-itemset X with the highest frequency probability

[3]The *accidents* dataset [98] was derived from the Frequent Itemset Mining Dataset Repository (`http://fimi.cs.helsinki.fi/data/`).

of all h-itemsets, where $h \in \{2, 3, 4\}$. Measuring the number of itemsets returned before X, it can be observed that the speed-up factor for ranking (and, thus, for top-k queries) is several orders of magnitude and increases exponentially in the length of requested itemset length. The reason is that **AP** must return all frequent itemsets of length $h-1$ before processing itemsets of length h, while **IP** quickly ranks itemsets in order of their frequentness probability, therefore leading to higher-quality results delivered to the user much earlier.

15.7 Summary

This chapter transferred the concepts of efficiently solving the problem of similarity ranking in probabilistic databases to the problem of probabilistic frequent itemset mining, where the basic task is to find itemsets in an uncertain transaction database that are (highly) likely to be frequent. It could be theoretically and experimentally shown that the proposed dynamic computation technique computes the exact support probability distribution of an itemset in linear time w.r.t. the number of transactions instead of the exponential runtime of a non-dynamic computation. Furthermore, it was demonstrated that the proposed probabilistic pruning strategy allows to prune non-frequent itemsets early, leading to a large performance gain. In addition, an iterative itemset mining framework was introduced which reports the most likely frequent itemsets first.

Chapter 16

Probabilistic Frequent Pattern Growth for Itemset Mining in Uncertain Databases

16.1 Introduction

16.1.1 Apriori and FP-Growth

It was shown in Chapter 15 that the use of the expected support [79, 80] has significant drawbacks, yielding misleading results. The proposed alternative was based on computing the entire probability distribution of itemsets' support. This was achieved in the same runtime as the expected support approach by employing the *Poisson Binomial Recurrence* technique (*PBR*) [147]. Chapter 15 adopted an Apriori-like approach, which is based on an *anti-monotonic* Apriori property [10] (i.e., if an itemset X is not frequent, then any itemset $Y \supseteq X$ is not frequent) and candidate generation. However, it is well-known that Apriori-like algorithms suffer a number of disadvantages. First, all candidates generated must fit into the main memory and the number of candidates can become prohibitively large. Secondly, checking whether a candidate is a subset of a transaction is not trivial. Finally, the entire database needs to be scanned multiple times. In uncertain databases, the effective transaction width is typically larger than in a certain transaction database which in turn can increase the number of candidates generated and the resulting space and time costs.

In certain transaction databases, the FP-Growth Algorithm [104] has become the established alternative. By building an FP-tree – effectively a compressed and highly indexed structure storing the information in the database –, candidate generation and multiple database scans can be avoided. However, extending this idea to mining probabilistic frequent patterns in uncertain transaction databases is not trivial. Previous extensions of FP-Growth to uncertain databases used the expected support approach [6, 117]. This is much easier, since these approaches ignore the probability distribution of support.

This chapter will introduce the *Probabilistic Frequent Pattern Tree*, which compresses

ID	Transaction
t_1	$(A, 1.0); (B, 0.2); (C, 0.5)$
t_2	$(A, 0.1); (D, 1.0)$
t_3	$(A, 1.0); (B, 1.0); (C, 1.0); (D, 0.4)$
t_4	$(A, 1.0); (B, 1.0); (D, 0.5)$
t_5	$(B, 0.1); (C, 1.0)$
t_6	$(C, 0.1); (D, 0.5)$
t_7	$(A, 1.0); (B, 1.0); (C, 1.0)$
t_8	$(A, 0.5); (B, 1.0)$

Table 16.1: Uncertain transaction database.

probabilistic databases and allows the efficient extraction of the existental probabilities required to compute the support probability distribution (SPDF) and the frequentness probability. Additionally, the *ProFPGrowth* algorithm will be proposed for mining all probabilistic frequent itemsets without candidate generation.

The basic terms that will be used in this chapter include the following: an *uncertain item* $x \in \mathcal{I}$, where $\mathcal{I}$ denotes the set of all possible items, is defined according to Definition 15.1 in Chapter 15. According to Definition 15.2, an *uncertain transaction* t contains uncertain items; an *uncertain transaction database* $\mathcal{T}$ contains $|\mathcal{T}| = N$ uncertain transactions. The table of notations given in Chapter 15 is also valid in the context of this chapter.

An example of a small uncertain transaction database is given in Table 16.1; this exemplary database will serve as a running example in this chapter. For each transaction t_i $(1 \leq i \leq 8)$, each item x is listed with its probability of existing in t_i. Items with an existential probability of 0 can be neglected. For simplicity, the consideration of other customers is omitted in this example.

16.1.2 Contributions and Outline

The problem definition, similar to Chapter 15, is the following. Given an uncertain transaction database $\mathcal{T}$, a minimum support scalar *minSup* and a frequentness probability threshold τ, the objective is to find all probabilistic frequent itemsets. Addressing this problem, this chapter makes the following contributions:

- The *Probabilistic Frequent Pattern Tree* (*ProFP-tree*) will be introduced, which is the first FP-tree type approach for handling uncertain or probabilistic data. This tree efficiently stores a probabilistic database and enables an efficient extraction of itemset occurrence probabilities and database projections.

- The *ProFPGrowth* algorithm will be proposed, which is based on the ProFPTree and which mines all itemsets that are frequent with a probability of at least τ without using expensive candidate generation.

- Finally, an intuitive and efficient method based on *Generating Functions* [154] will be introduced for computing the probability that an itemset is frequent and the entire SPDF of an itemset in $O(N)$ time, assuming *minSup* is a constant. Using the proposed approach, the algorithm that computes these probabilities has the same time complexity as the approach based on the PBR, which was presented in Chapter 15, but it is much more intuitive and, thus, offers various advantages, as will be shown.

The rest of this chapter is organized as follows. Section 16.2 will present the ProFP-tree; furthermore, it will be explained how the ProFP-tree is constructed. Additionally, the concept of conditional ProFPTrees will briefly be introduced. Section 16.3 will describe how probability information is extracted from a (conditional) ProFP-tree. Section 16.4 will introduce the *Generating Function* approach for computing the frequentness probability and the SPDF in linear time. Section 16.5 will describe how conditional ProFP-trees are built. Section 16.6 will describe the ProFP-Growth algorithm by drawing together the previous sections. The experimental evaluation will be presented in Section 16.7. Finally, Section 16.8 will conclude this chapter.

16.2 Probabilistic Frequent-Pattern Tree (ProFP-tree)

16.2.1 Components

This section will introduce a prefix-tree structure that enables the fast detection of probabilistic frequent itemsets without the costly candidate generation or multiple database scans that plague Apriori-style algorithms. The proposed structure is based on the *Frequent-Pattern tree* (*FP-tree* [104]). In contrast to the FP-tree, the ProFP-tree has the ability to compress uncertain transactions. If a dataset contains no uncertainty, it reduces to the (certain) FP-tree.

Definition 16.1 (ProFP-tree) *A* Probabilistic Frequent Pattern Tree (ProFP-tree) *is composed of the following three components:*

1. **Uncertain Item Prefix Tree (UIPT)***: A root labeled "null" pointing to a set of prefix trees, each associated with uncertain item sequences. Each node n in a prefix tree is associated with an (uncertain) item and consists of five fields:*

   - *n.item denotes the item label of the node. Let path*(n) *be the set of items on the path from root to n.*
   - *n.count is the number of* certain *occurrences of path*(n) *in the database.*
   - *n.uft, denoting "uncertain-from-this", is a set of transaction IDs* (TIDs)*. A transaction t is contained in uft if and only if n.item is uncertain in t (*$0 < P(n.item \in t) < 1$*) and* $P(path(n) \subseteq t) > 0$.
   - *n.ufp, denoting "uncertain-from-prefix", is also a set of TIDs. A transaction t is contained in ufp if and only if n.item is certain in t (*$P(n.item \in t) = 1$*) and* $0 < P(path(n) \subseteq t) < 1$.

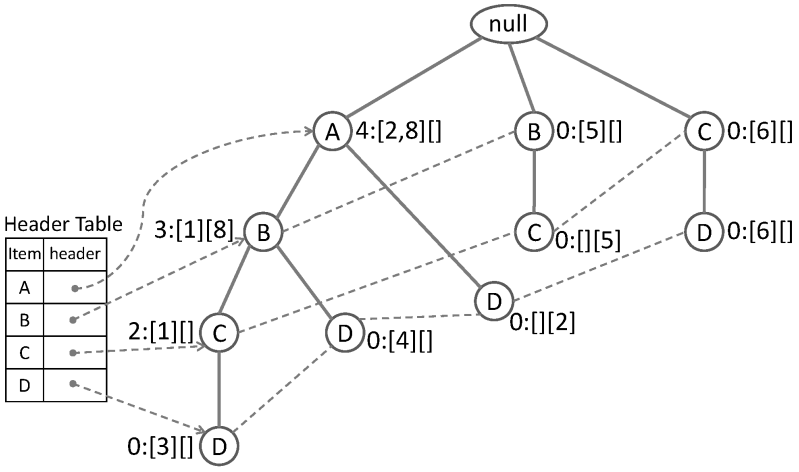


Figure 16.1: The ProFPTree generated from the uncertain transaction database given in Table 16.1: *UIPT* and *IHT*.

$(1, B) \rightarrow 0.2$
$(1, C) \rightarrow 0.5$
$(2, A) \rightarrow 0.1$
$(3, D) \rightarrow 0.4$
$(4, D) \rightarrow 0.5$
$(5, B) \rightarrow 0.1$
$(6, C) \rightarrow 0.1$
$(6, D) \rightarrow 0.5$
$(8, A) \rightarrow 0.5$

Table 16.2: *UILT*.

- *n.node-link links to the next node in the tree with the same item label if there exists one.*

2. **Item Header Table (IHT)***: This table maps all items to the first node in the* UIPT*.*

3. **Uncertain Item Lookup Table (UILT)***: This table maps* (TID, item) *pairs to the probability that* item *appears in* t_{TID} *for each transaction* t_{TID} *contained in a uft of a node n with n.item =* item*.*

The two sets, *uft* and *ufp*, are specialized fields required in order to handle the existential uncertainty of itemsets in transactions associated with $path(n)$. Here, two sets are needed in order to distinguish where the uncertainty of an itemset (path) comes from. Generally speaking, the entries in *n.uft* are used to keep track of existential uncertainties where the uncertainty is caused by *n.item*, while the entries in *ufp* keep track of uncertainties of itemsets caused by items in $path(n) - n.item$, but where *n.item* is certain.

Figure 16.1 illustrates the ProFP-tree of the example database of Table 16.1. Each node of the *UIPT* is labeled by the field *item*. The labels next to the nodes refer to the node fields *count : uft ufp*. The dotted lines denote the *node-links*. The *UILT* is illustrated in Table 16.2.

The ProFP-tree has the same advantages as a FP-tree. In particular, it avoids repeatedly scanning the database, since the uncertain item information is efficiently stored in a compact structure. Secondly, multiple transactions sharing identical prefixes can be merged into one with the number of certain occurrences registered by *count* and the uncertain occurrences reflected in the transaction sets *uft* and *ufp*.

16.2.2 ProFP-Tree Construction

The ProFP-tree construction algorithm is illustrated in Algorithm 11. Further illustration is provided by the example database of Table 16.1 and by the corresponding ProFP-tree in Figure 16.1. The (uncertain) items in the transactions are assumed to be lexicographically ordered, which is required for prefix tree construction.

First, the root of the *UIPT* labeled *null* is created (line 1). Then, the uncertain transactions are read, one at a time. While scanning the first transaction t_1, the first branch of the tree can be generated (line 4), leading to the first path composing entries of the form

$$(item, count, uft, ufp, node\text{-}link).$$

In the example of Table 16.1, the first branch of the tree is built by the following path:

$$\langle root, (A, 1, [], [], null), (B, 0, [1], [], null), (C, 0, [1], [], null)\rangle.$$

The entry "1" in the field *uft* of the nodes associated with B and C indicates that items B and C are uncertain in t_1.

The pseudocode of the insertion of a transaction t into the tree is further illustrated in Algorithm 12. The necessary steps for updating the node entries which come along with the insertion of t are illustrated in the pseudocode of Algorithm 13.

Next, the second transaction t_2 is scanned and the tree structure is updated accordingly. The itemset of transaction t_2 shares its prefix with the previous one, therefore we follow the existing path in the tree (Algorithm 12, line 3) starting at the root. Since the first item in t_2 is existentially uncertain, i.e., it exists in t_2 with a probability of 0.1, *count* of the first node in the path is not incremented. Instead, the current transaction t_2 is added to *uft* of this node (Algorithm 13, line 8). The next item in t_2 does not match with the next node on the path and, thus, it is needed to build a new branch leading to the leaf node n (Algorithm 12, line 5) with the entry $(D, 0, [], [2], null)$. Although item D is existentially certain in t_2, *count* of n is initialized with 0, because the itemset $\{A, D\}$ associated with the path from the root to node n is existentially uncertain in t_2, due to the existential uncertainty of item A. This case in indicated by setting the uncertainty flag *u_flag*. Hence, transaction t_2 is added to the *uncertain-from-prefix* (*ufp*) field of n (Algorithm 13, line 5). The resulting tree is illustrated in Figure 16.2(a).

The next transaction to be scanned is transaction t_3. Again, due to matching prefixes, it is required to follow the already existing path $\langle A, B, C\rangle$[1] while scanning the (uncertain) items in t_3. The resulting tree is illustrated in Figure 16.2(b). Since the first item A is existentially certain, *count* of the first node in the prefix path is incremented by one (Algorithm 13, line 3). The next items B and C, are registered in the tree in the same way by incrementing the *count* fields. The rationale for these *count* increments is that the corresponding itemsets are existentially certain in t_3. The final item D is processed by adding a new branch below the node C, leading to a new leaf node with the fields $(D, 0, [3], [], ptr)$, where the link *ptr* points to the next node in the tree labeled with item

[1]For illustration purposes, we use the *item* fields to address the nodes in a path.

