PhD Dissertation

International Doctorate School in Information and Communication Technologies

DIT - University of Trento

Modeling and Querying Data Series and Data Streams with Uncertainty

Michele Dallachiesa

Advisor:

Prof. Themis Palpanas

Università degli Studi di Trento

March 2014

To my parents for their endless love and support.

Abstract

Many real applications consume data that is intrinsically uncertain and error-prone. An uncertain data series is a series whose point values are uncertain. An uncertain data stream is a data stream whose tuples are existentially uncertain and/or have an uncertain value. Typical sources of uncertainty in data series and data streams include sensor data, data synopses, privacy-preserving transformations and forecasting models. In this thesis, we focus on the following three problems: (1) the formulation and the evaluation of similarity search queries in uncertain data series; (2) the evaluation of nearest neighbor search queries in uncertain data series; (3) the adaptation of sliding windows in uncertain data stream processing to accommodate existential and value uncertainty. We demonstrate experimentally that the correlation among neighboring time-stamps in data series can be leveraged to increase the accuracy of the results. We further show that the "possible world" semantics can be used as underlying uncertainty model to formulate nearest neighbor queries that can be evaluated efficiently. Finally, we discuss the relation between existential and value uncertainty in data stream applications, and verify experimentally our proposal of uncertain sliding windows.

Keywords
[Uncertain data, Similarity, Data series, Data streams]

Acknowledgements

First, and foremost, I would like to thank my advisor Themis Palpanas for his enormous help and encouragement not only in research but also in life. His patience, dedication, intelligence and positiveness will continue to inspire me for a long time to come. I would like to thank the many new friends that I encountered in these years, including the great colleagues at the dbTrento research group. I am grateful to the members of my Ph.D. committee, Prof. Johann-Christoph Freytag and Prof. Minos Garofalakis. I have been very lucky to work with Charu Aggarwal, Gabriela Jacques da Silva, Buğra Gedik and Kun-Lung Wu at the IBM T.J. Watson Research Center, and with Prof. Ihab F. Ilyas at the Qatar Computing Research Institute. I have really enjoyed working with these bright minds. I would like to thank my parents for their unconditional support and my grandparents for being my eternal advocates. Lastly, I am grateful to my girlfriend for her patience and love.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, the database and data mining community has investigated extensively the problem of modeling and querying uncertain data [4, 33], and several probabilistic database systems have been proposed [41, 7, 32, 10, 81]. Uncertainty can occur for different reasons, including the inherent imprecision in the sensor observations, approximations introduced by summarization techniques, privacy-preserving transformations of sensitive records, applications in data integration and predictive models whose output is intrinsically uncertain.

Uncertain data poses significant challenges to data management. First, uncertainty needs to be modeled and represented. Each model is a trade-off between its ability to represent the complex underlying dependencies in uncertain data and its tractability and efficiency in a database system. Second, the formulation of traditional database queries and mining tasks must be revisited to accommodate the differences in the data representation, typically introducing probabilistic thresholds and other quality guarantees.

In this thesis, we focus on the following three problems: (1) the formulation and the evaluation of similarity range queries in uncertain data series, reviewing analytically and experimentally prior studies and introducing

a novel model based on the moving average; (2) the efficient evaluation of nearest neighbor search queries in uncertain data series, unifying and extending prior works in the field; (3) the adaptation of sliding windows in uncertain data streams to accommodate the existential uncertainty of the processed tuples by introducing the novel concept of uncertain sliding windows.

In the next section, we describe some use cases, where modeling explicitly the uncertainty of the underlying data is beneficial in real-world applications, and we then introduce the three problems that we study in this thesis summarizing our contributions.

1.1 Motivating Scenarios

The fast growing availability of sensor measurements is fueled by the raise of wearable devices and connected devices, as well as more traditional data sources such as meteorology, astronomy, computer vision and industrial monitoring. Applications in the above domains usually organize these sequential measurements into time series, i.e., sequences of data points ordered along the temporal dimension, making time series a data type of particular importance. Several studies have recently focused on the problems of processing and mining time series with incomplete, imprecise and even misleading measurements [21, 58, 85, 86, 91]. Some examples in real scenarios are listed below:

- In manufacturing plants and engineering facilities, sensor networks are being deployed to ensure efficiency, product quality and safety [58]: unexpected vibration patterns in production machines, or changes in chemical composition in industrial processes, are used to predict failures, suggesting repairs or replacements. The same is true in environmental science [45], where sensor networks are used in hydrologic and

geologic observing systems, pollution management in urban settings, and application of water and fertilizers in precision agriculture. In transportation, sensor networks are employed to monitor weather and traffic conditions, and increase driving safety [75]. However, sensor data is inherently imprecise. Measurements are sampled from error-prone equipment [21], whose accuracy depends heavily on environmental conditions such as humidity and temperature. Furthermore, errors can be introduced during the processing and the wireless transmission in sensor networks. Repeated measurements can be obtained from multiple sensors to determine an aggregate value with higher confidence.

- Personal information contributed by individuals and corporations is steadily increasing, and there is a parallel growing interest in applications that can be developed by mining these datasets, such as location-based services and social network applications. In these applications privacy is a major concern, addressed by various privacy-preserving transforms [3, 40, 73], which introduce data uncertainty. The data can still be mined and queried, but it requires a re-design of the existing methods in order to address this uncertainty.

- Forecasting models are used to predict future events such as weather conditions and market trends. Nevertheless, predictive models are also used by researchers as a convenient language to encode the complex semantics of the studied phenomena in natural sciences and other disciplines. Uncertainty is an inherent property of such models, that can be regarded as a measure of the goodness of the model itself.

While the problem of managing and processing uncertain data has been studied extensively in the traditional database literature [4, 33], the attention of researchers was only recently focused on the specific case of

uncertain time series.

In other applications, the sensor measurements arrive continuously and are organized as data streams. Streams of sensor measurements are widely used as inputs of real-time analytics monitoring and applications. The strong demand for applications that continuously monitor the occurrence of interesting events (e.g., road-tunnel management [75] and health monitoring [84]) has driven the research in data stream processing systems [2, 43, 94]. In many of these application domains, the data sources available for processing can be considered *uncertain*. Some examples in real scenarios are listed below:

- In offshore drilling operations [71], data sources can be inaccurate, and pose significant challenges to the monitoring systems. Oil companies want to avoid shutting down operations as much as possible. To detect when operations must indeed be stopped, such companies deploy monitoring systems to collect real-time sensor measurements, such as pressure, temperature, and mass transport along the well path. Streaming applications process the sensor data through prediction models, which generate alarms and warnings with an associated confidence.

- Monitoring of car trajectories via GPS tracking devices by insurance companies. When customers install such tracking devices in their cars, they share the GPS data with the insurance company in exchange for premium discounts. The company can use such data to derive car trajectories and driving habits of customers, which are then used to offer bigger discounts to safe drivers. An important metric regarding safe driving is the amount of time (or the number of consecutive samples) by which two cars are apart from each other and whether this time is below a safety limit. As shown in previous work [17], the exact location of a car in a highly urbanized area is uncertain, as GPS provides

inaccurate data in such scenarios.

In the remainder of this Chapter we introduce the open problems and summarize our contributions.

1.2 Modeling and Querying Uncertain Data Series

In this thesis we consider the problem of evaluating similarity range queries in uncertain data series. A range query returns a set of uncertain series whose distance to the query is lower than a predefined threshold with a known confidence level. We note that two uncertain series may be very similar with low probability. At the same time, they can be very different with high probability. The semantics of similarity in uncertain data involves distance and probabilistic thresholds. The uncertainty model and the similarity semantics affect significantly the result set.

Two main approaches have emerged for modeling uncertain time series. In the first, a probability density function (pdf) over the uncertain values is estimated by using some a priori knowledge [99, 95, 79]. In the second, the uncertain data distribution is summarized by repeated measurements (i.e., samples) [11]. In this study, we revisit the techniques that have been proposed under these two approaches, with the aim of determining their advantages and disadvantages. This is the first study to undertake a rigorous comparative evaluation of the techniques proposed in the literature for similarity matching of uncertain time series. The importance of such a study is underlined by two facts: first, the widespread existence of uncertain time series; and second, the observation that similarity matching serves as the basis for developing various more complex analysis and mining algorithms. Therefore, acquiring a deep understanding of the techniques proposed in this area is essential for the further development of the field of uncertain time series processing, and the applications that are built on

top of it [89, 63, 66].

1.2.1 Contributions

Our evaluation reveals the effectiveness of the techniques that have been proposed in the literature under different scenarios. In the experiments, we stress-test the different techniques both in situations for which they were designed, as well as in situations that fall outside their normal operation (e.g., unknown distributions of the uncertain values). In the latter case, we wish to establish how strong the assumptions behind the design principles of each technique are, and to what extent these techniques can produce reliable and stable results, when these assumptions no longer hold. We note that such situations do arise in practice, where it is not always possible to know the exact data characteristics of the uncertain time series. Furthermore, we describe additional similarity measures for uncertain time series, inspired by the moving average, namely Uncertain Moving Average (UMA), and Uncertain Exponential Moving Average (UEMA). Even though these similarity measures are very simple, previous studies had not considered them. However, the experimental evaluation shows that they perform better than the more sophisticated techniques that have been proposed in the literature. We observe that UMA and UEMA incorporate some of the information inherent in the sequence of points in the time series, thus, taking a step back from the independence assumption of the other techniques.