Algorithm 11 ProFP-tree Construction.

Require: $\mathcal{T}$, *minSup*

create the root (*null*) of an *UIPT* T
initialize *IHT* and *UILT*
for all $t_i \in \mathcal{T}$ **do**
  insertTransaction(t_i, i, $T.root$, 0)
end for
return T

Algorithm 12 insertTransaction(t, id, n, u_flag)

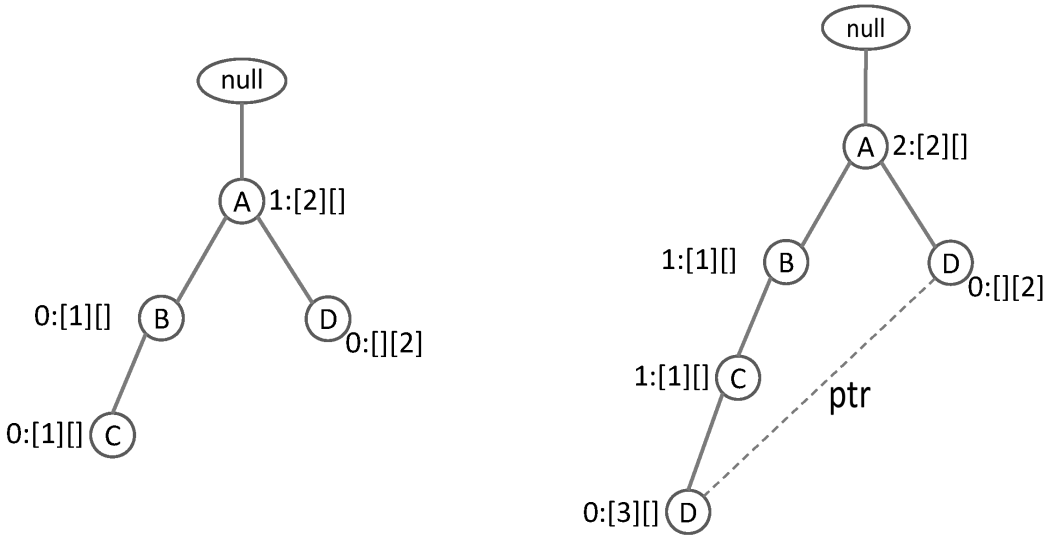
Require: transaction t, TID id, node n, u_flag

for all $x \in t$ **do**
  if n has a child n' with $n'.item = x$ **then**
    updateNodeEntries(t, id, x, n', u_flag) {follow existing path}
  else
    create new child n' of T {create new branch}
    updateNodeEntries(t, id, x, n', u_flag)
    if $x \notin IHT$ **then**
      IHT.insert(x, $ptr(n')$)
    else
      insert n' into the link list associated with x
    end if
    if $P(x \in t) < 1$ **then**
      $UILT$.insert(id, x, $P(x \in t)$)
    end if
    $n \leftarrow n'$
  end if
end for

Algorithm 13 updateNodeEntries(t, i, x, n, u_flag)

Require: transaction t, TID i, item x, node n, u_flag

if $P(x \in t) = 1$ **then**
  if $u\_flag = 0$ **then**
    $n.count \leftarrow n.count + 1$
  else
    $n.ufp$.insert(i)
  end if
else
  $n.uft$.insert(i)
  $u\_flag \leftarrow 1$
end if



(a) After inserting t_1 and t_2.

(b) After inserting t_1, t_2 and t_3.

Figure 16.2: *Uncertain item prefix tree* after insertion of the first transactions.

label D. Since item D is existentially uncertain in t_3, the *count* field is initialized with 0 and t_3 is registered in the *uft* set. The *UIPT* is completed by scanning all remaining transactions in a similar fashion.

Whenever a new item x is inserted into the tree structure, it is also inserted in the *IHT* with a pointer to the corresponding node (Algorithm 12, line 8). Otherwise, the link list associated with x is updated (line 10).

16.2.3 Construction Analysis

The construction of the ProFP-tree requires a single scan of the uncertain transaction database $\mathcal{T}$. For each processed transaction $t \in \mathcal{T}$, it is needed to follow and update or construct a single path of the tree, of a length equal to the number of items in t. Therefore, the ProFP-tree is constructed in linear time w.r.t. the size of the database.

Since the ProFP-tree is based on the original FP-tree, it inherits its compactness properties. In particular, the size of a ProFP-tree is bounded by the overall occurrences of the (un)certain items in the database and its height is bounded by the maximum number of (un)certain items in a transaction. For any transaction t in $\mathcal{T}$, there exists exactly one path in the *UIPT* starting below the *root* node. Each item in the transaction database can create no more than one node in the tree and the height of the tree is bounded by the number of items in a transaction (path). As with the FP-tree, the compression is obtained by sharing common prefixes.

The following part will show that the values stored at the nodes do not affect the bound on the size of the tree. In particular, with the following Lemma, it is possible to bound the *uncertain-from-this* (*uft*) and *uncertain-from-prefix* (*ufp*) sets.

Lemma 16.1 *Let T be the* UIPT *generated from an uncertain transaction database $\mathcal{T}$. The total space required by all the TID sets (uft and ufp) in all nodes in T is bounded by the total number of uncertain occurrences (entries in transactions with an existential probability in $(0,1)$) in $\mathcal{T}$.*

The rationale for the above lemma is that each occurrence of an uncertain item (with existential probability in $(0,1)$) in the database yields at most one TID entry in one of the TID sets assigned to a node in the tree. In general, there are three update possibilities for a node n: if the current item and all prefix items in the current transaction t are certain, there is no new entry in *uft* or *ufp*, as *count* is incremented. t is registered in $n.uft$ if and only if $n.item$ is existentially uncertain in t, while t is registered in $n.ufp$ if and only if $n.item$ is existentially certain in t, but at least one of the prefix items in t is existentially uncertain. Therefore, each occurrence of an item in $\mathcal{T}$ leads to either a count increment or a new entry in *uft* or *ufp*.

Finally, it should be clear that the size of the *UILT* is bounded by the number of uncertain (> 0 and < 1) entries in the database.

This section showed that the ProFP-tree inherits the compactness of the original FP-tree. The following section will show that the information stored in the ProFP-tree suffices to retrieve all probabilistic information required for probabilistic frequent itemset mining, thus proving completeness.

16.3 Extracting Certain and Uncertain Support Probabilities

The certain FP-Growth approach easily achieves the extraction of the support of an itemset X by summing up the support counts along the node links for X in a suitable *conditional FP-tree*. The probabilistic case, however, requires the SPDF of X. For that, it is first needed to determine both the number of certain occurrences as well as the probabilities $0 < P(X \subseteq t_i) < 1$. Both can be efficiently obtained using the ProFP-tree. To obtain the certain support of an item x, it is needed to follow the node links from the *IHT* and to accumulate both the counts and the number of transactions in which x is *uncertain-from-prefix*. The latter is counted, since we are interested in the support of x and by construction, transactions in *ufp* are known to be certain for x. To find the set of TIDs in which x is uncertain, follow the node links and accumulate all transactions that are in the *uncertain-from-this* (*uft*) list. The pseudocode of this process can be found in Algorithm 14.

Example 16.1 *By traversing the node list, it is possible to compute the certain support for item C in the ProFP-tree in Figure 16.1 by $2 + |\emptyset| + 0 + |\{t_5\}| + 0 + |\emptyset| = 3$. There is one transaction (t_5) in which C is* uncertain-from-prefix. *Similarly, it can be observed that the only transactions in which C is uncertain are t_1 and t_6. The exact appearance probabilities in these transactions can be obtained from the* UILT. *By comparing this to*

Algorithm 14 Extract Probabilities for an Item: extractProbs(x, T)

Require: item x, ProFP-tree T

{compute the certain support *count* and retrieve the uncertain TIDs (*UTIDs*)}
count $\leftarrow 0$, *UTIDs* $\leftarrow \emptyset$
for all $n \in T$ reachable from $IHT[x]$ **do**
  count $\leftarrow$ *count* $+ n.count$
  count $\leftarrow$ *count* $+ |n.ufp|$
  UTIDs $\leftarrow$ *UTIDs* $\cup\, n.uft$
end for
return *count*, *UTIDs*

Algorithm 15 Extract Probabilities for an Itemset X: computeProbabilities(X, *UTIDs*)

Require: itemset X, list of TIDs with uncertain support *UTIDs*

{compute the existential probabilities of an itemset X}
probs $\leftarrow$ []
for all $t \in$ *UTIDs* **do**
  $p \leftarrow \Pi_{x \in X}\, UILT[t,x]$
  probs.add(p)
end for
return *probs*

Table 16.1, it can be observed that the tree allows to obtain the correct certain support and the TIDs where C is uncertain.

To compute the support of an *itemset* $X = \{x_1, \ldots, x_k\}$, the conditional tree for items $x_2, \ldots, x_k$ is used and the certain support as well as the uncertain TIDs are extracted for x_1. The construction of conditional ProFP-trees will be discussed in Section 16.5.

By using the conditional tree, the above method computes the certain support of X and retrieves the set of TIDs in which itemset X is uncertain. To compute the probabilities $P(X \subseteq t) : 0 < P(X \subseteq t) < 1$, the assumption of independence among uncertain transactions is exploited (cf. Algorithm 15). Thus, it is possible to multiply the probabilities $P(x \in t)$ for each $x \in X$. The retrieval of $P(x \in t)$ is an $O(1)$ lookup in the *UILT*. Moreover, if additional information is given on the dependencies between items, this can be incorporated here.

The above part described how the certain support and all probabilities $P(X \subseteq t) : 0 < P(X \subseteq t) < 1$ can be efficiently extracted from the ProFPTree. Section 16.4 will show how this information is used to compute the SPDF of X.

16.4 Efficient Computation of Probabilistic Frequent Itemsets

This section will present a linear-time technique for computing the probabilistic support of an itemset using *Generating Functions* (*GFs*) as proposed in the context of probabilistic ranking in [154]. The problem is as follows:

Definition 16.2 *Given a set of N mutually independent but not necessarily identically distributed Bernoulli variables $\mathcal{X}_i = P(X \subseteq t_i) \in \{0, 1\}$, $1 \leq i \leq N$, compute the probability distribution of the random variable $\mathcal{S} = \sum_{i=1}^{N} \mathcal{X}_i$.*

A naive solution would be to count, for each $0 \leq i \leq N$, all possible worlds in which exactly i transactions contain itemset X and accumulate the respective probabilities. This approach, however, shows a complexity of $O(2^N)$. In Chapter 15, an approach was proposed that achieves an $O(N)$ complexity using the PBR. Generally, a runtime complexity of $O(N)$ is asymptotically optimal, since the computation involves at least $O(N)$ computations, namely $P(X \subseteq t_i)$ $\forall 1 \leq i \leq N$. The following subsection proposes a different approach that, albeit having the same linear asymptotical complexity, has other advantages.

Consider the function

$$\mathcal{F}^N(x) = \prod_{i=1}^{N} (a_i x + b_i)\,.$$

The coefficient c_i of x^i in $\mathcal{F}^N(x)$ is given by

$$c_i = \sum_{|\beta|=i} \prod_{j:\beta_j=1} a_j \prod_{j:\beta_j=0} b_j,$$

where $\beta = \langle \beta_1, \ldots, \beta_N \rangle$ is a Boolean vector, and $|\beta|$ denotes the number of 1s in β. Now consider the following *Generating Function*:

$$\mathcal{F}^j = \prod_{t \in \{t_1,\ldots,t_j\}} (P(X \subseteq t) \cdot x + (1 - P(X \subseteq t))) = \sum_{i \in \{0,\ldots,j\}} c_i \cdot x^i$$

The coefficient c_i of x^i in the expansion of $\mathcal{F}^j$ is the probability that X occurs in exactly i of the first j transactions; that is, the probability that the support of X is i in the first j transactions, since $\mathcal{F}^j$ contains at most $j + 1$ nonzero terms. From

$$\mathcal{F}^j = \mathcal{F}^{j-1} \cdot (P(X \subseteq t_j) \cdot x + (1 - P(X \subseteq t_j))),$$

it can be observed that $\mathcal{F}^j$ can be computed in $O(j)$ time, given $\mathcal{F}^{j-1}$. Since the basic case $\mathcal{F}^0 = 1 \cdot x^0 = 1$ is given by definition, the conclusion is that $\mathcal{F}^N$ can be computed in $O(N^2)$ time, if $j = N$. To reduce the complexity to $O(N)$, the fact can be exploited that it is only needed to consider the coefficients c_i in the *Generating Function* $\mathcal{F}^j$ where $i < minSup$, since

- the frequentness probability of X is defined as $P(X \text{ is frequent}) = P(\mathcal{S}(X) \geq minSup)) = 1 - P(\mathcal{S}(X) < minSup) = 1 - \sum_{i=0}^{minSup-1} c_i$ and
- a coefficient c_i in $\mathcal{F}^j$ is independent of any c_k in $\mathcal{F}^{j-1}$ where $k > i$. That means in particular that the coefficients c_k, $k \geq minSup$ are not required to compute the c_i, $i < minSup$.

Thus, keeping only the coefficients c_i where $i < minSup$, $\mathcal{F}^j$ contains at most $minSup$ coefficients, leading to a total complexity of $O(minSup \cdot N)$ ($O(N)$ as typically $minSup \ll N$, cf. Chapter 15) to compute the frequentness probability of an itemset.

Example 16.2 *As an example, consider itemset $\{A, D\}$ from the running example database in Table 16.1. Using the ProFP-tree (cf. Figure 16.1), we can efficiently extract, for each transaction t_i, the probability $P(\{A, D\} \in t_i)$, where $0 < P(\{A, D\} \in t_i) < 1$, and also the number of certain occurrences of $\{A, D\}$. Itemset $\{A, D\}$ certainly occurs in no transaction and occurs in t_2,t_3 and t_4 with a probability of 0.1, 0.4 and 0.5, respectively. Assuming that $minSup = 2$ yields the following:*

$$\mathcal{F}^1 = \mathcal{F}^0 \cdot (0.1x + 0.9) = 0.1x^1 + 0.9x^0 = 0.1x^1 + 0.9$$

$$\mathcal{F}^2 = \mathcal{F}^1 \cdot (0.4x + 0.6) = 0.04x^2 + 0.42x^1 + 0.54x^0 \stackrel{*}{=} 0.42x^1 + 0.54x^0 = 0.42x^1 + 0.54$$

$$\mathcal{F}^3 = \mathcal{F}^2 \cdot (0.5 + 0.5x) = 0.21x^2 + 0.48x^1 + 0.27x^0$$

$$\stackrel{*}{=} 0.48x^1 + 0.27x^0 = 0.48x^1 + 0.27$$

Thus, $P(\mathcal{S}(\{A, D\}) = 0) = 0.27$ and $P(\mathcal{S}(\{A, D\}) = 1) = 0.48$. This yields $P(\mathcal{S}(\{A, D\}) \geq 2) = 1 - P(\mathcal{S}(\{A, D\}) < 2) = 1 - 0.48 - 0.27 = 1 - 0.75 = 0.25$. Thus, $\{A, D\}$ is not returned as a frequent itemset if τ is greater than 0.25. Equations marked with $*$ exploit that it is only needed to compute the c_i where $i < minSup$. The coefficients where $i \geq minSup$ can, thus, be neglected.

At each iteration of computing $\mathcal{F}^i$, it can be checked whether $1 - \sum_{i<minSup} c_i \geq \tau$. If that is the case, the computation can be stopped; then, the conclusion is that the respective itemset (for which $\mathcal{F}$ is the *Generating Function*) is frequent. Intuitively, the reason is that if an itemset X is already frequent considering the first i transactions only, X will still be frequent if more transactions are considered. This intuitive pruning criterion corresponds to the pruning criterion of Lemma 15.5 proposed in Chapter 15 for the approach that utilizes the PBR.

The *Generating Function* technique can be seen as a variant of the PBR. However, using GFs instead of the complicated recursion formula provides a much cleaner view on the problem. In addition, using GFs, the SPDF can be updated easily if a transaction t_i changes its probability of containing an itemset X. That is, if the probability $p = P(X \subseteq t_i)$ changes to p', then it is simply possible to obtain the expanded polynomial by dividing

Algorithm 16 Conditional Tree Construction: buildConditionalProFPTree(T_X, w, $\mathcal{N}_w$)

Require: conditional ProFPTree T_X, item w, set of nodes $\mathcal{N}_w$ where $item = w$
1: $T_{X\cup\{w\}} \leftarrow$ clone of the subtree of T_X reachable from the IHT for w
2: **for all** $n \in T_{X\cup\{w\}} \setminus \mathcal{N}_w$ **do**
3: $n.count \leftarrow 0$, $n.uft \leftarrow \emptyset$, $n.ufp \leftarrow \emptyset$
4: **end for**
5: propagate($T_{X\cup\{w\}}$, w)
6: **return** $T_{X\cup\{w\}}$

the old SPDF by $px + (1-p)$ (using polynomial division) to remove the effect of t_i and by multiplying $p'x + (1-p')$ to incorporate the new probability of t_i containing X. That is,

$$\widehat{\mathcal{F}}^i(x) = \frac{\mathcal{F}^i(x)}{(px + 1 - p)} \cdot (p'x + 1 - p'),$$

where $\widehat{\mathcal{F}}^i$ is the *Generating Function* of the SPDF of X in the changed database containing the modified transaction t_i' instead of t_i.

16.5 Extracting Conditional ProFP-Trees

This section will describe how conditional ProFP-trees are constructed from other (potentially conditional) ProFP-trees. The method for doing this is more involved than the analogous operation for the certain FPGrowth algorithm, since it is needed to ensure that the information capturing the source of the uncertainty remains correct, i.e., whether the uncertainty at that node comes from the prefix or from the present node. Recall from Section 16.3 that this is required in order to extract the correct probabilities from the tree. A conditional ProFP-tree for itemset X (T_X) is equivalent to a ProFP-tree built on only those transactions in which X occurs with a nonzero probability. In order to generate a conditional ProFP-tree for itemset $X \cup \{w\}$ ($T_{X\cup\{w\}}$) where w occurs lexicographically prior to any item in X, the starting point is the conditional ProFP-tree for X. When $X = \emptyset$, T_X is simply the complete ProFP-tree (cf. Algorithm 16). Here, let $\mathcal{N}_w$ be the set of nodes n with $n.item = w$ (these are obtained by following the links from the IHT). Line 3 initializes the needed structures *count*, *uft* and *ufp* of all nodes $\notin \mathcal{N}_w$.

Then, $T_{X\cup\{w\}}$ is constructed by propagating the values at the nodes n (i.e., $n.count$, $n.uft$ and $n.ufp$) with $n.item = w$ upwards in the corresponding path and accumulating these at the nodes n' closer to the root. The propagation loop for each $n \in \mathcal{N}_w$ is listed in Algorithm 17. Having propagated the values for one occurrence of n, n itself is removed from $T_{X\cup\{w\}}$ (line 7).

The values for every node n' on the path from n to the root in $T_{X\cup\{w\}}$ are computed according to Algorithm 18. Let n'_{old} denote the corresponding node in T_X. The sets $n'_{old}.uft$ and $n'_{old}.ufp$ are assumed to be available in n' and are, respectively, denoted by uft_{old} and ufp_{old}. Examples are illustrated in Figure 16.3 for three instances of a transaction t. Here,

Algorithm 17 Value Propagation: propagate(T, w)

Require: ProFPTree T, item w

1: **for all** $n \in T$ accessible from IHT for w **do**
2: $n' \leftarrow n.parent$
3: **while** $n' \neq null$ **do**
4: n'.cumulateValues(n)
5: $n' \leftarrow n'.parent$
6: **end while**
7: remove n from T
8: **end for**

Algorithm 18 Cumulation of Values: cumulateValues(n)

Require: node n with conditioning item

1: $count \leftarrow count + n.count$
2: $uft \leftarrow uft \cup n.uft$
3: **for all** $t \in n.ufp$ **do**
4: **if** ufp_{old}.contains(t) **then**
5: $ufp \leftarrow ufp \cup \{t\}$
6: **else if** uft_{old}.contains(t) **then**
7: $uft \leftarrow uft \cup \{t\}$
8: **else**
9: $count \leftarrow count + 1$
10: **end if**
11: **end for**

the values for every node n'_{old} are shown to the left of the respective node, whereas the updated values of n' are shown to the right.

If t is certain, then $n'.count = n.count$ (line 1, e.g. Figure 16.3(a)). If t is uncertain, then $n'.uft = n.uft$, since the conditioning is performed on an item that is uncertain in this transaction and, hence, any node on this path in the final conditional tree will also be uncertain for this transaction (line 2, e.g. Figure 16.3(b)). If the currently collected transaction for n is uncertain from the prefix (i.e., $t \in n.ufp$), it is needed to determine whether the item $n'.item$ caused this uncertainty. If the corresponding node n'_{old} in T_X contained transaction t in its ufp (i.e., $n'_{old}.item$ was not uncertain in t), then t is also in $n'.ufp$ (line 5, e.g. node labeled with C in Figure 16.3(c)). If $n'_{old}.item$ was uncertain in t, then n'_{old} in T_X would have t listed in its uft, and this must also remain the case for the conditional tree (line 7, e.g. node labeled with B in Figure 16.3(b)). If $t \in n'.ufp$ is neither in $n'_{old}.ufp$ nor in $n'_{old}.uft$ in T_X, then it must be certain for $n'.item$, and $n'.count$ is incremented (line 9, e.g. node labeled with A in Figure 16.3(b)). Thus, it is possible to avoid storing all transactions for which an item is certain. This is a key idea in the proposed ProFP-tree.

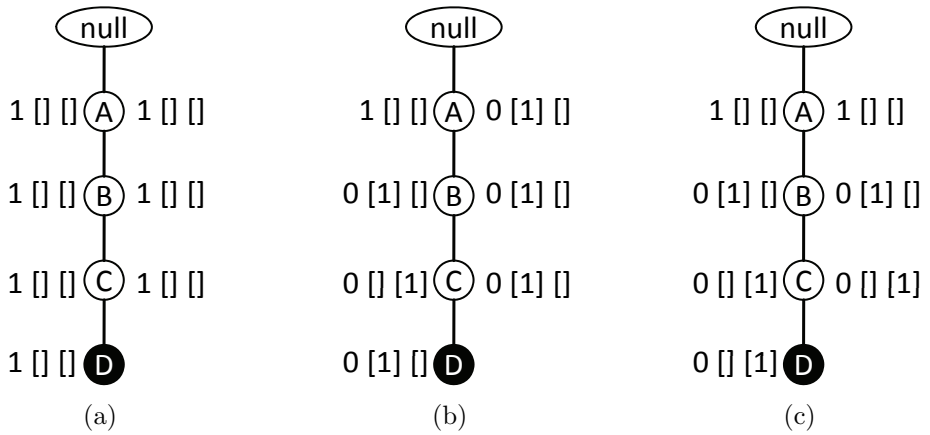


Figure 16.3: Examples for conditioning a ProFP-tree to item D, where (a) t={$(A, 1)$; $(B, 0.5)$; $(C, 1)$; $(D, 1)$}, (b) t={$(A, 1)$; $(B, 0.5)$; $(C, 1)$; $(D, 0.5)$} and (c) t={$(A, 1)$; $(B, 0.5)$; $(C, 1)$; $(D, 1)$}

16.6 ProFP-Growth Algorithm

The above part has now described the three fundamental operations of the *ProFP-Growth* algorithm: building the ProFPTree (Section 16.2), efficiently extracting the certain support and uncertain transaction probabilities from it (Section 16.3), computing the frequentness probability and determining whether an item(set) is a probabilistic frequent itemset (Section 16.4) and the construction of the conditional ProFPTrees (Section 16.5). Together with the fact that probabilistic frequent itemsets possess an anti-monotonicity property (Lemma 15.7 in Chapter 15), a similar approach to the certain FPGrowth algorithm can be used to mine all probabilistic frequent itemsets. Since, in principle, this is not substantially different from substituting the corresponding steps in FP-Growth, further details will be omitted.

16.7 Experimental Evaluation

16.7.1 Datasets and Experimental Setup

This section will present performance experiments using the proposed probabilistic frequent itemset mining algorithm, in the following denoted as **ProFP-Growth**, and will compare the results to the incremental Apriori-based solution (denoted as **ProApriori**) that was presented in Chapter 15. It will also be analyzed how various database characteristics and parameter settings affect the performance of **ProFP-Growth**. While the runtime experiments in Chapter 15 evaluated the approaches that compute the frequentness probabilities, the runtime measurements here aim at evaluating the requirements of the complete algorithms that mine frequent itemsets. It is important to note that the performance of **ProApriori** is independent of the chosen strategy (which were denoted by **AP** and **IP** in

Chapter 15). **AP** returns all frequent items in an Apriori manner (in ascending order of their size), while **IP** utilizes a priority queue w.r.t. the frequentness probability. Both approaches, however, generate the same candidate set of potentially frequent itemsets, but in a different order.

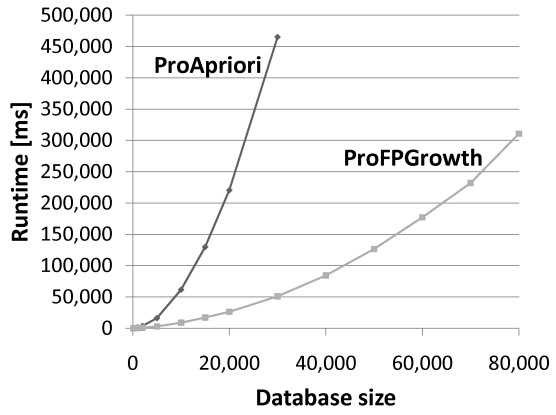
All experiments were performed on an Intel Xeon with 32 GB of RAM and a 3.0 GHz processor. The major part of the experiments used artificial datasets with a variable number of transactions and items. Each item x has a probability $P_1(x)$ of appearing for certain in a transaction, and a probability $P_0(x)$ of not appearing at all in a transaction. With a probability of $1 - P_0(x) - P_1(x)$, item x is therefore uncertain in a transaction. In this case, the probability that x exists in a transaction is picked randomly from a uniform $(0, 1)$ distribution. The sum of the probabilities of uncertainty and certainty, which is $[1 - P_0(x) - P_1(x)] + P_1(x) = 1 - P_0(x)$, corresponds to the *density* evaluated in Chapter 15. Here, the certain and uncertain occurrences will be examined separately (cf. Subsection 16.7.4).

For the scalability experiments that will be presented in Subsections 16.7.2 and 16.7.3, the number of items and transactions was scaled, and $P_0(x) = 0.5$ and $P_1(x) = 0.2$ were chosen for each item (yielding a density of 0.5). We measured the runtime required to mine all probabilistic frequent itemsets that have a minimum support of 10% of the database size with a probability of a least $\tau = 0.9$.