1.3 Top-k Nearest Neighbor Search in Uncertain Data Series

The problem of finding the top-k nearest neighbors is crucial in many data mining applications, including classifiers, recommendation and search en-

gines, and location-based services. A nearest neighbor search returns the uncertain series in the dataset whose distance to the query is the smallest one. A top-k nearest neighbor search returns the k closest uncertain series to the query. Similarly to range queries, the semantics depend on the adopted uncertainty model and the formulation of nearest neighbor in uncertain data.

The problem of identifying the top-k nearest neighbors has been widely studied in traditional database systems [48]. Its popularity is largely due to the simplicity of tuning the k parameter in contrast to other queries such as similarity search, where a distance threshold has to be provided. Similarly, the evaluation of top-k nearest queries has received considerable attention in probabilistic databases [26, 76, 56, 83, 16, 62, 28]. The high dimensionality and the highly correlated dimensions of uncertain data series pose significant new challenges to the efficient evaluation of top-k nearest neighbor searches in uncertain series.

1.3.1 Contributions

We introduce different formal definitions for uncertain data series discussing their different properties. We introduce a variety of algorithms based on the iterative refinement of probability bounds, incorporating and extending the proposals of prior studies. We further investigate the efficient retrieval of candidates, considering both spatial and metric pruning strategies. We evaluate our proposal under a variety of settings using 45 real datasets from diverse domains and synthetic datasets. The results show that modeling uncertainty with probabilistic models can lead to more accurate results. Our proposal proved to be up to orders of magnitude more efficient than previously proposed techniques not specifically designed for uncertain data series.

1.4 Management of Sliding Windows in Uncertain Data Streams

An uncertain data stream is a data stream whose tuples are existentially uncertain. The value assigned to each tuple is uncertain. In many applications, data streams are processed through sliding windows. A sliding window of size w identifies the set of the most w recent tuples and advances when a new tuple comes in. The semantics of sliding windows need to be revisited to accommodate the existential uncertainty of the stream tuples. In this study, we investigate the adaptation of sliding windows to uncertain data streams.

Current research in processing uncertain data streams focuses mostly on the development of specific stream operators (e.g., joins [57, 61] and aggregates [49]) and specific queries (e.g., top-k [51, 97] and clustering [5]) that can operate in the presence of value uncertainty. These works are not designed with the integration into current general-purpose stream processing engines in mind. This is because they ignore the challenges arising from operator composition (different operators are connected to form an operator graph), which is a common development paradigm when writing streaming queries [2, 46, 70]. One such challenge is to consider streams with existential uncertainty. Existential uncertainty arises when applying certain transformations to streams with value uncertainty. For example, tuples may be generated when an event is triggered. If the event is uncertain, then the new tuple may not exist in some possible world instantiation. As a result, the regular sliding windows can over-estimate the window size, not considering the possibility that some data values do not exist in the window. Processing streams with existential uncertainty has an impact on *window management*, which is one of the basic building blocks of stream processing algorithms [2, 42, 51, 61]. Windows are often used by streaming

algorithms that require access to the most recent history of a stream, such as aggregations, joins, and sorts. Windows can have different behaviors (e.g., tumbling and sliding) and configurations (e.g., size). Window sizes can be defined based on time (e.g, all tuples collected in the last x seconds) or based on a count (e.g., last x tuples). Count-based windows are especially useful for coping with the unpredictable incoming rate of data streams. By limiting the size of the windows, developers can ensure that the memory consumed by the operator can be bounded. In existentially certain streams, establishing the boundaries of a window is trivial, since every tuple processed is guaranteed to be present in the stream. However, how should one manage such windows considering that in existentially uncertain streams it is not guaranteed that a tuple is indeed present in a given window bound? We note that the characteristics of the data streams may vary over time and a constant, larger window size may lead to overestimates of the desired window size, eventually causing undesired and unexpected effects. In this study, we investigate this problem.

1.4.1 Contributions

The main contributions of this study are as follows. We demonstrate how streams with value uncertainty can lead to existential uncertainty and vice versa, after stream operator transformations. We provide a formal definition of uncertain sliding windows, which serves as a basic building block for generic stream processing operators that need to maintain recent tuples as state. We provide exact and approximate algorithms for managing existentially uncertain sliding windows and show that previous existing state-of-the-art similarity join techniques can be easily adapted to operate on uncertain sliding windows. We present an experimental evaluation on real-world data sets, and show improvement (on all 17 datasets) over a state-of-the-art approach [61] adapted to handle existential uncertainty.

1.5 Structure of the Thesis

The thesis is structured as follows. In Chapter 2, we review the state of the art and introduce the preliminaries in Chapter 3. In Chapter 4, we compare prior studies to evaluate similarity range queries in uncertain data series and propose novel models based on the moving average. In Chapter 5, we present our algorithms for the efficient evaluation of top-k nearest neighbor queries in uncertain data series. Our adaptation of sliding windows to uncertain data streams is presented in Chapter 6. We offer our conclusions in in Chapter 7.

Chapter 2

Related Work

The problem of modeling, querying and mining uncertain data has been investigated extensively in recent years [4, 33]. A comprehensive review of the models and algorithms introduced by the database community can be found in [6]. Several database systems supporting uncertain data have been proposed, such as Conquer [41], Trio [7], MistiQ [32], MayMBS [10] and Orion [81].

The "possible worlds" model formalizes uncertainty by defining the space of the possible instantiations of the database. Instantiations must be consistent with the semantics of the data. For example, in a spatio-temporal database there may be two distinct possible trajectories representing the uncertain trajectory of a moving object, but an object cannot be in two different locations at the same time. The main advantage of the "possible worlds" model is that the formulations of the queries originally designed for certain data can be directly applied on each possible instantiation. Many different alternatives have then been proposed to aggregate the results across the different instantiations.

Despite its attractiveness, the number of possible worlds explodes very quickly and even their enumeration becomes an intractable problem. To overcome these issues, simplifying assumptions have been introduced to

leverage its simplicity: The tuple- and the attribute-uncertainty models [50, 6]. In the attribute-uncertainty model, the uncertain tuple is represented by means of multiple samples drawn from its Probability Density Function (PDF). In contrast, in the tuple-uncertainty model the value of the tuple is fixed but the tuple itself may not exist with some probability.

In the context of time series databases, a series can be formalized as a point in a high-dimensional space with correlated dimensions. An uncertain series can then be represented by enumerating its possible instantiations under the "possible world" semantics. Prior works on uncertain time series [79, 95, 11] introduce the additional assumption of independence across different timestamps. Nevertheless, temporal correlation is a well known property of time series data and ignoring it may lead to erroneous results.

We proceed reviewing prior studies focusing on the evaluation of top-k nearest neighbor searches in uncertain data.

2.1 Nearest Neighbor Queries

The evaluation of Top-k nearest neighbor queries on uncertain data is a well recognized problem, that can be tracked back to the seminal work of Cheng et al. [26]. Subsequently many different formulations of "nearest neighbor" in uncertain data have been proposed. A detailed review of the state of the art can be found in [50].

In [26], the authors dissect the processing of NN (Nearest Neighbor) queries in four steps: projection, pruning, bounding and evaluation. The projection phase returns the regions bounding the object uncertainties, the pruning phase removes from the list of candidate objects with zero probability of being the NN, and finally the bounding and evaluation phases refine the probability bounds until the NN object is identified. The traits

of this four-step approach can be found in nearly all the subsequent studies tackling the same problems.

In [76] Re et al. proposed the Multi-Simulation (MS) algorithm to discern the Top-K most probable NN objects by running in parallel several Monte-Carlo simulations. The objects that cannot be safely added to the result set or that cannot be discarded are identified by using the notion of critical region. The critical region is a region in the probability space. Each object is represented by its probability interval of being the NN. The objects having their probability intervals overlapping with the critical region are selected for another simulation step until convergence (i.e., the critical region is empty). In contrary to what considered in our study, the NN probability bounds for different objects are not correlated.

In [56] Kriegel et al. studied the efficient evaluation of probabilistic NN queries as formulated in [26] where each uncertain object is represented by a set of possible instantiations.The samples of each object are clustered using the k-means algorithm, thus obtaining a set of bounding regions that represent the identified clusters. The clusters are then indexed using an R-tree. Similarly to [76], Monte-Carlo simulations are efficiently evaluated on the constructed R-tree to determine the NN probability estimates.

In [83] Soliman et al. introduced the *U-Top-k* and *U-kRanks* queries and algorithms for their efficient evaluation on traditional DBMS. Objects are retrieved in minimum-distance order relying on the underlying DBMS capabilities and candidate result sets are then determined. The search terminates when any non-retrieved object can be safely pruned.