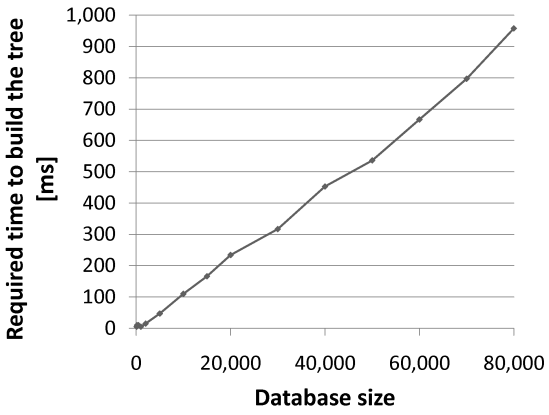
16.7.2 Effect of the Number of Transactions

The first experiment scaled the number of transactions and used 20 items (cf. Figure 16.4(a)). In this setting, **ProFP-Growth** significantly outperforms **ProApriori**; the latter suffers from the overhead of computing the frequentness probability for all generated candidates. The time required to build the ProFP-tree w.r.t. the number of transactions is illustrated in Figure 16.4(b). The observed linear runtime indicates a constant time required to insert a transaction into the tree. This is expected, since the maximum height of the ProFP-tree is equal to the number of items. Finally, the size of the ProFP-tree was evaluated for this experiment, shown in Figure 16.4(c). The number of nodes in the ProFP-tree increases and then plateaus, as the number of transactions increases. This is because new nodes have to be created for those transactions where a suffix of the transaction is not yet contained in the tree. As the number of transactions increases, the overlap between transaction prefixes increases, requiring fewer new nodes to be created. It is expected that this overlap increases faster if the items are correlated. Therefore, the next experiment evaluates the size of the ProFP-tree on subsets of the real-world dataset *accidents*[2], denoted by ACC. It consists of $340,184$ transactions and a reduced number of 20 items whose occurrences in transactions were randomized; with a probability of 0.5, each item appearing for certain in a transaction was assigned a value drawn from a uniform distribution in $(0, 1]$. The number of transactions from ACC was varied up to the first $300,000$. As can be seen in

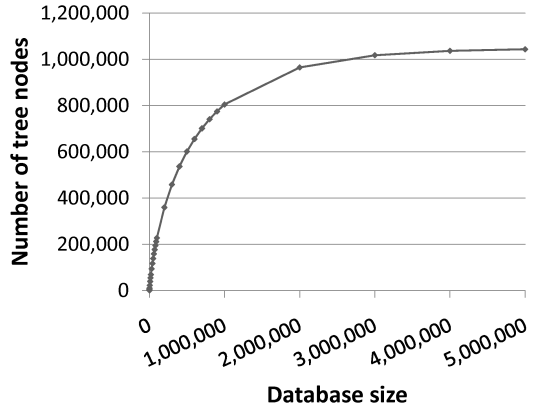
[2]The *accidents* dataset [98] was derived from the Frequent Itemset Mining Dataset Repository (`http://fimi.cs.helsinki.fi/data/`).



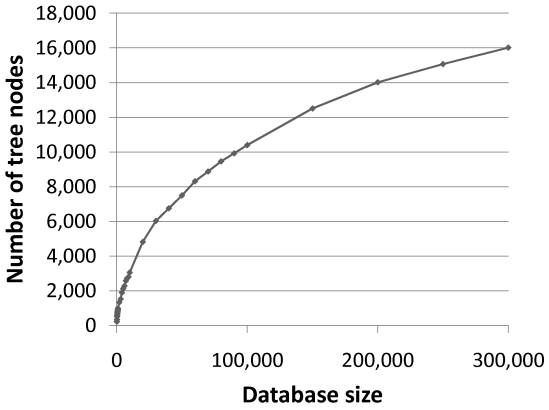
(a) Total runtime.



(b) Tree generation.

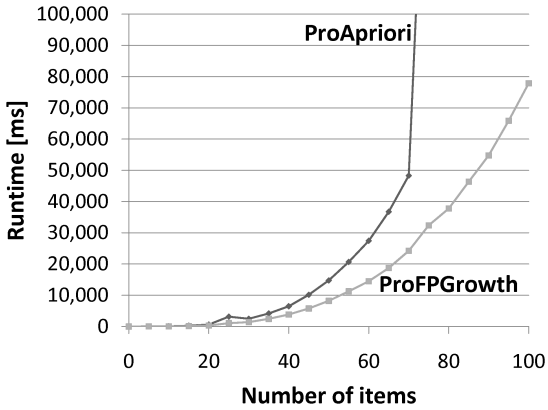


(c) Tree size (synthetic).

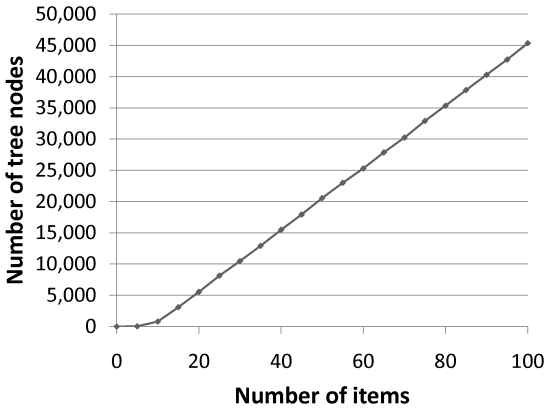


(d) Tree size (ACC).

Figure 16.4: Scalability w.r.t. the number of transactions.

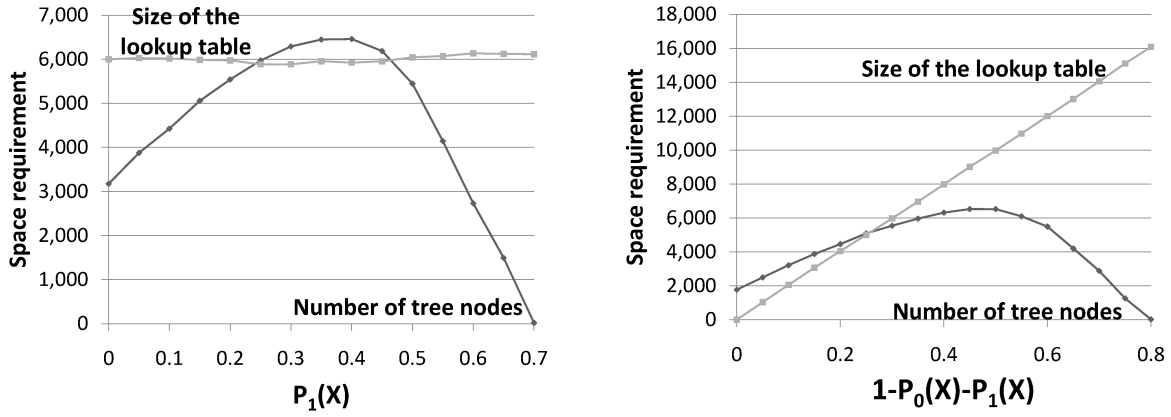


(a) Runtime.



(b) Tree size.

Figure 16.5: Scalability w.r.t. the number of items.



(a) Varying $P_1(x)$ ($1 - P_0(x) - P_1(x)$ fixed). (b) Varying $1 - P_0(x) - P_1(x)$ ($P_1(x)$ fixed).

Figure 16.6: Effect of (un)certainty on the ProFP-tree size and uncertain item lookup table.

Figure 16.4(d), there is more overlap between transactions, since the growth in the number of nodes used is slower (compared to Figure 16.4(c)).

16.7.3 Effect of the Number of Items

The next experiment scaled the number of items using $1,000$ transactions. The runtimes for 5 to 100 items can be seen in Figure 16.5(a), which shows the expected exponential runtime inherent in FIM problems. It can clearly be seen that the **ProFP-Growth** approach vastly outperforms **ProApriori**, since the latter has to cope with an exponentially increasing number of candidates. Figure 16.5(b) shows the number of nodes used in the ProFP-tree. Except for very few items, the number of nodes in the tree grows linearly.

16.7.4 Effect of Uncertainty and Certainty

In this experiment, the number of transactions was set to $N = 1,000$ and the number of items was set to 20; the parameters $P_0(x)$ and $P_1(x)$ were varied. First, the probability that items are uncertain ($1 - P_0(x) - P_1(x)$) was fixed at 0.3. $P_1(x)$ was successively increased from 0 (which means that no items exist for certain) to 0.7 (cf. Figure 16.6(a)). It can be observed that the number of nodes initially increases. This is what would be expected, since more items existing in $\mathcal{T}$ increases the nodes required. However, as the number of certain items increases, an opposite effect reduces the number of nodes in the tree. This effect is caused by the increasing overlap of the transactions – in particular, the increased number and length of shared prefixes. When $P_1(x)$ reaches 0.7 (and thus $P_0(x) = 0$), each item is contained in each transaction with a probability > 0, and, thus, all transactions contain the same items with nonzero probability. In this case, the ProFP-tree degenerates to a linear list containing exactly one node for each item. The size of the *UILT* is constant, since the

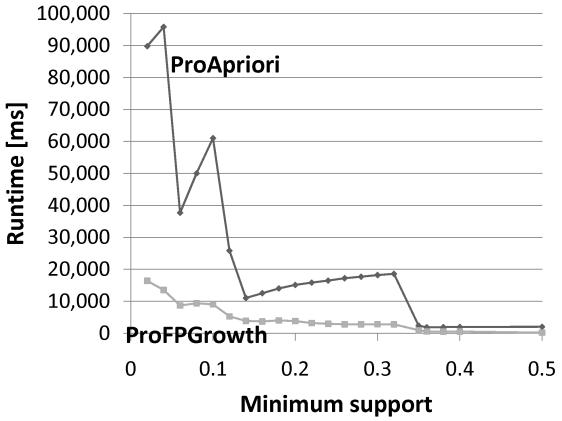


Figure 16.7: Effect of *minSup*.

expected number of uncertain items is constant at $0.3 \cdot N \cdot |\mathcal{I}| = 0.3 \cdot 1,000 \cdot 20 = 6,000$. In Figure 16.6(b), $P_1(x)$ was fixed at 0.2, and $P_0(x)$ was successively decreased from 0.8 to 0, thus increasing the probability that items are uncertain from 0 to 0.8. Here, a similar pattern can be observed as in Figure 16.6(a) for the number of nodes, for similar reasons. As expected here, the size of *UILT* increases as the number of uncertain items increases.

16.7.5 Effect of the Minimum Support

Finally, the minimum support threshold *minSup* was varied using an artificial database of 10,000 transactions and 20 items. Figure 16.7 shows the results. For low values of *minSup*, both algorithms have a high runtime due to the large number of probabilistic frequent itemsets. It can be observed that **ProFP-Growth** significantly outperforms **ProApriori** for all settings of *minSup*. If *minSup* increases to a level where only few frequent itemsets are present, the candidate generation overhead of **ProApriori** is no more significant. The unsteadiness in the curves can be explained with implementing issues w.r.t. garbage collection.

16.8 Summary

The problem of Probabilistic Frequent Itemset Mining has two components; efficiently computing the support probability distribution and the frequentness probability, and efficiently mining all probabilistic frequent itemsets. To solve the first problem in linear time, this chapter proposed a different method to Chapter 15 based on *Generating Functions*. To solve the second problem, this chapter proposed the first probabilistic frequent pattern tree and pattern growth algorithm, which achieved superior results to the approach of Chapter 15.

Part IV

Conclusions

Chapter 17

Summary

17.1 Preliminaries

This thesis presented methods for similarity processing – i.e., similarity-based querying and mining – in *multi-observation data*, which is defined by objects consisting of multiple occurrences with two key properties: *temporal variability* and *uncertainty*. Part I provided an introduction to Knowledge Discovery in Databases, to common similarity query types and to the term of multi-observation data.

Part II addressed the key property of *temporal variability* by considering data objects with a temporal ordering of multiple observations in the context of time series analysis, where many applications are faced with the existence of periodic patterns. Chapter 3 motivated the challenges for periodic pattern analysis in time series and the need of query acceleration techniques, followed by a review of related work in Chapter 4. Section 17.2 will summarize the research contributions provided in this part. These contributions contain publications in the field of activity recognition [39, 197, 198], and of the support of the data mining process by tools [41] (Subsection 17.2.1). Since similarity models for time series are often based on feature extraction methods, efficient processing of similarity queries can be supported by indexes structures for potentially high-dimensional feature spaces. The contributions in this direction [32, 33, 40] will be reviewed in Subsection 17.2.2.

In Part III, the key property of *uncertainty* was addressed, where existential dependencies among multiple observations of a data object pose diverse challenges. Similarity processing techniques coping with the presence of incomplete and uncertain information were introduced. Chapter 9 motivated the challenges with the presence of uncertain data, followed by a review of related work in Chapter 10. Section 17.3 will now summarize the contributions for the research area of probabilistic databases. These contributions contain approaches for efficient processing of probabilistic ranking and inverse ranking [43, 44, 45, 49], which will be reviewed in Subsection 17.3.1, as well as for relevant problems in probabilistic data mining [46, 47, 48], which will be summarized in Subsection 17.3.2.

17.2 Temporal Variability (Part II)

17.2.1 Time Series Analysis

Part II presented an approach of analyzing periodic time series in the context of activity recognition. The challenge with working on time series data emerges from the temporal characteristics of sequentially ordered observations. Therefore, similarity processing on time series data requires dedicated similarity measures that capture these characteristics. An introduction to time series analysis and in particular to the need of activity recognition was given in Chapter 3, followed by a review of related work in the area of time series similarity and activity recognition in Chapter 4.

Chapter 5 presented the time series analysis framework *Knowing* [41]. *Knowing* allows a fast integration of data mining techniques into the development process, so that information and data can be managed and processed more effectively. The application scenario of activity recognition utilizes the integration of *Knowing* for medical monitoring purposes.

Chapter 6 [39] provided an effective solution for activity recognition, which has been implemented and evaluated by the use of the *Knowing* framework. The approach emerged from the application scenario of [197, 198] and processes periodic time series that are collected from accelerometers. The proposed solution extends existing methods by integrating additional processing steps, such as a reconstruction of the data peaks and a reclassification step as well as a utilization of suitable features to improve the classification results. The experimental part showed an improved recognition quality in comparison with existing work.

17.2.2 Indexing of High-Dimensional Feature Spaces

The similarity model of the activity recognition approach provided in Chapter 6 is based on feature extraction methods and classification. Thus, efficient processing of similarity queries in the context of similarity-based classification can be supported by indexing solutions for potentially high-dimensional spaces.

Chapter 7 [40] addressed the variant of indexing the full-dimensional space for k-nearest neighbor queries and extended a prior technique for high-dimensional feature spaces based on vertically decomposed data. The proposed techniques support the vertical decomposition and a better approximation of vectors in the high-dimensional feature space, while they do not depend on a particular distribution of the data. Combining the techniques of the partitioning the data space and tightening the distance bounds using minimum bounding rectangles, the resulting approach achieves superior performance to prior work.