In [16] Beskales et al. proposed a method inspired by [83] to evaluate Top-K-NN queries as formulated in [26]. The non-retrieved objects are modeled by means of a special "virtual" object that represents them all. When the "virtual" object is considered for insertion in the result set, a new real object is retrieved in minimum-distance order to be processed. The

retrieved objects are represented by spatial regions that bound their uncertain location. When the bounding regions overlap, they are partitioned to obtain a more fine-grained representation of the object uncertainties. The algorithm terminates when the "virtual" object (and thus all non-retrieved objects) can be safely pruned and the bounding regions of the objects have been sufficiently refined to discriminate their NN probabilities.

In [62] Lian et al. studied the efficient evaluation of *pRank* queries (equivalent to *U-kRanks* queries [83]). The uncertain objects are bounded to spatial regions and indexed using an R-tree. The index is then used to prune the candidate objects that have zero probability of being part of the result set. Pre-computed values of the inverse of the Cumulative Density Function (CDF) of the objects are then used to determine their Top-K-NN probability bounds. Eventually the object probabilities are estimated using exact numerical methods if the Top-K-NN probability bounds are not enough tight to discriminate the objects in the result set with enough accuracy.

In [28] Cheng et al. introduced the concept of *probabilistic verifiers* to evaluate efficiently the approximate answer to Top-K-NN queries as formulated in [26]. Similarly to prior works, spatial regions bounding the uncertain objects are indexed using an R-tree to perform spatial pruning. Histograms representing the object Probability Density Functions (PDFs) are then used to determine the NN probability bounds. If further tightening of the probability bounds is required, histogram bins are approximated to point values until convergence.

The problem of indexing spatial uncertain data has been addressed by different studies, leading to the development of the following methods: x-bounds [27], U-trees [88], U-grids [52], APLA-trees [65] and Gauss-trees [18]. We observe that the pruning power of spatial indexes is bounded by the *indexability* of the regions that represent the uncertain points. Ide-

ally a NN search on a spatial index will return only the NN point. In practice a larger number of points is returned because their index representations are not enough accurate. The ratio of pruned candidates is a measure of the dataset *indexability* and serve as a measure of the goodness of the index structure. Minimum Bounding Rectangles (MBRs) are a popular representation of bounding regions in spatial indexes. Envelopes are their natural extension to data series. An envelope is a pair of series s.t. their value at each time-stamp t represents respectively the minimum and maximum values of the represented series at time-stamp t. Index structures and multi-resolution representations of data series that rely on envelope synopses include iSAX [20] and Haar wavelet transforms [22]. Recently, a new line of research focused on indexing uncertain objects in metric spaces [9]. In [9] Angiulli et al. introduced the *UP-Index* to support the efficient evaluation of range queries by maintaining statistics on the distance distribution between the indexed objects and a set of pivot objects. These statistics are then used at query time to prune the search space using probability bounds based on the reverse triangular inequality. The algorithm cannot be easily adapted to evaluate Top-K-NN queries since it doesn't model the relationships between the indexed objects. At best of our knowledge, no prior works considered the problem of evaluating Top-K-NN queries for uncertain data in metric spaces.

2.2 Uncertain Data Streams

In the last decade, several database and stream processing systems with support for uncertainty have been proposed [12, 36, 53, 82, 31, 51, 90, 91], eventually leading to two emerging tuple models.

The *x-tuple* model [12] represents uncertain tuples by multiple alternatives and their respective occurring probabilities. If the sample prob-

abilities do not sum up to one, there exists possible instantiations of the uncertain stream where the tuple does not exist. Uncertain tuples are processed according to the possible worlds semantics [44].

In the *attribute* model [36, 82], uncertainty is more fine-grained and it refers to single tuple attributes. An uncertain attribute is represented by a random variable s.t. its distribution is assumed to be known. The distribution may be continuous or discrete, and it is fully described by its Probability Density Function (PDF). The baseline formalization of this model fails to capture correlations among attributes. Extensions have been proposed to address this limitation [82].

In this study we adopt the *x-tuple* model. This choice is motivated by the following observations. First, it can capture correlations among attributes without considering more complex extensions (i.e., making explicit the tuple distribution by means of a set of drawn samples). Second, it supports both value uncertainty and existential uncertainty of tuples. Third, real-world uncertain data is often provided by means of discrete samples drawn from unknown distributions. Fourth, possible worlds semantics provide an intuitive bridge between semantics of stream operators in certain data streams and their respective adaptations for uncertain data streams. Last but not the least, we observe that applying stream operators to uncertain streams can lead to complex distributions that do not have a closed form. This requires capturing data stream dynamics by reasoning on complex distributions, relying on methods like Monte Carlo estimation, which usually cannot be performed efficiently.

In what follows, we give an overview of relevant work in the literature on processing data streams with uncertainty, adopting the uncertainty models described above.

Lian and Chen [61] propose novel techniques for answering similarity matching queries between uncertain data streams. Methods for spatial and

probabilistic pruning are used to filter the search space efficiently. The two data streams are processed through a pair of sliding windows, and candidate matches are identified by the sliding window contents. This study is orthogonal to our proposal, and it is used to evaluate the effectiveness of our techniques.

Diao et al. [36] propose a data stream processing system that supports uncertainty modeled by continuous random variables. It also contributes two real-world use cases, namely object tracking on RFID networks and monitoring of hazardous weather conditions.

Ré et al. [77] propose an event processing system for probabilistic event streams by using Markovian models to infer hidden (possibly correlated) variables, e.g., a person's location from RFID readings. It is worth noting that this system can produce output events that are existentially uncertain.

Dallachiesa et al. [31] perform an extensive experimental and analytical comparison of methods for answering similarity matching queries on uncertain time series.

In [30], an augmented R-tree indexes a dataset of spatial points with existential uncertainty. The authors represent existential uncertainty by independent probability values associated to the indexed points. Intermediate nodes maintain aggregate statistics, summarizing the existential probabilities of the indexed points in their subtrees. Augmented R-trees support probabilistic range queries, reporting only matching points with existential probabilities higher than a user-defined threshold.

In [51], the authors propose a general framework to answer top-k queries on uncertain data streams. Each item in the data stream exists with some independent probability. Given a user-defined sliding window size, possible worlds are enumerated and the top-k items are identified accordingly to different possible semantics supported by the model. The window size is fixed, and it is used to enumerate all possible worlds.

In [60], the authors consider the problem of identifying frequent itemsets in uncertain data streams. Uncertain data streams are processed through a sliding window containing a fixed number of batches (each batch contains a fixed number of transactions). The existential probability of each transaction is represented by an independent probability value. Also in this study, the window size is fixed and it does not change over time.

Zhang et al. [98] propose an efficient method to maintain skylines over uncertain data streams. A skyline is a set of items s.t. they are not dominated by any other item. An item i dominates item j if it is "better" than j in at least one tuple attribute and not "worse" than j in all the other tuple attributes. The definitions of "better" and "worse" are domain-specific. The skyline is maintained over a sliding window. The window size is fixed. The probability for each item to belong to the skyline is then estimated by enumerating all the possible worlds. Only skyline items with probability higher than a user-defined thresholds are reported.

In the aforementioned papers, the occurrence probabilities of items in a data stream do not affect the sliding window size. The window size is fixed and does not depend on data uncertainty. In our study, we extend the semantics of sliding window query processing by referring to the window size as the number of truly existing tuples in the uncertain data stream. Our contribution is a basic building block for processing sliding windows on uncertain data streams, and it is orthogonal to past studies. As shown in Section 6.4, previous works on streaming operations with sliding windows can be easily adapted to accommodate our extensions.

In this work, we take advantage of previously developed methods for efficiently evaluating the CDF of Poisson-binomial distributions, e.g. the sum of n independent Bernoulli trials. Some methods include Bernecker et al. [15], which proposes an algorithm with time cost $O(n^2)$ based on dynamic programming, and Sun et al. [87], which proposes an algorithm

based on divide-and-conquer with time cost $O(nlog^2n)$. Other approximation algorithms also exist [92, 19, 35]. We use one exact method (RF1) and three approximations (Poisson, Normal, and Refined Normal Approximations), as reviewed in [47]. Independence is a simplifying assumption widely used in prior studies on uncertain data management [4].

Chapter 3

Preliminaries

In this section, we overview the definitions of uncertain data series and uncertain data stream.

A data series[1] S is an ordered sequence of n real valued numbers $S = S[t]$, $1 \leq t \leq n$. For ease of exposition, we refer to the ith point of series S also as s_i. Where not specified otherwise, we assume normalized time series with zero mean and unit variance. Notice that normalization is a preprocessing step that requires particular care to address specific situations [64]. An uncertain data series X is a data series whose values at each time-stamp are uncertain. We adopt the attribute-uncertainty model under the "possible world" semantics to represent uncertain series. Under the attribute-uncertainty model, the series always exists but its value is uncertain. The value uncertainty along the series is represented by means of repeated instantiations, i.e., samples.