In some cases, only subsets of attributes chosen at query time may be relevant for the similarity of feature vectors. Thus, finding similar objects in the high-dimensional space can be reduced to subspaces at query time. To address this problem, Chapter 8 [32, 33] proposed and studied index-based solutions for supporting k-nearest neighbor queries queries in arbitrary subspaces of a multidimensional feature space. Therefore, two different approaches were studied. One of the main problems this chapter addressed was how to

schedule the available information from the various dimensions in order to obtain good distance approximations of the objects for an early pruning of candidates. The evaluation showed that the proposed solutions perform superior to the competitors.

17.3 Uncertainty (Part III)

17.3.1 Probabilistic Similarity Ranking

In Part III, efficient solutions for querying and mining uncertain objects in probabilistic databases were proposed and studied. In particular, the problem of probabilistic similarity ranking, including the variant of probabilistic inverse ranking, as well as two different mining tasks for uncertain data could be enhanced by extending a technique of dynamic programming.

Chapter 11 [45, 49] provided a variety of semantics to compute an unambiguous result for the problem of similarity ranking in spatially uncertain databases. Furthermore, several methods were introduced within a framework that break down the high computational complexity required to compute the rank probability distribution, i.e., the probability of an uncertain object to appear on different ranking positions w.r.t. the distance to a query object. Although the first (divide-and-conquer-based) approach achieves a significant speed-up compared to the naïve solution incorporating each possible database instance, its runtime is still exponential in the database size. Utilizing a dynamic-programming technique called *Poisson Binomial Recurrence*, this complexity could be reduced to a quadratic runtime.

The solution proposed in Chapter 12 [43] achieved a significant improvement of the runtime complexity of computing the rank probability distribution, overall yielding a linear complexity in the database size under specific assumptions. The concepts of incrementally computing the probabilities were theoretically and empirically proved to be superior to all existing approaches, as an improved variant of the *Poisson Binomial Recurrence* was introduced.

Chapter 13 [44] presented a solution to efficiently answering continuous inverse ranking queries on uncertain streams, further extending the technique of dynamic programming presented in Chapter 12. State-of-the-art approaches solving the probabilistic inverse ranking query problem for static data have not been applicable for stream data due to the originally quadratic complexity of the *Poisson Binomial Recurrence*. Chapter 13 theoretically and experimentally showed that a linear cost for probability updates can be achieved, which makes the approach applicable for uncertain stream databases.

17.3.2 Probabilistic Data Mining

In addition to the solutions for probabilistic similarity ranking, two mining problems were tackled in the context of uncertain data, further extending the dynamic-programming approach.

Chapter 14 [48] proposed an efficient approach for the detection of probabilistic *hot items*, i.e., objects for which there exists a sufficiently high population of other objects which are similar to this object. In particular, the proposed approach computes, for each object in an uncertain database, the probability to be a hot item. Therefore, methods were proposed that break down the high computational complexity required to compute this probability. Theoretical and experimental proofs showed that the proposed approach can efficiently solve the problem in polynomial time, while the competing techniques have exponential runtime.

Chapter 15 [46] transferred the concepts of efficiently solving the problem of similarity ranking in probabilistic databases to the problem of probabilistic frequent itemset mining, where the basic task is to find itemsets in an uncertain transaction database that are (highly) likely to be frequent. It could be theoretically and experimentally shown that the proposed dynamic computation technique computes the exact support probability distribution of an itemset in linear time w.r.t. the number of transactions instead of the exponential runtime of a non-dynamic computation. Furthermore, it was demonstrated that the proposed probabilistic pruning strategy allows to prune non-frequent itemsets early, leading to a large performance gain. In addition, an iterative itemset mining framework was introduced which reports the most likely frequent itemsets first. Finally, Chapter 16 [47] solved the problem of probabilistic frequent itemset mining without the expensive step of generating candidates. Therefore, the first probabilistic frequent pattern tree and pattern growth algorithm were introduced that are based on the full probabilistic information. To compute the support probability distribution in linear time, Chapter 16 proposed a different method to Chapter 15 based on the intuitive *Generating Functions*.

Chapter 18

Future Directions

18.1 Temporal Variability (Part II)

18.1.1 Time Series Analysis

Part II introduced an approach of similarity processing on time series motivated by the application scenario of activity recognition. This solution provides various potentials for further examination and analysis.

Future work with the *Knowing* framework, which was presented in Chapter 5, includes the integration of more well-known data mining frameworks and the extension of the data mining GUI for faster testing of machine learning techniques.

While the effectiveness of the solution presented in Chapter 6 was the main focus and could be proved in the experiments, the efficiency of the approach is still an open issue for future work. Commonly used acceleration techniques including dimensionality reduction and feature transformation were performed in order to reduce the computational effort of the classification step. Overall, suggesting a cost model for each step of the activity recognition process would be very valuable, as it is planned to incorporate the algorithm in the firmware on the accelerometer provided by the *Sendsor GmbH*. This should in particular include a trade-off evaluation regarding the classification accuracy and the required runtime of the feature selection step, which is currently performed via forward-backward search, as well as for the peak reconstruction and the reclassification step. The current experiments provided in Chapter 6 comprise five activity classes (walking, running, cycling, in-line skating and elliptical trainer). An important goal here is to evaluate additional activity classes like, for example, swimming or climbing stairs. Also, the evaluation then has to be performed on larger datasets. In addition, future work could include a hierarchical classification of activities w.r.t. activity intensities, as a user may, for example, want to distinguish between slow, medium and fast walking. For this purpose, the evaluated activity classes would be divided into several subclasses, e.g., a low, medium and high intensity of walking can be defined according an ordinal scale by slow, moderate or fast velocity. A definition of intensity in the context of activity analysis is given in [11], where the concrete value of the intensity is based on the *Metabolic Expenditure of Task* (MET). In order to

simplify the process of analyzing the intensity, accelerometers provide an acceptably representative basis for measuring the intensity of activities [59]. An objective definition of intensity in the context of physical activity is, in general, bound to several parameters, such as body mass [69]. In order to exploit useful intensity information from accelerometer data only, a parameter-free solution could be a meaningful goal.

18.1.2 Indexing of High-Dimensional Feature Spaces

The potentials for the indexing approaches of high-dimensional feature spaces will be suggested in the following.

For future work of the *BeyOND* approach presented in Chapter 7, a reasonable plan is to further evaluate the trade-off between split level (i.e., storage overhead) and pruning power. For one more split, the memory consumption increases exponentially in the dimensionality. If the gained pruning power does not increase by a particular threshold, a higher split level would not be necessary. Regarding the use of minimum bounding rectangles, more accurate information about potential pruning power gain could be obtained by considering the vector distribution within a subcube or a minimum volume decrease. The limitation threshold of minimum bounding rectangles is also an issue for further experiments. Furthermore, the pruning power is expected to have a direct impact on the processing time. Nevertheless, a comparison of the examined approaches in terms of efficiency could also be valuable. Also, the impact of the parameter k that determines the number of nearest neighbors could be examined. Finally, a suitable resolve order of the dimensions depending on the query vector could be examined. The prior approach [85] showed that, for particular data characteristics, additional assumptions on the resolve order could increase the pruning power significantly. Nevertheless, this would introduce another restriction and, thus, the solution could no more be generalized to any type of data. Another aim is at finding solutions to abandon the restriction that the minimum and maximum values of the feature vectors need to be known in advance.

As the prior approach [85] also provides methods for weighted queries and, thus, solves the problem for similarity search in subspaces implicitly, *BeyOND* could also be adapted in this direction. An additional evaluation could then be augmented by additional comparison partners that solve k-nearest neighbor queries in subspaces, e.g., the Partial VA-file [136] and the approaches that were presented in Chapter 8 [32, 33]. For the latter, a more extensive evaluation is also a potential for future work. In particular for the bottom-up approach, further heuristics could be studied that are based on the obtained results. Regarding the performance of both presented solutions, a trade-off approach between the bottom-up approach and the top-down approach could be investigated, since the bottom-up approach supports the lower-dimensional case, whereas the top-down approach performs better with higher dimensionality. Moreover, future work could include to perform a broad evaluation to study the impact of different data characteristics on all existing approaches for subspace similarity search.

18.1.3 Further Remarks

As already stated above, an analysis and an evaluation of the runtime requirements for the process of activity recognition is an important part of potential future work. Here, the impact of acceleration techniques should also be examined. The most effective used classifier, which was Naïve Bayes, proved to achieve a recognition result of high quality. Nevertheless, the need for a more efficient solution should take further methods into account that have not been considered so far. The current 15-dimensional feature vectors that represented the activity segments in Chapter 6 may be high-dimensional enough in order to accelerate queries by the approach of Chapter 7. The suggested indexing methods for subspace queries of Chapter 8 also promise to provide good quality and can apply if a user wants to examine arbitrary combinations of features. Using similarity-based classification methods, such as k-nearest neighbor classification, the activity recognition process could be accelerated significantly.

Classical segmentation and indexing techniques for time series, such as *iSAX* [65, 189] and the *TS-tree* [16], have not been examined in this work. In order to provide alternative solutions for activity recognition, an application of these techniques could be useful. However, since the current solution, which is working with a traditional segmentation scheme and a feature-based representation of the segments, provides excellent results, it has to be examined whether completely different approaches apply in this case.

18.2 Uncertainty (Part III)

18.2.1 Probabilistic Similarity Ranking

Part III investigated efficient solutions for probabilistic ranking approaches in uncertain databases. The potentials for further research in this direction are the following.

The problem of probabilistic similarity ranking has been extensively studied in Part III. As the solution provided in Chapter 11 could be improved by follow-up work, the logical consequence is to suggest approaches for future work based on Chapter 12. Regarding the dynamic-programming approach, this method is currently only applicable to the discrete uncertainty model based on the ULDB model. Possible future work could include an extension of the concepts to further uncertainty models. For continuous uncertainty, this has been addressed by the solution provided in [34]. Further work has recently shown that the approach can also be applied to semantically different starting positions in uncertain data, e.g., in fuzzy object databases [38].

The inverse ranking framework proposed in Chapter 13 can be easily adapted to tackle further variants of inverse ranking queries on streams; for example the probabilistic threshold inverse ranking query, that returns exactly those ranking positions for which the probability that the query object is on this rank at a particular time is greater than a user-specified parameter τ, as proposed in [158]. A further aspect of future work of is to develop an approximate approach, which efficiently copes with continuous data models. The idea is to derive, for each database object, a lower and an upper bound of the probability that

this object has a higher score than the query object. Using these approximations, it is possible to apply the concept of *Uncertain Generating Functions* [34] in order to obtain an (initial) approximated result of a probabilistic inverse ranking query, which guarantees that the true result is bounded correctly. The problem at hand is to update the *Uncertain Generating Functions* efficiently when an update is fetched from the stream.

18.2.2 Probabilistic Data Mining

Finally, future work in the context of the presented probabilistic mining solutions comprises the following tasks.

The detection of hot items, which was performed in Chapter 14, is a preliminary stage to probabilistic variants of density-based clustering. The approach is able to compute the probability of an uncertain object of being a probabilistic core point. However, further relevant steps of the clustering task, such as the computation of density-reachability and density-connectivity of objects, have not been addressed here. Thus, potential for future work lies in the integration of the probabilistic hot item detection approach in probabilistic variants of clustering methods, in particular *DBSCAN* [90] and *OPTICS* [14]. For both problems, there exist respective solutions with *FDBSCAN* [140] and *FOPTICS* [141]. However, the models used to determine the probabilistic density neglect the mutual exclusiveness of observations. The missing conditional probability in the mentioned approaches leads to approximate results only, which disqualifies these approaches from the accurate detection of hot items. The hot item problem has been applied to static data in this work. Similarly to the problem of continuous probabilistic inverse ranking, one could imagine that objects change their uncertain positions over time. Then, the incremental variant of the used dynamic-programming technique can be utilized, as proposed in Chapter 12 in order to efficiently cope with position updates.

At the current state of Chapters 15 and 16, the minimum support is chosen to be a small constant. With this assumption, a linear runtime complexity in the number of transactions can be achieved. With the introduction of an approximate approach, a solution of linear complexity can even be provided if the minimum support is chosen relative to the database size. A solution which approximates the support probability distribution based on standard parametric distributions like Gaussian or Poisson was recently published in in [31], which provides a trade-off solution between approximation quality and efficiency. Possible future plans with this model-based approach are to use it for other problems of mining uncertain data, e.g., for clustering. Also, for probabilistic frequent itemset mining, different approximation techniques, such as sampling, are planned to be examined. Moreover, here as well, an update of frequentness probabilities of items could be regarded by considering the problem of probabilistic frequent itemset mining for streams. Here, transaction updates cause items to gain or lose the state of being frequent. It could be examined how the techniques applied in this work, namely the *Poisson Binomial Recurrence* or the *Generating Functions*, support efficient updates in a continuous environment, evaluating them against existing approaches in this field.

18.3 Combining the Key Properties

While both Part II and Part III focused on the consideration of one key property of multi-observation data, the general case with both key properties, a combination of these two properties – temporal variability and uncertainty – has been neglected in this thesis. A very essential conclusion that has to be made here is that, in general, the two key properties coexist and, thus, do not exclude each other. There are various research directions dealing with this coexistence. Publications in the field of time series analysis often also include *uncertain time series*, for example [5]. In this work, however, the observations at different points in time are assumed to be independent, which is not a realistic assumption.

Working not only with observation values, but, for example, with observed spatial locations, as also considered in some contributions of Part III, leads to the work with uncertain trajectories – finally generalized as *uncertain spatio-temporal data*. Recent work in this direction includes [30, 88, 89], where most common query types have been addressed under the incorporation of statistical models.

Thus, a final goal for future work of this thesis is to examine the way of correctly integrating uncertainty under consideration of dependencies into temporal evolutions of multi-observation data as defined in the first chapter. With this, both key properties can be considered at the same time; an exemplary question here may be "Could uncertain similarity ranking be a valuable task for activity recognition?". The goal is, finally, still to find appropriate solutions of answering such queries with high quality and efficiency.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] E. Achtert, T. Bernecker, H.-P. Kriegel, E. Schubert, and A. Zimek. ELKI in time: ELKI 0.2 for the performance evaluation of distance measures for time series. In *Proceedings of the 11th International Symposium on Spatial and Temporal Databases (SSTD), Aalborg, Denmark*, pages 436–440, 2009.