An instantiation can be represented by a real-valued sample drawn independently from the value distribution at every time-stamp or by a series sample drawn from the full-joint distribution of the uncertain series. The properties and the implications of these two alternative models are discussed in Chapter 5.

A data stream S is a sequence of tuples s_i, where $0 \leq i \leq \eta$ and $\eta \in \mathbb{N}$,

[1] In the rest of this thesis we use the terms *time series*, *data series*, *series* and *sequence*, interchangeably.

where η is the index of the most recent tuple received from stream S. We refer to i as the index of a tuple in a stream. Without loss of generality, a tuple s_i is a $d-$dimensional real-valued point [2]. We define a subsequence of stream S as $S_{[i,j]} = \langle s_i, \ldots, s_j \rangle$. We define a count-based sliding window $W(S, w)$ as the subsequence $S_{[\eta-w+1,\eta]}$, where $w \in \mathbb{N}$ indicates the size of the window. When not implicit from the context, we refer to data streams without uncertainty as *certain* data streams.

An uncertain data stream U is a sequence of uncertain tuples u_i, where $0 \leq i \leq \eta$ and $\eta \in \mathbb{N}$. Tuple u_i is represented by a set of l possible materializations, i.e., $u_i = \{u_{i,1}, \ldots, u_{i,l}\}$. If $|u_i| > 1$, then the tuple has value uncertainty. A sample materialization $u_{i,j} \in u_i$ occurs with a given probability $Pr(u_{i,j})$. The existential probability $Pr(u_i)$ of tuple u_i is defined as

$$Pr(u_i) = \sum_{u_{i,j} \in u_i} Pr(u_{i,j}). \tag{3.1}$$

Tuple u_i is said to exist in stream U if $Pr(u_i) = 1$. If $Pr(u_i,) < 1$, tuple u_i is considered existentially uncertain. As demonstrated in Chapter 6, applying commonly used stream transformations to uncertain data streams can (i) introduce existential uncertainty from value uncertainty, and (ii) introduce value uncertainty from existential uncertainty.

In the next Chapter we will review and compare experimentally the different models that have been proposed to evaluate similarity range queries in uncertain data series.

[2] Each dimension can be considered as an attribute.

Chapter 4

Uncertain Time-Series Similarity: Return to the Basics

In the last years there has been a considerable increase in the availability of continuous sensor measurements in a wide range of application domains, such as Location-Based Services (LBS), medical monitoring systems, manufacturing plants and engineering facilities to ensure efficiency, product quality and safety, hydrologic and geologic observing systems, pollution management, and others.

Due to the inherent imprecision of sensor observations, many investigations have recently turned into querying, mining and storing uncertain data. Uncertainty can also be due to data aggregation, privacy-preserving transforms, and error-prone mining algorithms.

In this study, we survey the techniques that have been proposed specifically for modeling and processing uncertain time series, an important model for temporal data. We provide an analytical evaluation of the alternatives that have been proposed in the literature, highlighting the advantages and disadvantages of each approach, and further compare these alternatives with two additional techniques that were carefully studied before. We conduct an extensive experimental evaluation with 17 real datasets, and discuss some surprising results, which suggest that a fruitful research

direction is to take into account the temporal correlations in the time series. Based on our evaluations, we also provide guidelines useful for the practitioners in the field.

The rest of this chapter is structured as follows. In Section 4.1 we survey the principal representations and distance measures proposed for similarity matching of uncertain time series. In Section 4.2, we analytically compare the methods proposed for uncertain time series modeling, and in Section 4.3, we present the experimental comparison. We describe new measures of similarity matching inspired by the moving average in Section 4.4, and evaluate their performance in relation to the other measures. Finally, in Section 4.5 we summarize the results, and Section 4.6 concludes the study.

4.1 Similarity Matching for Uncertain Time Series

Recall that time series are sequences of points, typically real valued numbers, ordered along the temporal dimension. We assume constant sampling rates and discrete timestamps.

In this study, we focus on uncertain time series where uncertainty is localized and limited to the points. Formally, an uncertain time series T is defined as a sequence of random variables $< t_1, t_2, ..., t_n >$ where t_i is the random variable modeling the real valued number at time-stamp i. All the three models we review and compare fit under this general definition.

The problem of similarity matching has been extensively studied in the past [8, 38, 78, 54, 23, 69, 68, 64] : given a user-supplied query sequence, a similarity search returns the most similar time series according to some distance function. More formally, given a collection of time series $C = \{S_1, ..., S_N\}$, where N is the number of time series, we are interested in evaluation the range query function $RQ(Q, C, \epsilon)$:

$$RQ(Q, C, \epsilon) = \{S : S \in C \wedge distance(Q, S) \leq \epsilon\} \quad (4.1)$$

In the above equation, ϵ is a user-supplied distance threshold. A survey of representation and distance measures for time series can be found in [37].

A similar problem arises also in the case of uncertain time series, and the problem of probabilistic similarity matching has been introduced in the last years. Formally, given a collection of uncertain time series $C = \{T_1, ..., T_N\}$, we are interested in evaluation the probabilistic range query function $PRQ(Q, C, \epsilon, \tau)$:

$$PRQ(Q, C, \epsilon, \tau) = \{T : T \in C \wedge Pr(distance(Q, T) \leq \epsilon) \geq \tau\} \quad (4.2)$$

In the above equation, ϵ and τ are the user-supplied distance threshold and the probabilistic threshold, respectively.

In the recent years three techniques have been proposed to evaluate PRQ queries, namely MUNICH[1] [11], PROUD [95], and DUST [79]. As we discuss below, these methods assume that neighboring points of the time series are independent, i.e., the point at timestamp i is independent from the point at timestamp $i + 1$. Evidently, this is a simplifying assumption, since in real-world datasets neighboring points are correlated. We revisit this issue in the following sections.

We now discuss each one of the above three techniques in more detail.

4.1.1 MUNICH

In [11], uncertainty is modeled by means of repeated observations at each timestamp, as depicted in Figure 4.2.

[1]We will refer to this method as *MUNICH* (it was not explicitly named in the original paper), since all the authors were affiliated with the University of Munich.

Assuming two uncertain time series, X and Y, MUNICH proceeds as follows. First, the two uncertain sequences X, Y are materialized to all possible certain sequences: $TS_X = \{< v_{11}, ..., v_{n1} >, ..., < v_{1s}, ..., v_{ns} >\}$ (where v_{ij} is the j-th observation in timestamp i), and similarly for Y with TS_Y. Thus, we have now defined TS_X, TS_Y. The set of all possible distances between X and Y is then defined as follows:

$$dists(X, Y) = \{L^p(x, y) | x \in TS_X, y \in TS_Y\} \tag{4.3}$$

The uncertain L^p distance is formulated by means of counting the feasible distances:

$$Pr(distance(X, Y) \leq \epsilon) = \frac{|\{d \in dists(X, Y) | d \leq \epsilon\}|}{|dists(X, Y)|} \tag{4.4}$$

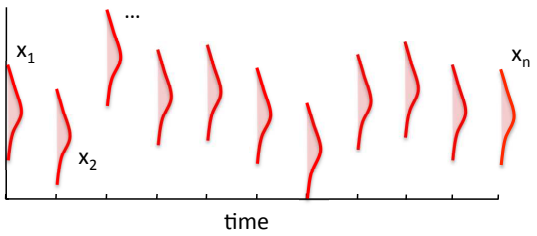


Figure 4.1: Example of uncertain time series $X = \{x_1, ..., x_n\}$ modeled by means of *pdf* estimation.

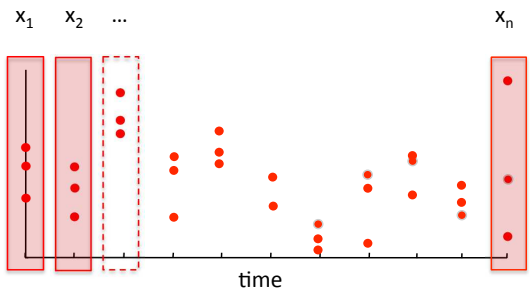


Figure 4.2: Example of uncertain time series $X = \{x_1, ..., x_n\}$ modeled by means of repeated observations.

Once we compute this probability, we can determine the result set of PRQs similarity queries by filtering all uncertain sequences using Equation 4.4.

Note that the naive computation of the result set is unfeasible, because of the very large space that leads to an exponential computational cost: $|dists(X,Y)| = s_X^n s_X^n$, where s_X, s_Y are the number of samples at each timestamp of X, Y, respectively, and n is the length of the sequences. Efficiency can be ensured by upper and lower bounding the distances, and summarizing the repeated samples using minimal bounding intervals [11]. This framework has been applied to Euclidean and Dynamic Time Warping (DTW) [13] distances and guarantees no false dismissals in the original space [11].

4.1.2 PROUD

In [95], an approach for processing queries over PRObabilistic Uncertain Data streams (PROUD) is presented. Inspired by the Euclidean distance, the PROUD distance is modeled as the sum of the differences of the streaming time series random variables, where each random variable represents the uncertainty of the value in the corresponding timestamp. This model is illustrated in Figure 4.1.

Given two uncertain time series X, Y, their distance is defined as:

$$distance(X,Y) = \sum_i {D_i}^2 \tag{4.5}$$

where $D_i = (x_i - y_i)$ are random variables, as shown in Figure 4.3.

According to the central limit theorem, we have that the cumulative distribution of the distances approaches a normal distribution:

$$distance(X,Y)_{norm} = \frac{distance(X,Y) - \sum_i E[d_i^2]}{\sqrt{\sum_i Var[D_i^2]}} \tag{4.6}$$

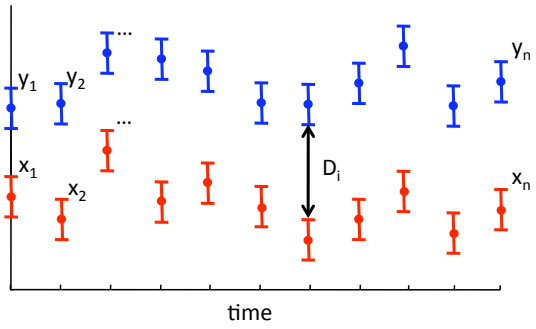


Figure 4.3: The probabilistic distance model.

The normalized distance follows a standard normal distribution, thus we can obtain the normal distribution of the original distance as follows:

$$distance(X,Y) \propto N(\sum_i E[D_i^2], \sum_i Var[D_i^2]) \tag{4.7}$$

The interesting result here is that, regardless of the data distribution of the random variables composing the uncertain time series, the cumulative distribution of their distances (1) is defined similarly to their Euclidean distance and (2) approaches a normal distribution. Recall that we want to answer PRQs similarity queries. First, given a probability threshold τ and the cumulative distribution function (*cdf*) of the normal distribution, we compute ϵ_{limit} such that:

$$Pr(distance(X,Y)_{norm} \leq \epsilon_{limit}) \geq \tau \tag{4.8}$$

The *cdf* of the normal distribution can be formulated in terms of the well-known *error-function*, and ϵ_{limit} can be determined by looking up the statistics tables. Once we determined ϵ_{limit}, we proceed by computing also the normalized ϵ:

$$\epsilon_{norm}(X,Y) = \frac{\epsilon^2 - E[distance(X,Y)]}{\sqrt{Var[distance(X,Y)]}} \tag{4.9}$$

If a candidate uncertain series Y satisfies the inequality:

$$\epsilon_{norm}(X, Y) \geq \epsilon_{limit} \tag{4.10}$$

then the following equation holds:

$$Pr(distance(X, Y)_{norm} \leq \epsilon_{norm}(X, Y)) \geq \tau \tag{4.11}$$

Therefore, Y can be added to the result set. Otherwise, it is pruned away. This distance formulation is statistically sound and only requires knowledge of the general characteristics of the data distribution, namely, its mean and variance.

4.1.3 DUST

In [79], the authors propose a new distance measure, DUST. In contrast to MUNICH, it does not depend on the existence of multiple observations and is computationally more efficient. Similarly to [95], DUST is inspired by the Euclidean distance, but works under the assumption that all the time series values follow some specific distribution.

Given two uncertain time series X, Y, the distance between two uncertain values x_i, y_i is defined as the distance (L_1 norm) between their true (unknown) values $r(x_i), r(y_i)$: $dist(x_i, y_i) = L_1(r(x_i), r(y_i))$. This distance can then be used to define a function ϕ that measures the similarity of two uncertain values:

$$\phi(|x_i - y_i|) = Pr(dist(|r(x_i) - r(y_i)|) = 0) \tag{4.12}$$

This basic similarity function is then used inside the *dust* dissimilarity function:

$$\begin{aligned} dust(x,y) &= \sqrt{-\log(\phi(|x-y|)) - k} \\ &\text{with} \\ k &= -\log(\phi(0)) \end{aligned}$$

The constant k has been introduced to support reflexivity. Once we define the *dust* distance between uncertain values, we are ready to extend it to the entire sequences:

$$DUST(X,Y) = \sqrt{\sum_i dust(x_i, y_i)^2} \tag{4.13}$$

The handling of uncertainty is isolated inside the ϕ function, and its evaluation requires to know exactly the data distribution. In contrast to the techniques we reviewed earlier, the DUST distance is a real number that measures the dissimilarity between uncertain time series. Thus, it can be used in all mining techniques for certain time series, by simply substituting the existing distance function.

Finally, we note that DUST is equivalent to the Euclidean distance, in the case where the error of the time series values follows the normal distribution.

4.2 Analytical Comparison

In this section, we compare the three models of similarity matching for uncertain time series, namely, MUNICH, PROUD and DUST, along the following dimensions: uncertainty models used and assumptions made by the algorithms; type of distance measures; and type of similarity queries.

4.2.1 Uncertainty Models and Assumptions

All three reviewed techniques are based on the assumption that the values of the time series are independent from one another. That is, the value at each timestamp is assumed to be independently drawn from a given distribution. Evidently, this is a simplifying assumption, since neighboring values in time series usually have a strong temporal correlation.

The main difference between MUNICH and the other two techniques is that MUNICH represents the uncertainty of the time series values by recording multiple observations for each timestamp. This can be thought of as sampling from the distribution of the value errors. In contrast, PROUD and DUST consider each value of time series to be a continuous random variable following a certain probability distribution.

The amount of preliminary information, i.e. a priori knowledge of the characteristics of the time series values and their errors, varies greatly among the techniques. MUNICH does not need to know the distribution of the time series values, or the distribution of the value errors. It simply operates on the observations available at each timestamp.

On the other hand, PROUD and DUST need to know the distribution of the error at each value of the data stream. In particular, PROUD requires to know the standard deviation of the uncertainty error, and a single observed value for each timestamp. PROUD assumes that the standard deviation of the uncertainty error remains constant across all timestamps.

DUST uses the largest amount of information among the three techniques. It takes as input a single observed value of the time series for each timestamp, just like PROUD. In addition, DUST needs to know the distribution of the uncertainty error at each time stamp, as well as the distribution of the values of the time series. This means that, in contrast to PROUD, DUST can take into account mixed distributions for the un-

certainty errors (albeit, they have to be explicitly provided in the input).

Overall, we observe that the three techniques make different initial assumptions about the amount of information available for the uncertain time series, and have different input requirements. Consequently, when deciding which technique to use, users should take into account the information available on the uncertainty of the time series to be processed.

4.2.2 Type of Distance Measures

All the considered techniques use some variation of the Euclidean distance. MUNICH and PROUD use this distance in a pretty straightforward manner. Moreover, MUNICH and DUST can be employed to compute the Dynamic Time Warping distance [80], which is a more flexible distance measure.

DUST is a new type of distance measure that is specifically designed for uncertain time series. In other words, DUST is not a similarity matching technique per se, but rather a new distance measure. It has been shown that DUST is proportional to the Euclidean distance in the cases where the value errors are normally distributed [79]. Moreover, the authors of [79] note that it is better to use the Euclidean distance if all the value errors follow the same distribution. DUST becomes useful when the value errors are modeled by multiple error distributions.

4.2.3 Type of Similarity Queries

MUNICH and PROUD are designed for answering probabilistic range queries (defined in Section 4.1). DUST being a distance measure can be used to answer top-k nearest neighbor queries.

MUNICH and PROUD solve the similarity matching problem that is described by Equation 4.8, resulting in a set of time series that belong

to the answer with probability τ. On the other hand, DUST produces a single value that is an exact (i.e., not probabilistic) distance between two uncertain time series.

In Section 4.3, we describe the methodology we used in order to compare all three techniques using the same task, that of similarity matching.

4.3 Comparative Study

In this section, we present the experimental evaluation of the three techniques. We first describe the methodology and datasets used, and then discuss the results of the experiments.

All techniques were implemented in C++, and the experiments were run on a PC with a 2.13GHz CPU and 4GB of RAM.

The source code for all the algorithms used in our experiments, as well as the datasets upon which we tested them are publicly available[1].

4.3.1 Experimental Setup

Datasets

Similarly to [11, 95, 79], we used existing time series datasets with exact values as the ground truth, and subsequently introduced uncertainty through perturbation. Perturbation models errors in measurements, and in our experiments we consider *uniform*, *normal* and *exponential* error distributions with zero mean and varying standard deviation within interval $[0.2, 2.0]$.

We considered 17 real datasets from the UCR classification datasets collection [1], representing a wide range of application domains: 50words, Adiac, Beef, CBF, Coffee, ECG200, FISH, FaceAll, FaceFour, Gun_Point, Lighting2, Lighting7, OSULeaf, OliveOil, SwedishLeaf, Trace, and synthetic_control. The training and testing sets were joined together, and we

obtained on average 502 time series of length 290 per dataset. We stress the fact that each dataset contains several time series instances.

Since DUST requires to know the distribution of values of the time series, and additionally makes the assumption that this distribution is uniform [79], we tested the datasets to check if this assumption holds. According to the Chi-square test, the hypothesis that the datasets follow the uniform distribution was rejected (for all datasets) with confidence level $\alpha = 0.01$. Evidently, the above assumption does not hold on all datasets, however DUST still needs it in order to operate.