[2] J. Aßfalg, T. Bernecker, H.-P. Kriegel, P. Kröger, and M. Renz. Periodic pattern analysis in time series databases. In *Proceedings of the 14th International Conference on Database Systems for Advanced Applications (DASFAA), Brisbane, Australia*, pages 354–368, 2009.

[3] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Similarity search on time series based on threshold queries. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT), Munich, Germany*, pages 276–294, 2006.

[4] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Similarity search in multimedia time series data using amplitude-level features. In *Proceedings of the 14th IEEE International MultiMedia Modeling Conference (MMM), Kyoto, Japan*, pages 123–133, 2008.

[5] J. Aßfalg, H.-P. Kriegel, P. Kröger, and M. Renz. Probabilistic similarity search for uncertain time series. In *Proceedings of the 21st International Conference on Scientific and Statistical Database Management (SSDBM), New Orleans, LA*, pages 435–443, 2009.

[6] C. C. Aggarwal, Y. Li, J. Wang, and J. Wang. Frequent pattern mining with uncertain data. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, pages 29–38, 2009.

[7] C. C. Aggarwal and P. S. Yu. A framework for clustering uncertain data streams. In *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancún, Mexico*, pages 150–159, 2008.

[8] P. Agrawal, O. Benjelloun, A. Das Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *Proceedings of the*

32nd International Conference on Very Large Data Bases (VLDB), Seoul, Korea, pages 1151–1154, 2006.

[9] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO), Chicago, IL*, pages 69–84, 1993.

[10] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), Santiago de Chile, Chile*, pages 487–499, 1994.

[11] B. E. Ainsworth, W. L. Haskell, S. D. Herrmann, N. Meckes, D. R. Bassett Jr., C. Tudor-Locke, J. L. Greer, J. Vezina, M. C. Whitt-Glover, and A. S. Leon. 2011 compendium of physical activities: a second update of codes and met values. *Medicine and Science in Sports and Exercise*, 43(8):1575–1581, 2011.

[12] B. E. Ainsworth, D. R. Jacobs Jr., and A. S. Leon. Validity and reliability of self-reported physical activity status: the lipid research clinics questionnaire. *Medicine and Science in Sports and Exercise*, 25(1):92–98, 1993.

[13] F. R. Allen, E. Ambikairajah, N. H. Lovell, and B. G. Celler. An adapted gaussian mixture model approach to accelerometry-based movement classification using time-domain features. In *Proceedings of the 28th International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS)*, pages 3600–3603, 2006.

[14] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Philadelphia, PA*, pages 49–60, 1999.

[15] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancún, Mexico*, pages 983–992, 2008.

[16] I. Assent, R. Krieger, F. Afschari, and T. Seidl. The TS-tree: efficient time series search and retrieval. In *Proceedings of the 11th International Conference on Extending Database Technology (EDBT), Nantes, France*, pages 252–263, 2008.

[17] A. Avci, S. Bosch, M. Marin-Perianu, R. Marin-Perianu, and P. Havinga. Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. *23rd International Conference on Architecture of Computing Systems (ARCS)*, pages 1–10, 2010.

[18] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Madison, WI*, pages 1–16, 2002.

[19] A. Badel, J. P. Mornon, and S. Hazout. Searching for geometric molecular shape complementarity using bidimensional surface profiles. *Journal of Molecular Graphics*, 10(4):205–211, 1992.

[20] L. Bao and S. Intille. Activity recognition from user-annotated acceleration data. In *2nd International Conference on Pervasive Computing (PERVASIVE), Vienna, Austria*, pages 1–17, 2004.

[21] N. C. Barengo, G. Hu, T. A. Lakka, H. Pekkarinen, A. Nissinen, and J. Tuomilehto. Low physical activity as a predictor for total and cardiovascular disease mortality in middle-aged men and women in finland. *European Heart Journal*, 25(24):2204–2211, 2004.

[22] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972.

[23] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Atlantic City, NJ*, pages 322–331, 1990.

[24] R. Bellman. *Adaptive Control Processes. A Guided Tour*. Princeton University Press, 1961.

[25] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB), Seoul, Korea*, pages 1249–1264, 2006.

[26] J. L. Bentley and M. Douglas McIlroy. Engineering a sort function. In *Software-Practice and Experience*, pages 1249–1265, 1993.

[27] S. Berchtold, C. Böhm, H. V. Jagadish, H.-P. Kriegel, and J. Sander. Independent Quantization: An index compression technique for high-dimensional data spaces. In *Proceedings of the 16th International Conference on Data Engineering (ICDE), San Diego, CA*, pages 577–588, 2000.

[28] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB), Bombay, India*, pages 28–39, 1996.

[29] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases, Seattle, WA*, pages 359–370, 1994.

[30] T. Bernecker, L. Chen, T. Emrich, H.-P. Kriegel, N. Mamoulis, and A. Züfle. Managing uncertain spatio-temporal data. In *2nd ACM SIGSPATIAL International Workshop on Querying and Mining Uncertain Spatio-Temporal Data (QUeST)*, pages 16–20, 2011.

[31] T. Bernecker, R. Cheng, D. W. Cheung, H.-P. Kriegel, S. D. Lee, M. Renz, F. Verhein, L. Wang, and A. Züfle. Model-based probabilistic frequent itemset mining. *Knowledge and Information Systems (KAIS)*, pages 1–37, October 2012.

[32] T. Bernecker, T. Emrich, F. Graf, H.-P. Kriegel, P. Kröger, M. Renz, E. Schubert, and A. Zimek. Subspace similarity search: Efficient k-NN queries in arbitrary subspaces. In *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management (SSDBM), Heidelberg, Germany*, pages 555–564, 2010.

[33] T. Bernecker, T. Emrich, F. Graf, H.-P. Kriegel, P. Kröger, M. Renz, E. Schubert, and A. Zimek. Subspace similarity search using the ideas of ranking and top-k retrieval. In *Proceedings of the 26th International Conference on Data Engineering (ICDE) Workshop on Ranking in Databases (DBRank), Long Beach, CA*, pages 4–9, 2010.

[34] T. Bernecker, T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle. A novel probabilistic pruning approach to speed up similarity queries in uncertain databases. In *Proceedings of the 27th International Conference on Data Engineering (ICDE), Hannover, Germany*, pages 339–350, 2011.

[35] T. Bernecker, T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, S. Zhang, and A. Züfle. Inverse queries for multidimensional spaces. In *Proceedings of the 12th International Symposium on Spatial and Temporal Databases (SSTD), Minneapolis, MN*, pages 330–347, 2011.

[36] T. Bernecker, T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, S. Zhang, and A. Züfle. Spatial inverse query processing. *GeoInformatica*, pages 1–39, August 2012.

[37] T. Bernecker, T. Emrich, H.-P. Kriegel, M. Renz, S. Zankl, and A. Züfle. Efficient probabilistic reverse nearest neighbor query processing on uncertain data. In *Proceedings of the 37th International Conference on Very Large Data Bases (VLDB), Seattle, WA*, pages 669–680, 2011.

[38] T. Bernecker, T. Emrich, H.-P. Kriegel, M. Renz, and A. Züfle. Probabilistic ranking in fuzzy object databases. In *Proceedings of the 21st ACM Conference on Information and Knowledge Management (CIKM), Maui, HI*, pages 2647–2650, 2012.

[39] T. Bernecker, F. Graf, H.-P. Kriegel, C. Mönnig, D. Dill, and C. Türmer. Activity recognition on 3D accelerometer data. Technical report, Institute for Informatics, Ludwig-Maximilians-Universität München, Germany, August 2012.

[40] T. Bernecker, F. Graf, H.-P. Kriegel, C. Mönnig, and A. Zimek. *BeyOND* – Unleashing *BOND*. In *Proceedings of the 37th International Conference on Very Large Data Bases (VLDB) Workshop on Ranking in Databases (DBRank), Seattle, WA*, pages 34–39, 2011.

[41] T. Bernecker, F. Graf, H.-P. Kriegel, N. Seiler, C. Türmer, and D. Dill. Knowing: A generic data analysis application. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT), Berlin, Germany*, pages 630–633, 2012.

[42] T. Bernecker, H.-P. Kriegel, P. Kröger, and M. Renz. TiP: Analyzing periodic time series patterns. In *Proceedings of the 12th International Symposium on Spatial and Temporal Databases (SSTD), Minneapolis, MN*, pages 491–495, 2011.

[43] T. Bernecker, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle. Scalable probabilistic similarity ranking in uncertain databases. *IEEE Transactions on Knowledge and Data Engineering*, 22(9):1234–1246, 2010.

[44] T. Bernecker, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle. Continuous inverse ranking queries in uncertain streams. In *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management (SSDBM), Portland, OR*, pages 37–54, 2011.

[45] T. Bernecker, H.-P. Kriegel, and M. Renz. ProUD: probabilistic ranking in uncertain databases. In *Proceedings of the 20th International Conference on Scientific and Statistical Database Management (SSDBM), Hong Kong, China*, pages 558–565, 2008.

[46] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Züfle. Probabilistic frequent itemset mining in uncertain databases. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, pages 119–128, 2009.

[47] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Züfle. Probabilistic frequent pattern growth for itemset mining in uncertain databases. In *Proceedings of the 24th International Conference on Scientific and Statistical Database Management (SSDBM), Chania, Crete, Greece*, pages 38–55, 2012.

[48] T. Bernecker, H.-P. Kriegel, M. Renz, and A. Züfle. Hot item detection in uncertain data. In *Proceedings of the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Bangkok, Thailand*, pages 673–680, 2009.

[49] T. Bernecker, H.-P. Kriegel, M. Renz, and A. Züfle. Probabilistic ranking in uncertain vector spaces. In *Proceedings of the 2nd International Workshop on Managing Data Quality in Collaborative Information Systems (MCIS), Brisbane, Australia*, pages 122–136, 2009.

[50] G. Beskales, M. A. Soliman, and I. F. IIyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. *Proceedings of the VLDB Endowment*, 1(1):326–339, 2008.

[51] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory (ICDT), Jerusalem, Israel*, pages 217–235, 1999.

[52] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[53] C. Böhm, B. Braunmüller, M. Breunig, and H.-P. Kriegel. High performance clustering based on the similarity join. In *Proceedings of the 9th ACM Conference on Information and Knowledge Management (CIKM), Washington, DC*, pages 298–305, 2000.

[54] C. Böhm, A. Pryakhin, and M. Schubert. The Gauss-tree: Efficient object identification in databases of probabilistic feature vectors. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE), Atlanta, GA*, page 9, 2006.

[55] C. Böhm, A. Pryakhin, and M. Schubert. Probabilistic ranking queries on gaussians. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM), Vienna, Austria*, pages 169–178, 2006.

[56] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.

[57] A. Björck. Least squares problems. In *Encyclopedia of Optimization*, pages 1856–1866. 2009.

[58] C. V. C. Bouten, K. T. M. Koekkoek, M. Verduin, R. Kodde, and J. D. Janssen. A triaxial accelerometer and portable data processing unit for the assessment of daily physical activity. *IEEE Transactions on Biomedical Engineering*, 44(3):136–147, 1997.

[59] C. V. C. Bouten, W. P. Verboeket-van de Venne, K. R. Westerterp, M. Verduin, and J. D. Janssen. Daily physical activity assessment: comparison between movement registration and doubly labeled water. *Journal of Applied Physiology*, 81(2):1019–1026, 1996.

[60] G. E. P. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976.

[61] S. Brecheisen, H.-P. Kriegel, P. Kröger, M. Pfeifle, M. Schubert, and A. Zimek. Density-based data analysis and similarity search. In *Multimedia Data Mining and Knowledge Discovery*, pages 94–115. 2007.

[62] E. O. Brigham and C. K. Yuen. The fast fourier transform. *IEEE Transactions on Systems, Man and Cybernetics*, 8(2), 1978.

[63] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using R-trees. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Washington, DC*, pages 237–246, 1993.

[64] Y. Cai and R. Ng. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Paris, France*, pages 599–610, 2004.

[65] A. Camerra, T. Palpanas, J. Shieh, and E. J. Keogh. iSAX 2.0: indexing and mining one billion time series. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM), Sydney, Australia*, pages 58–67, 2010.

[66] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering (ICDE), Sydney, Australia*, pages 126–133, 1999.

[67] M. A. Cheema, X. Lin, W. Wang, W. Zhang, and J. Pei. Probabilistic reverse nearest neighbor queries on uncertain data. *IEEE Transactions on Knowledge and Data Engineering*, 22(4):550–564, 2010.

[68] J. Chen and R. Cheng. Efficient evaluation of imprecise location-dependent queries. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey*, pages 586–595, 2007.

[69] K. Y. Chen and M. Sun. Improving energy expenditure estimation by using a triaxial accelerometer. *Journal of Applied Physiology*, 83(6):2112–2122.

[70] L. Chen and R. Ng. On the marriage of Lp-norms and edit distance. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada*, pages 792–803, 2004.

[71] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Baltimore, ML*, pages 491–502, 2005.

[72] Y. Chen, K. Chen, and M. A. Nascimento. Effective and efficient shape-based pattern detection over streaming time series. *IEEE Transactions on Knowledge and Data Engineering*, 24(2):265–278, 2012.

[73] Y. Chen, M. A. Nascimento, B. C. Ooi, and A. K. H. Tung. SpADe: On shape-based pattern detection in streaming time series. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey*, pages 786–795, 2007.

[74] R. Cheng, J. Chen, M. Mokbel, and C. Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancún, Mexico*, pages 973–982, 2008.

[75] R. Cheng, L. Chen, J. Chen, and X. Xie. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In *Proceedings of the 12th International Conference on Extending Database Technology (EDBT), Saint-Petersburg, Russia*, pages 672–683, 2009.