Comparison Methodology

In our evaluation, we consider all three techniques, namely, MUNICH, PROUD, and DUST, and we additionally compare to Euclidean distance. When using Euclidean distance, we do not take into account the distributions of the values and their errors: we just use a single value for every timestamp, and compute the traditional Euclidean distance based on these values.

The goal of our evaluation is to compare the performance of the different techniques on the same task. Observe that we cannot use the top-k search task for this comparison. The reason is that the MUNICH and PROUD techniques have a notion of probability (Equation 4.2). This means that these techniques can produce different rankings when the threshold ε changes. For example, assume that we increase ε (maintaining τ fixed). Then the ordering of the time series in a top-k ranking may change, since the probability that the time series are similar within distance $\varepsilon_1 \geq \varepsilon$ may increase. Thus, in the case of uncertain time series, MUNICH and PROUD might produce very different top-k answers even if ε varies a little. This, in turn, means that the top-k task is not suitable for comparing the three techniques.

We instead perform the comparison using the task of time series similarity matching. Even though DUST is not a similarity matching technique (like PROUD and MUNICH), it can still be used to find similar time series, when we specify a maximum threshold on the distance between time series. In [79], the evaluation of DUST was based on top-k similar time series. However, we note that this problem includes the problem of similarity matching [37], where the most similar time series form the answer to the top-k query.

Following the above discussion, in order to perform a fair comparison we need to specify distance thresholds for all three techniques. This translates to finding equivalent thresholds ε for each one of the techniques. We proceed as follows.

Since the distances in MUNICH and PROUD are based on the Euclidean distance, we will use the same threshold for both methods, ε_{eucl}. Then, we calculate an equivalent threshold for DUST, ε_{dust}. Given a query q and a dataset C, we identify the 10th nearest neighbor of q in C. Let that be time series c. We define ε_{eucl} as the Euclidean distance on the observations between q and c and ε_{dust} as the DUST distance between q and c. This procedure is repeated for every query q.

The quality of results of the different techniques is evaluated by comparing the query results to the ground truth. We performed experiments for each dataset separately, using each one of the time series as a query and performing a similarity search. In the graphs, we report the averages of all these results, as well as the 95% confidence intervals[2].

[2]Please note that the results we report are not directly comparable to those in the original papers. In our study, we use a different experimental setup, in order to make possible the comparison of the three techniques.

4.3.2 Quality Performance

In order to evaluate the quality of the results, we used the two standard measures of *recall* and *precision*. Recall is defined as the percentage of the truly similar uncertain time series that are found by the algorithm. Precision is the percentage of similar uncertain time series identified by the algorithm, which are truly similar. Accuracy is measured in terms of F_1 score to facilitate the comparison. The F_1 score is defined by combining precision and recall:

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{4.14}$$

We verify the results with the exact answer using the ground truth, and compare the results with the algorithm output (as described in Section 4.3.1).

Accuracy

The first experiment represents a special case with restricted settings. This was necessary to do, because the computational cost of MUNICH was prohibitive for a full scale experiment. We compare MUNICH, PROUD, DUST and Euclidean on the Gun_Point dataset, truncating it to 60 time series of length 6. For each timestamp, we have 5 samples as input for MUNICH. Results are averaged on 5 random queries. For both MUNICH and PROUD we are using the optimal probabilistic threshold, τ, determined after repeated experiments. Distance thresholds are chosen (according to Section 4.3.1) such that in the ground truth set they return exactly 10 time series.

The results with Gaussian error (refer to Figure 4.4(a)) show that all techniques perform well (F_1 >80%) when the standard deviation of the errors is low ($\sigma = 0.2$), with MUNICH being the best performer (F_1=88%).

However, as the standard deviation increases to 2, the accuracy of all techniques decreases. This is expected, since a larger standard deviation means that the time series have more uncertainty. The behavior of MUNICH though, is interesting: its accuracy falls sharply for $\sigma > 0.6$.

This trend was verified also with uniform and exponential error distributions, as reported in Figures 4.4(b) and 4.4(c). With exponential error, the performance of MUNICH is slightly better than with normal, or uniform error distributions. However, MUNICH still performs much worse than PROUD and DUST for $\sigma > 0.6$.

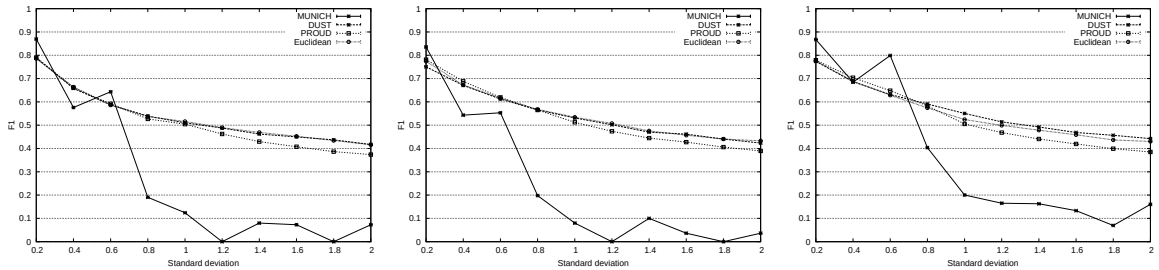
Figure 4.5(a) shows the results of the same experiment, but just for PROUD, DUST, and Euclidean. In this case (and for all the following experiments), we report the average results over the full time series for all datasets. Once again, the error distribution is normal, and PROUD is using the optimal threshold, τ, for every value of the standard deviation.

The results show that there is virtually no difference among the different techniques. This observation holds across the entire range of standard deviations that we tried ($0.2 \leq \sigma \leq 2$).

The results for the uniform and exponential distributions are very similar, and reported in Figures 4.5(b) and 4.5(c). With both uniform and exponential errors, PROUD performs slightly better for $\sigma = 0.2$, and its performance drops slightly below DUST and Euclidean for larger error standard deviations.

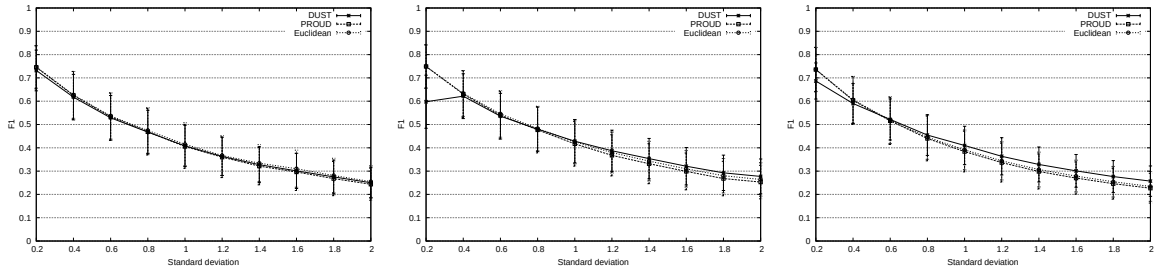
With uniform error, the accuracy of DUST drops by nearly 10% for $\sigma = 0.2$ (refer to Figures 4.5(b)). This apparently insignificant observation turned out to be due to how the DUST lookup tables are determined: When the error is uniformly distributed, $\phi(|x_i - y_i|)$ may be equal to zero in some cases. Consequently, $dust(x, y)$ cannot be evaluated for these cases, as it degenerates to the logarithm of zero. We tried to solve this technical problem by adding two tails to the uniform error, so that the error

probability density function is never exactly zero. This workaround proved useful, but did not completely solve the problem.



(a) normal error distribution (b) uniform error distribution (c) exponential error distribution

Figure 4.4: F_1 score for MUNICH, PROUD, DUST and Euclidean on Gun_Point truncated dataset, when varying the error standard deviation: normal error distribution (left), uniform (center), exponential (right).



(a) normal error distribution (b) uniform error distribution (c) exponential error distribution

Figure 4.5: F_1 score for PROUD, DUST and Euclidean, averaged over all datasets, when varying the error standard deviation: normal error distribution (left), uniform (center), exponential (right).

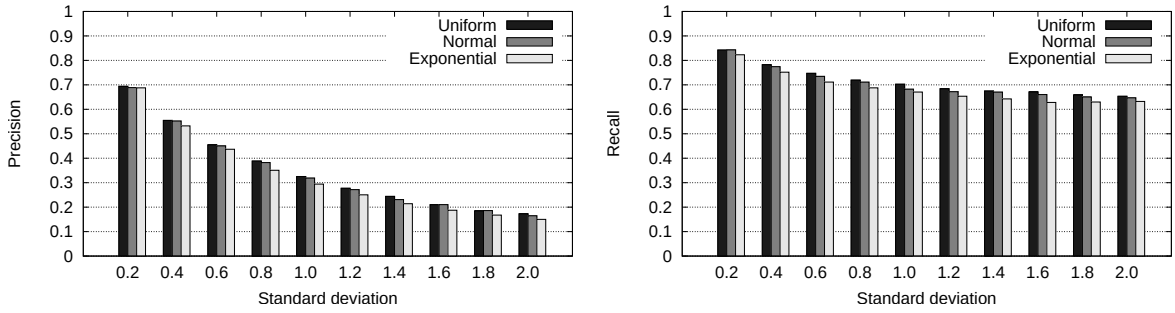
Precision and Recall

In order to better understand the behavior of the different techniques, we take a closer look at precision and recall. Figures 4.6(a) and 4.6(b) show the precision and recall for PROUD, as a function of the error standard deviation, when the distribution of the error follows a uniform, a normal,

and an exponential distribution. PROUD is using the optimal threshold, τ, for *every* value of the standard deviation.

The graphs show that recall always remains relatively high (between $63\% - 83\%$). On the contrary, precision is heavily affected, decreasing from 70% to a mere 16% as standard deviation increases from 0.2 to 2. Therefore, processing uncertain time series with an increasing standard deviation in their error does not have a significant impact on the false positives. However, this leads to many false negatives, which may be an undesirable effect.

The corresponding results for DUST are shown in Figures 4.7(a) and 4.7(b). We observe the same trends as before, the only difference being that DUST achieves slightly better precision, but lower recall.

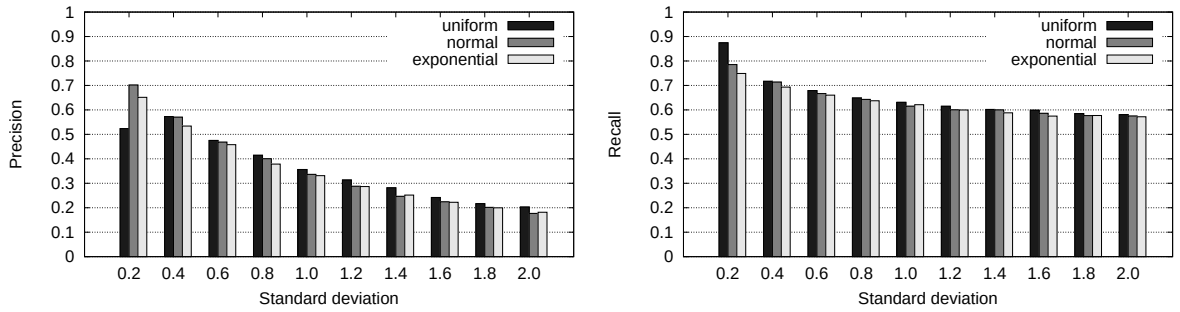


(a) (b)

Figure 4.6: Precision and recall for PROUD, averaged over all datasets, when varying error standard deviation and error distribution.

Mixed Error Distributions

While in all previous experiments the error distribution is constant across all the values of a time series, in this experiment we evaluate the accuracy of PROUD, DUST, and Euclidean when we have different error distributions present in the same time series (Figure 4.8). Each time series has been



(a) (b)

Figure 4.7: Precision and recall for DUST, averaged over all datasets, when varying error standard deviation and error distribution.

perturbed with normal error, but of varying standard deviation. Namely, the error for 20% of the values has standard deviation 1, and the rest 80% has standard deviation 0.4.

We note that this is a case that PROUD cannot handle, since it does not have the ability to model different error distributions within the same time series (in this experiment, PROUD was using a standard deviation setting of 0.7). Therefore, PROUD does not produce better results than Euclidean. On the other hand, DUST is taking into account these variations of the error, and achieves a slightly improved accuracy (3% more than PROUD and Euclidean).

We also conducted the same experiment by changing the following settings: (i) inform DUST (wrongly) that the standard deviation is 0.7, and (ii) perturb a time series with a mixture of uniform, normal, and exponential distributions (this situation cannot be handled by PROUD).

As shown in Figures 4.9 and 4.10, in both these experiments the accuracy of all techniques (PROUD, DUST, and Euclidean) is almost the same, and consistently lower for the second experiment. These results indicate that in situations where we do not have enough, or accurate information on the distribution of the error, PROUD and DUST do not offer an advantage

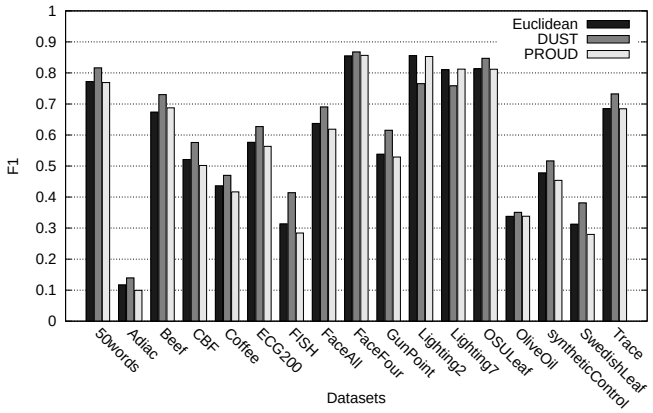


Figure 4.8: F_1 score for PROUD, DUST, and Euclidean on all the datasets with mixed error distribution (normal), 20% with standard deviation 1.0, and 80% with standard deviation 0.4.

when compared to Euclidean.

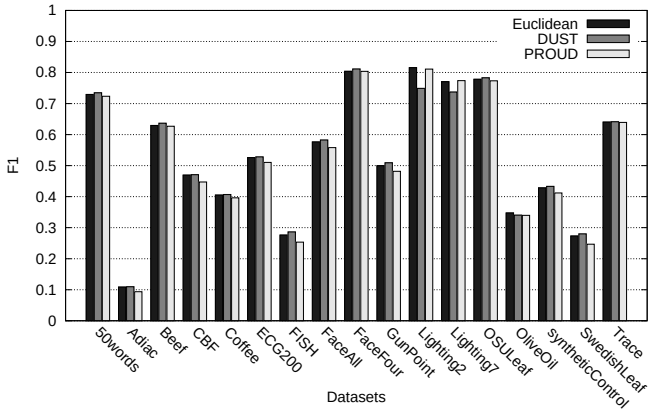


Figure 4.9: F_1 score for PROUD, DUST, and Euclidean on all the datasets with mixed error distribution (uniform, normal, and exponential), 20% with standard deviation 1.0, and 80% with standard deviation 0.4.

4.3.3 Time Performance

In Figure 4.11, we report the CPU time per query for the normal error distribution when varying the error standard deviation in the range $[0.2, 2.0]$. The results for uniform and exponential distributions are very similar, and omitted for brevity.

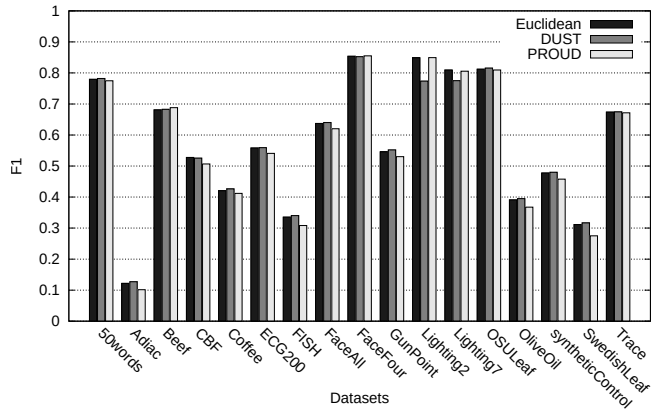


Figure 4.10: F_1 score for PROUD, DUST, and Euclidean on all the datasets with mixed error distribution: normal, with standard deviation erroneously reported as constant 0.7.

The graph shows that the standard deviation of the normal distribution only slightly affects performance for DUST. As expected, the execution time for Euclidean is not affected at all when the standard deviation for the error of the uncertain time series varies, and exhibits the best time performance of all techniques.

We note that for PROUD we did not use the wavelet synopsis, since we did not use any summarization technique for the other techniques either. However, it is possible to apply PROUD on top of a Haar wavelet synopsis. This results in CPU time for PROUD that is equal or less to the CPU time of Euclidean, while maintaining high accuracy [95].

We did not include the time performance for MUNICH in this graph, because it is orders of magnitude more expensive than the other techniques (i.e., in the order of minutes).

In Figure 4.12, we report the CPU time per query for the normal error distribution when varying the time series length between 50 and 1000 time points. Time series of different lengths have been obtained resampling the raw sequences. The graph shows that the time grows linearly to the time series length. The results for uniform and exponential distributions are very similar, and omitted for brevity.

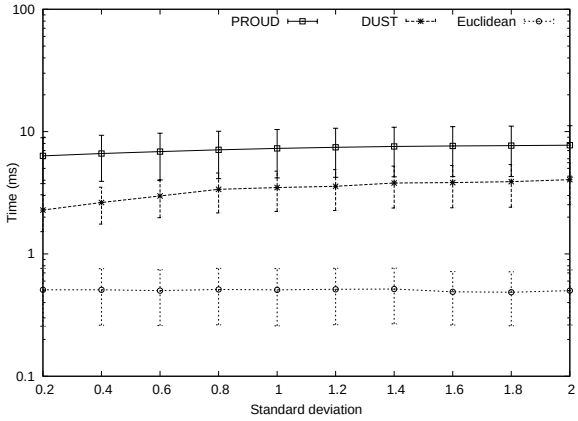


Figure 4.11: Average time per query for PROUD, DUST, and Euclidean, averaged over all datasets, when varying the error standard deviation with normal error distribution.