[76] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), San Diego, CA*, pages 551–562, 2003.

[77] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1112–1127, 2004.

[78] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada*, pages 876–887, 2004.

[79] C.-K. Chui and B. Kao. A decremental approach for mining frequent itemsets from uncertain data. In *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Osaka, Japan*, pages 64–75, 2008.

[80] C.-K. Chui, B. Kao, and E. Hung. Mining frequent itemsets from uncertain data. In *Proceedings of the 11th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Nanjing, China*, pages 47–58, 2007.

[81] P. Comon. Independent component analysis, a new concept? *Signal Processing*, 36(3):287–314, 1994.

[82] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Beijing, China*, pages 281–292, 2007.

[83] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected results. In *Proceedings of the 25th International Conference on Data Engineering (ICDE), Shanghai, China*, pages 305–316, 2009.

[84] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4):523–544, 2007.

[85] A. P. de Vries, N. Mamoulis, N. Nes, and M. Kersten. Efficient k-NN search on vertically decomposed data. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Madison, WI*, pages 322–333, 2002.

[86] T. Eiter and H. Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34(2):103–133, 1997.

[87] T. Emrich, F. Graf, H.-P. Kriegel, M. Schubert, M. Thoma, and A. Cavallaro. CT slice localization via instance-based regression. In *Proceedings of the SPIE Medical Imaging Conference 2010: Image Processing (SPIE), San Diego, CA*, volume 7623, page 762320, 2010.

[88] T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle. Indexing uncertain spatio-temporal data. In *Proceedings of the 21st ACM Conference on Information and Knowledge Management (CIKM), Maui, HI*, pages 395–404, 2012.

[89] T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle. Querying uncertain spatio-temporal data. In *Proceedings of the 28th International Conference on Data Engineering (ICDE), Washington, DC*, 2012.

[90] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd ACM International Conference on Knowledge Discovery and Data Mining (KDD), Portland, OR*, pages 226–231, 1996.

[91] M. Ester and J. Sander. *Knowledge Discovery in Databases: Techniken und Anwendungen*. Springer, 2000.

[92] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.

[93] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Minneapolis, MN*, pages 419–429, 1994.

[94] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *Proceedings of the 2nd ACM International Conference on Knowledge Discovery and Data Mining (KDD), Portland, OR*, pages 82–88, 1996.

[95] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[96] A. Follmann, M. A. Nascimento, A. Züfle, M. Renz, P. Kröger, and H.-P. Kriegel. Continuous probabilistic count queries in wireless sensor networks. In *Proceedings of the 12th International Symposium on Spatial and Temporal Databases (SSTD), Minneapolis, MN*, pages 279–296, 2011.

[97] Information Centre for Health and Social Care. Health survey for england - 2008. *National Health Service*, 1, 2008.

[98] K. Geurts, G. Wets, T. Brijs, and K. Vanhoof. Profiling high frequency accident locations using association rules. In *Proceedings of the 82nd Annual Meeting of the Transportation Research Board (TRB), Washington, DC*, page 18, 2003.

[99] J. M. Geusebroek, G. J. Burghouts, and A.W.M. Smeulders. The Amsterdam Library of Object Images. *International Journal of Computer Vision*, 61(1):103–112, 2005.

[100] H. Ghasemzadeh, V. Loseu, E. Guenterberg, and R. Jafari. Sport training using body sensor networks: a statistical approach to measure wrist rotation for golf swing. In *Proceedings of the 4th International Conference on Body Area Networks (BodyNets)*, pages 2:1–2:8, 2009.

[101] A. Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Boston, MA*, pages 47–57, 1984.

[102] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations*, 11(1):10–18, 2009.

[103] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Academic Press, 2nd edition, 2006.

[104] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, 29(2):1–12, 2000.

[105] X. He. Incremental semi-supervised subspace learning for image retrieval. In *Proceedings of the 13th ACM International Conference on Multimedia (ACM MM), Singapore*, pages 2–8, 2005.

[106] E. A. Heinz, K. Kunze, M. Gruber, D. Bannach, and P. Lukowicz. Using wearable sensors for real-time recognition tasks in games of martial arts - an initial experiment. In *Proceedings of the 2nd IEEE Symposium on Computational Intelligence and Games (CIG), Reno/Lake Tahoe, NV*, pages 98–102, 2006.

[107] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Proceedings of the 4th International Symposium on Large Spatial Databases (SSD), Portland, ME*, pages 83–95, 1995.

[108] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Vancouver, BC*, pages 673–686, 2008.

[109] N. Hubig, A. Züfle, T. Emrich, M. A. Nascimento, M. Renz, and H.-P. Kriegel. Continuous probabilistic sum queries in wireless sensor networks with ranges. In *Proceedings of the 24th International Conference on Scientific and Statistical Database Management (SSDBM), Chania, Crete, Greece*, pages 96–105, 2012.

[110] Y. Iijima and Y. Ishikawa. Finding probabilistic nearest neighbors for query objects with imprecise locations. In *Proceedings of the 10th International Conference on Mobile Data Management (MDM), Taipei, Taiwan*, pages 52–61, 2009.

[111] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, 2008.

[112] Y. Ishikawa, Y. Iijima, and J. X. Yu. Spatial range querying for gaussian-based imprecise query objects. In *Proceedings of the 25th International Conference on Data Engineering (ICDE), Shanghai, China*, pages 676–687, 2009.

[113] T. S. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), New Orleans, LA*, pages 346–355, 2007.

[114] J. Jestes, G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 23(12):1903–1917, 2011.

[115] C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin. Sliding-window top-k queries on uncertain streams. *The VLDB Journal*, 1(1):301–312, 2008.

[116] I. T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 2nd edition, 2002.

[117] M. Mateo K. Leung and D. Brajczuk. A tree-based approach for frequent pattern mining from uncertain data. In *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Osaka, Japan*, 2008.

[118] D. M. Karantonis, M. R. Narayanan, M. Mathie, N. H. Lovell, and B. G. Celler. Implementation of a real-time human movement classifier using a triaxial accelerometer for ambulatory monitoring. *IEEE Transactions on Information Technology in Biomedicine*, 10(1):156–167, 2006.

[119] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Tucson, AZ*, pages 369–380, 1997.

[120] E. J. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems (KAIS)*, 3(3):263–286, 2001.

[121] E. J. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Santa Barbara, CA*, pages 151–162, 2001.

[122] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *Proceedings of the 1st IEEE International Conference on Data Mining (ICDM), San Jose, CA*, pages 289–296, 2001.

[123] E. J. Keogh and M. J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Proceedings of the 4th ACM International Conference on Knowledge Discovery and Data Mining (KDD), New York City, NY*, pages 239–243, 1998.

[124] E. J. Keogh and M. J. Pazzani. Relevance feedback retrieval of time series data. In *Proceedings of the 22nd International Conference on Research and Development in Information Retrieval (SIGIR), Berkeley, CA*, pages 183–190, 1999.

[125] E. J. Keogh and P. Smyth. A probabilistic approach to fast pattern matching in time series databases. In *Proceedings of the 3rd ACM International Conference on Knowledge Discovery and Data Mining (KDD), Newport Beach, CA*, pages 24–30, 1997.

[126] A. M. Khan, Y. K. Lee, and T.-S. Kim. Accelerometer signal-based human activity recognition using augmented autoregressive model coefficients and artificial neural nets. In *Proceedings of the 30th International Conference of the IEEE Engineering in Medicine and Biology Society EMBS 2008*, pages 5172–5175, 2008.

[127] A. M. Khan, Y.-K. Lee, S.-Y. Lee, and T.-S. Kim. A triaxial accelerometer-based physical-activity recognition via augmented-signal features and a hierarchical recognizer. *IEEE Transactions on Biomedical Engineering*, 14(5):1166–1172, 2010.

[128] F. Korn, H. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Tucson, AZ*, pages 289–300, 1997.

[129] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB), Bombay, India*, pages 215–226, 1996.

[130] M. Koskela, J. Laaksonen, and E. Oja. Use of image subset features in image retrieval with self-organizing maps. In *Proceedings of the 3rd International Conference on Image and Video Retrieval (CIVR), Dublin, Ireland*, pages 508–516, 2004.

[131] A. Koski, M. Juhola, and M. Meriste. Syntactic recognition of ecg signals by attributed finite automata. *Pattern Recognition*, 28(12):1927–1940, 1995.

[132] A. Krause, M. Ihmig, E. Rankin, D. Leong, Smriti Gupta, D. Siewiorek, A. Smailagic, M. Deisher, and U. Sengupta. Trading off prediction accuracy and power consumption for context-aware wearable computing. In *Proceedings of the 9th International Symposium on Wearable Computers (ISWC), Osaka, Japan*, pages 20–26, 2005.

[133] A. Krause, D. Siewiorek, A. Smailagic, and J. Farringdon. Unsupervised, dynamic identification of phsiological and activity context in wearable computing. In *Proceedings of the 7th International Symposium on Wearable Computers (ISWC), White Plains, NY*, pages 88–97, 2003.

[134] H.-P. Kriegel, T. Bernecker, M. Renz, and A. Züfle. Probabilistic join queries in uncertain databases (a survey of join methods for uncertain data). In *Managing and Mining Uncertain Data*, pages 257–298. 2009.

[135] H.-P. Kriegel, P. Kröger, P. Kunath, and M. Renz. Generalizing the optimality of multi-step k-nearest neighbor query processing. In *Proceedings of the 10th International Symposium on Spatial and Temporal Databases (SSTD), Boston, MA*, pages 75–92, 2007.

[136] H.-P. Kriegel, P. Kröger, M. Schubert, and Z. Zhu. Efficient query processing in arbitrary subspaces using vector approximations. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM), Vienna, Austria*, pages 184–190, 2006.

[137] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1):1–58, 2009.

[138] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Probabilistic similarity join on uncertain data. In *Proceedings of the 11th International Conference on Database Systems for Advanced Applications (DASFAA), Singapore*, pages 295–309, 2006.

[139] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA), Bangkok, Thailand*, pages 337–348, 2007.

[140] H.-P. Kriegel and M. Pfeifle. Density-based clustering of uncertain data. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Chicago, IL*, pages 672–677, 2005.

[141] H.-P. Kriegel and M. Pfeifle. Hierarchical density-based clustering of uncertain data. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM), Houston, TX*, pages 689–692, 2005.

[142] H.-P Kriegel, A. Pryakhin, and M. Schubert. An EM-approach for clustering multi-instance objects. In *Proceedings of the 10th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Singapore*, pages 139–148, 2006.

[143] H.-P. Kriegel, M. Renz, M. Schubert, and A. Züfle. Statistical density prediction in traffic networks. In *Proceedings of the 8th SIAM International Conference on Data Mining (SDM), Atlanta, GA*, pages 692–703, 2008.

[144] H.-P. Kriegel and B. Seeger. Multidimensional order preserving linear hashing with partial expansions. In *International Conference on Database Theory*, pages 203–220, 1986.

[145] S. A. Kripke. A completeness theorem in modal logic. *J. Symb. Log.*, 24(1):1–14, 1959.

[146] J. R. Kwapisz, G. M. Weiss, and S. A. Moore. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.*, 12:74–82, 2011.

[147] K. Lange. *Numerical analysis for statisticians*. Statistics and computing. 1999.

[148] V. Lavrenko, M. Schmill, D. Lawrie, P. Ogilvie, D. Jensen, and J. Allan. Mining of concurrent text and time series. In *KDD-2000 Workshop on Text Mining*, pages 37–44, 2000.

[149] M. C. K. Lee, M. Ye, and W.-C. Lee. Reverse ranking query over imprecise spatial data. In *Proceedings of the 1st International Conference on Computing for Geospatial Research & Application (COM.Geo), Washington, DC*, pages 17:1–17:8, 2010.

[150] C. K.-S. Leung, C. L. Carmichael, and B. Hao. Efficient mining of frequent patterns from uncertain data. In *Proceedings of the 7th IEEE International Conference on Data Mining Workshops (ICDMW), Omaha, NE*, pages 489–494, 2007.

[151] C. K.-S. Leung and S. K. Tanbeer. Fast tree-based mining of frequent itemsets from uncertain data. In *Proceedings of the 17th International Conference on Database Systems for Advanced Applications (DASFAA), Busan, South Korea*, pages 272–287, 2012.

[152] C. Li. *Enabling Data Retrieval: By Ranking and Beyond*. PhD thesis, University of Illinois, Urbana-Champaign, 2007.

[153] C.-S. Li, P. S. Yu, and V. Castelli. MALM: a framework for mining sequence database at multiple abstraction levels. In *Proceedings of the 7th ACM Conference on Information and Knowledge Management (CIKM), Bethesda, MD*, pages 267–272, 1998.

[154] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *Proceedings of the VLDB Endowment*, 2(1):502–513, 2009.

[155] X. Lian and L. Chen. Probabilistic ranked queries in uncertain databases. In *Proceedings of the 11th International Conference on Extending Database Technology (EDBT), Nantes, France*, pages 511–522, 2008.

[156] X. Lian and L. Chen. Similarity search in arbitrary subspaces under Lp-norm. In *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancún, Mexico*, pages 317–326, 2008.

[157] X. Lian and L. Chen. Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. *The VLDB Journal*, 18(3):787–808, 2009.

[158] X. Lian and L. Chen. Probabilistic inverse ranking queries over uncertain data. In *Proceedings of the 14th International Conference on Database Systems for Advanced Applications (DASFAA), Brisbane, Australia*, pages 35–50, 2009.

[159] J. Lin, E. J. Keogh, S. Lonardi, and B. Y. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD), San Diego, CA*, pages 2–11, 2003.

[160] J. Lin, E. J. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[161] W. Litwin. Linear hashing: A new tool for file and table addressing. In *Proceedings of the 6th International Conference on Very Large Data Bases (VLDB), Montreal, Canada*, pages 212–223, 1980.

[162] V. Ljosa and A. K. Singh. APLA: Indexing arbitrary probability distributions. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey*, pages 946–955, 2007.

[163] J. E. Manson, M. J. Stampfer, G. A. Colditz, W. C. Willett, B. Rosner, C. H. Hennekens, F. E. Speizer, E. B. Rimm, and A. S. Krolewski. Physical activity and incidence of non-insulin-dependent diabetes mellitus in women. *The Lancet*, 338(8770):774–778, 1991.

[164] U. Maurer, A. Smailagic, D.P. Siewiorek, and M. Deisher. Activity recognition and monitoring using multiple sensors on different body positions. In *International Workshop on Wearable and Implantable Body Sensor Networks (BSN), Cambridge, MA*, pages 116–121, 2006.

[165] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Philadelphia, PA*, pages 935–940, 2006.

[166] J. Mäntyjärvi, J. Himberg, and T. Seppänen. Recognizing human motion with multiple acceleration sensors. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC), Tucson, AZ*, 2001.

[167] Y. Morinaka, M. Yoshikawa, T. Amagasa, and S. Uemura. The L-index: An indexing structure for efficient subsequence matching in time sequence databases. In *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Hong Kong, China*, 2001.

[168] S. Muthukrishnan. Data streams: algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005.

[169] M. N. Nyan, F. E. H. Tay, K. H. W. Seah, and Y. Y. Sitoh. Classification of gait patterns in the time-frequency domain. *Journal of Biomechanics*, 39(14):2647–2656, 2006.

[170] K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(6):559–572, 1901.

[171] S. Pirttikangas, K. Fujinami, and T. Nakajima. Feature selection and activity recognition from wearable sensors. In *Proceedings of the 3rd International Symposium on Ubiquitous Computing Systems (UCS),Seoul, Korea*, pages 516–527, 2006.

[172] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods – Support Vector Learning*, pages 185–208. 1998.

[173] S. J. Preece, J. Y. Goulermas, L. P. J. Kenney, and D. Howard. A comparison of feature extraction methods for the classification of dynamic activities from accelerometer data. *IEEE Transactions on Biomedical Engineering*, 56(3):871–879, 2009.

[174] Y. Qu, C. Wang, and X. S. Wang. Supporting fast search in time series for movement patterns in multiple scales. In *Proceedings of the 7th ACM Conference on Information and Knowledge Management (CIKM), Bethesda, MD*, pages 251–258, 1998.

[175] R project. `http://www.r-project.org/`.

[176] C. A. Ratanamahatana, E. J. Keogh, A. J. Bagnall, and S. Lonardi. A novel bit level time series representation with implication of similarity search and clustering. In *Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Hanoi, Vietnam*, pages 771–777, 2005.

[177] C. Ré, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probalistic databases. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey*, pages 886–895, 2007.

[178] C. Ré, J. Letchner, M. Balazinksa, and D. Suciu. Event queries on correlated probabilistic streams. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Vancouver, BC*, pages 715–728, 2008.

[179] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the 1st IEEE International Conference on Neural Networks (ICNN), San Francisco, CA*, pages 586–591, 1993.

[180] R. Ross, V. S. Subrahmanian, and J. Grant. Aggregate operators in probabilistic databases. *Journal of the ACM*, 52(1):54–101, 2005.

[181] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: foundations of research. pages 673–695. 1988.

[182] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26:43–49, 1978.

[183] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[184] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

[185] T. Seidl and H.-P. Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB), Athens, Greece*, pages 506–515, 1997.

[186] N. Seiler, T. Bernecker, F. Graf, C. Türmer, D. Dill, H.-P. Kriegel, and B. Wolf. MedMon – Eine Applikation zur Auswertung medizinischer Sensordaten. In *Electronics goes Medical (EgM), Munich, Germany*, 2012.

[187] R. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey*, pages 596–605, 2007.

[188] H. Shatkay and S. B. Zdonik. Approximate queries and representations for large data sequences. In *Proceedings of the 12th International Conference on Data Engineering (ICDE), New Orleans, LA*, pages 536–545, 1996.

[189] J. Shieh and E. J. Keogh. iSAX: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Las Vegas, NV*, pages 623–631, 2008.

[190] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[191] M. A. Soliman and I. F. Ilyas. Ranking with uncertain scores. In *Proceedings of the 25th International Conference on Data Engineering (ICDE), Shanghai, China*, pages 317–328, 2009.

[192] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey*, pages 896–905, 2007.

[193] A. Sugimoto, Y. Hara, T. W. Findley, and K. Yoncmoto. A useful method for measuring daily physical activity by a three-direction monitor. *Scandinavian Journal of Rehabilitation Medicine*, 29(1):37–42, 1997.

[194] J. Tang, Z. Chen, A. W.-C. Fu, and D. W. Cheung. Enhancing effectiveness of outlier detections for low density patterns. In *Proceedings of the 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Taipei, Taiwan*, pages 535–548, 2002.

[195] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), Trondheim, Norway*, pages 922–933, 2005.

[196] C. Türmer. Konzeptionierung eines Aktivitätsmonitoring-Systems für medizinische Applikationen mit dem 3D-Accelerometer der Sendsor GmbH. Master's thesis, Technische Universität München, Germany, 2009.

[197] C. Türmer, D. Dill, A. Scholz, M. Gül, T. Bernecker, F. Graf, H.-P. Kriegel, and B. Wolf. Concept of a medical activity monitoring system improving the dialog between doctors and patients concerning preventions, diagnostics and therapies. In *Forum Medizin 21, Evidenzbasierte Medizin (EbM), Salzburg, Austria*, 2010.

[198] C. Türmer, D. Dill, A. Scholz, M. Gül, A. Stautner, T. Bernecker, F. Graf, and B. Wolf. Conceptual design for an activity monitoring system concerning medical applications using triaxial accelerometry. In *Austrian Society for Biomedical Engineering (BMT), Rostock, Germany*, 2010.

[199] F. Verhein and S. Chawla. Geometrically inspired itemset mining. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM), Hong Kong, China*, pages 655–666, 2006.

[200] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings of the 18th International Conference on Data Engineering (ICDE), San Jose, CA*, pages 673–684, 2002.

[201] H. Vullings, M. Verhaegen, and H. Verbruggen. ECG segmentation using time-warping. 1280:275–285, 1997.

[202] C. Wang and X. S. Wang. Supporting content-based searches on time series via approximation. In *Proceedings of the 12th International Conference on Scientific and Statistical Database Management (SSDBM), Berlin, Germany*, pages 69–81, 2000.

[203] S. Wang, J. Yang, N. Chen, X. Chen, and Q. Zhang. Human activity recognition with user-free accelerometers in the sensor networks. In *Proceedings of the IEEE International Conference on Neural Networks and Brain (ICNN&B), Beijing, China*, pages 1212 –1217, 2005.

[204] D. E. R. Warburton, C. W. Nicol, and S. S. D. Bredin. Health benefits of physical activity: the evidence. *Canadian Medical Association Journal*, 174(6):801–809, 2006.

[205] J. A. Ward, P. Lukowicz, and G. Tröster. Gesture spotting using wrist worn microphone and 3-axis accelerometer. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies (sOc-EUSAI), Grenoble, France*, pages 99–104, 2005.

[206] R. A. Washburn and H. J. Montoye. The assessment of physical activity by questionnaire. *American Journal of Epidemiology*, 123(4):563–576, 1986.

[207] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB), New York City, NY*, pages 194–205, 1998.

[208] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.

[209] O. Wolfson, P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3):257–387, 1999.

[210] Y. Xia, Y. Yang, and Y. Chi. Mining association rules with non-uniform privacy concerns. In *Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery (DKMD), Paris, France*, pages 27–34, 2004.

[211] A. K. Yancey, C. M. Wold, W. J. McCarthy, M. D. Weber, B. Lee, P. A. Simon, and J. E. Fielding. Physical inactivity and overweight among los angeles county adults. *American Journal of Preventive Medicine*, 27(2):146–152, 2004.

[212] J.-Y. Yang, J.-S. Wang, and Y.-P. Chen. Using acceleration measurements for activity recognition: An effective learning algorithm for constructing neural classifiers. *Pattern Recognition Letters*, 29:2213–2220, 2008.

[213] B. K. Yi, H. Jagadish, and C. Faloutsos. Fast time sequence indexing for arbitrary Lp norms. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB), Cairo, Egypt*, pages 385–394, 2000.

[214] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases. In *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancún, Mexico*, pages 1406–1408, 2008.

[215] M. L. Yiu, N. Mamoulis, X. Dai, Y. Tao, and M. Vaitis. Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data. *IEEE Transactions on Knowledge and Data Engineering*, 21(1):108–122, 2009.

[216] H. Yoon, K. Yang, and C. Shahabi. Feature subset selection and feature ranking for multivariate time series. *Knowledge and Data Engineering, IEEE Transactions on*, 17(9):1186–1198, 2005.

[217] Q. Zhang, F. Li, and K. Yi. Finding frequent items in probabilistic data. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Vancouver, BC*, pages 819–832, 2008.

[218] T. Zhang. Adaptive forward-backward greedy algorithm for sparse learning with linear models. In *Proceedings of the 22nd Annual Conference on Neural Information Processing Systems (NIPS), Vancouver, BC*, pages 1921–1928, 2008.

[219] X. Zhang and J. Chomicki. On the semantics and evaluation of top-k queries in probabilistic databases. In *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancún, Mexico*, pages 556–563, 2008.

[220] Y. Zhang, X. Lin, W. Zhang, J. Wang, and Q. Lin. Effectively indexing the uncertain space. *IEEE Transactions on Knowledge and Data Engineering*, 22(9):1247–1261, 2010.

Acknowledgements

During the time I was working on this thesis I have received a lot of encouragement and support.

First of all, I would like to express my deepest gratitude to my supervisor, Prof. Dr. Hans-Peter Kriegel, who gave me the opportunity to work in his excellent group. The atmosphere has been very inspiring for my work; the scientific discourse and the discussions with my colleagues have been very valuable for teaching and research. In particular, I would like to thank Dr. Elke Achtert, Dr. Johannes Aßfalg, Tobias Emrich, Dr. Franz Graf, Gregor Jossé, Dr. Peer Kröger, Markus Mauder, Johannes Niedermayer, Dr. Eirini Ntoutsi, Dr. Marisa Petri, Dr. Matthias Renz, Klaus Arthur Schmid, Erich Schubert, Dr. Matthias Schubert, Dr. Florian Verhein, Dr. Arthur Zimek and Andreas Züfle for their great collaboration.

I would like to thank all external collaborators, in particular Prof. Nikos Mamoulis and Prof. Mario Nascimento, who partially initiated scientific exchanges to Hong Kong and Canada in which I could take part, and also Dr. Alexander Scholz, Dieter Dill and Christoph Türmer from the Sendsor GmbH for the productive collaboration and for providing the hardware sensors needed to record data for this work.

Moreover, I want to mention Susanne Grienberger and Franz Krojer, who provided as much help as they could in terms of administrative and technical support.

I further received helpful support by my numerous students – their names are listed on the following page – for their contributions to my research and for their assistance regarding my teaching duties.

Finally, I want to express my deepest gratitude to my family and my friends for their continuous mental support and encouragement during my research; special thanks go to Teresa Müller for her patience, her regular traveling and her ability to make me appreciate the little things in life.

Thomas Bernecker

München, August 24, 2012

The publications [43, 44] contained in this thesis have been supported by a grant from the Germany/Hong Kong Joint Research Scheme sponsored by the Research Grants Council of Hong Kong (Reference No. G_HK030/09) and the Germany Academic Exchange Service of Germany (Proj. ID 50149322).

The publications [32, 33, 40, 41] contained in this thesis have been supported in part by the THESEUS program in the MEDICO and CTC projects. They are funded by the German Federal Ministry of Economics and Technology under the grant number 01MQ07020.

Further support for the evaluation of several publications was provided by J. Remis and R. Santos from the Institute of Medical Psychology (IMP) at the Ludwig-Maximilians-Universität München, who made the neurospora datasets available (used in [45, 49]), as well as by U. Böllmann and M. Meindl, who provided the environmental datasets from the Bavarian State Office for Environmental Protection, Augsburg, Germany (used in [43, 45, 48, 49]).

Andrej Abramov, Harald Böhringer, Sarah Breining, Sveta Danti, Carina Demel, David Fendt, Andreas Fichtner, Christiane Gargitter, Nicolas Gembler, Uli Glogowsky, Thomas Hopf, Nina Hubig, Martin Hutter, Bernhard Khaled, Andreas Martin Kolb, Sebastian Lehrmann, Verena Link, Theo Mazilu, Robert Mißbichler, Christian Mönnig, Henri Palleis, Sascha Petrak, Stephan Picker, Benjamin Sauer, Andreas Schneider, Nepomuk Seiler, Alexander Stautner, Sebastian Steuer, Sarah Tausch, Alice Thudt, Yuliya Tsukanava, Christian Walonka, Larissa Wiesner, Fabian Daniel Winter, Guilin Yang, Stefan Zankl, Xuan Zheng, Philipp Zormeier, Irina Zueva

About the Author

Thomas Bernecker was born on December 14, 1980 in München, Germany. He finished his secondary education at the Wilhelm-Hausenstein-Gymnasium in München in 2000. In the following year, he fulfilled his alternative civilian service at the Blindeninstitutsstiftung in München.

In 2001, he began studying computer science and applied statistics at the Ludwig-Maximilians-Universität (LMU) München. During his studies, he was working at O_2/Telefónica in München as a working student. In 2007, he successfully finished his diploma thesis *Threshold-basierte Ähnlichkeitssuche auf multivariaten Messreihen* in the database group of Prof. Dr. Hans-Peter Kriegel.

Afterwards, he started in Prof. Kriegel's group as an assistant lecturer and researcher. During this time, he participated at scientific exchanges with collaborators at the University of Hong Kong and at the University of Alberta, Edmonton, Canada. Furthermore, he was also involved in a cooperation with the Sendsor GmbH in the field of activity recognition on sensor data. His research comprised time series analysis, indexing in high-dimensional spaces and similarity processing in probabilistic databases.

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Name, Vorname

....

Ort, Datum Unterschrift Doktorand/in

Formular 3.2