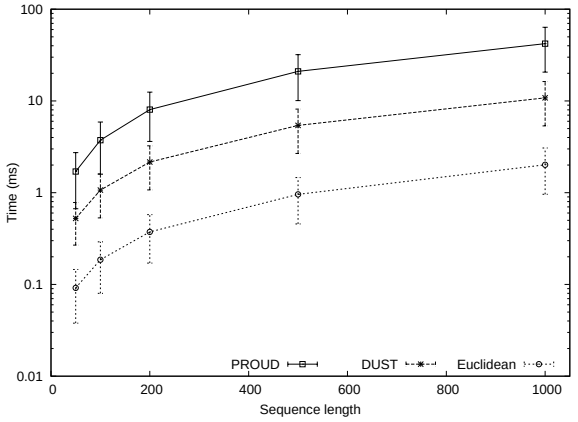


Figure 4.12: Average time per query for PROUD, DUST, and Euclidean, averaged over all datasets, when varying the time series length with normal error distribution.

4.4 Moving Average for Uncertain Time Series

The moving average is among the simplest filters for noise reduction in signal processing. In this section, we compare some basic adaptations of the moving average to the DUST and Euclidean distances, and evaluate their effectiveness. We note that similar to the Euclidean and DUST distances, it does not provide any quality guarantees in the context of uncertain time series similarity matching. (In contrast, MUNICH and PROUD provide an additional probabilistic measure of certainty for the computed similarity

value.) Nevertheless, the measures we describe below take a first step away from the assumption, which the techniques we examined so far make, that neighboring points in a time series are independent.

4.4.1 Neighborhood-Aware Models

Given a series of noisy measurements $S =< v_1, v_2, ..., v_n >$, the moving average of these measurements S_m is obtained by substituting each value v_i with m_i, defined as the average of values $v_{i-w}, ..., v_i, v_{i+w}$:

$$m_i = \frac{\sum_{j=i-w}^{i+w} v_j}{2w+1} \tag{4.15}$$

where w is a user-defined parameter that defines the window width $2w+1$ to be considered in the average. In the moving average, all samples are weighted equally.

A variant of the moving average, namely the exponential moving average, has been introduced to weigh more the nearest neighbors of the current value, through an exponentially decaying factor. The exponential moving average of sequence S, S_e, is obtained by substituting each value v_i with e_i, defined as follows:

$$e_i = \frac{\sum_{j=i-w}^{i+w} v_j e^{-\lambda|j-i|}}{\sum_{j=i-w}^{i+w} e^{-\lambda|j-i|}} \tag{4.16}$$

where λ controls the exponential decaying factor.

The above two moving average filters require no a priori knowledge of the data distribution, and their parameters are intuitive and easy to tune, thus making these techniques widely adopted in the real world. In the next paragraphs, we introduce two variants of the moving and exponential moving averages that exploit the a priori knowledge of the error standard deviation.

Intuitively, we can weigh less the observations drawn from random variables that exhibit larger error standard deviation, as we have less confidence on the correctness of their value. The Uncertain Moving Average (UMA) is based on the moving average, where sequence S is substituted by S_p, and the point pm_i is defined as follows:

$$pm_i = \frac{\sum_{j=i-w}^{i+w} \frac{v_j}{s_j}}{2w+1} \tag{4.17}$$

where s_j is the standard deviation of random variable t_j.

The Uncertain Exponential Moving Average (UEMA) is based on the exponential moving average, where sequence S is substituted by S_e, and point pe_i is defined as follows:

$$pe_i = \frac{\sum_{j=i-w}^{i+w} v_j \frac{e^{-\lambda|j-i|}}{s_j}}{\sum_{j=i-w}^{i+w} e^{-\lambda|j-i|}} \tag{4.18}$$

At this point, we have introduced the UMA and UEMA filters. These filters allow us to reduce the signal noise, but do not define any distance function. In the subsequent experiments, we consider the Euclidean distance computed on the sequences filtered by UMA and UEMA techniques. Thus, Euclidean, UMA, and UEMA share the same distance function, but the input sequence is different.

4.4.2 Performance

We first examine the behavior of UMA and UEMA when we vary the parameters window size, w, and decaying factor, λ. Figure 4.13 depicts the effect of varying w between $0-20$ on the F_1 score. The results are averaged over all datasets. Note that when $w = 0$, UMA and UEMA degenerate to the simple Euclidean distance. We observe that the accuracy for UMA increases by 13% as we increase w from 0 to 2, and then starts falling again

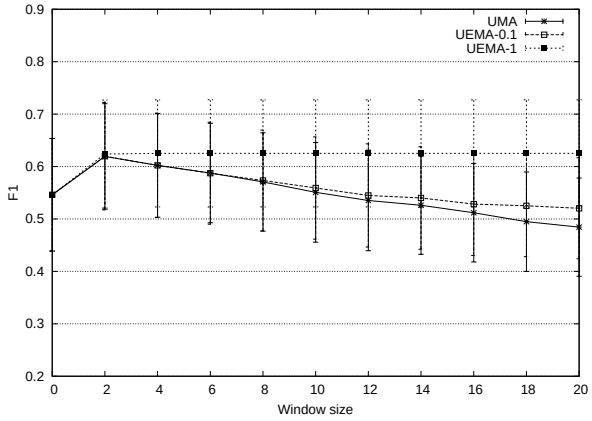


Figure 4.13: F1 score varying the window size, w, for UMA and UEMA (with $\lambda = 0.1, 1$).

as we further increase w. Evidently, aggregating many points (i.e., large w) is equally ineffective as not aggregating at all (i.e., $w = 0$), since distant neighbors do not carry much (if at all) information about the current point.

The graphs also shows the performance of UEMA for two different λ settings. For a small decaying factor, $\lambda = 0.1$, UEMA performs very close to UMA, since all the points in the window are assigned similar weights. This effect diminishes as w increases and λ introduces a higher variation among the weights of the near and distant neighbors of the current point. When we use a high value for the decaying factor, $\lambda = 1$, the effect of the distant neighbors diminishes much faster, thus, rendering the size of the window irrelevant for the performance of UEMA.

In Figure 4.14 we illustrate how the accuracy of UEMA varies when we change λ (the case $\lambda = 0$ is equivalent to UMA). The experiments show that λ has only a small effect on the performance of the algorithm, especially when the size of the window is small.

Overall, we note that UMA and UEMA exhibit a relatively stable behavior with respect to their parameters. For the rest of this study, we assume a decaying factor of $\lambda = 1$ for UEMA, and a moving average window length $W = 5$ (i.e., $w = 2$) for both UMA and UEMA.

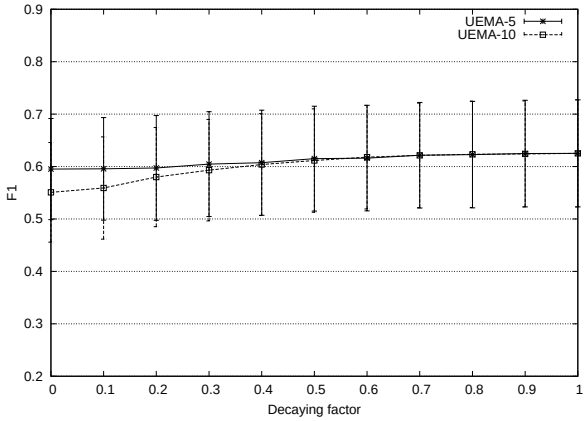


Figure 4.14: F1 score varying the decaying factor, λ, for UEMA (for $w = 5, 10$).

In the next set of experiments, we compare the accuracy of Euclidean, DUST[3], UMA, and UEMA techniques on all datasets perturbed with normal mixed error distribution, where 20% points with error standard deviation 1.0, and the remaining 80% with error standard deviation 0.4. This setting was chosen to stress-test the techniques. Every time series in each dataset was used as a query, and the results are averaged over all these time series.

Figure 4.15 depicts the results for the above experiment. The accuracy of DUST and Euclidean is almost the same, while UMA and UEMA perform consistently better, with the latter achieving the best performance among all techniques. Similar results were obtained for the uniform and exponential mixed error distributions, as shown in Figures 4.15 and 4.17, respectively.

The graphs show that (on average, across all datasets) Euclidean is always the worst performer, with a drop of 9% in its performance for the mixed exponential error distribution, which represents the hardest case. DUST performs close to Euclidean for the mixed normal and uniform dis-

[3]Based on the previous experiments, DUST performs at least as good, or better than MUNICH and PROUD for a variety of settings. Therefore, we only report the performance of DUST in these experiments for ease of exposition.

tribution, but manages to maintain the same level of performance for the mixed exponential distribution as well.

UMA and UEMA exhibit the highest accuracy levels, averaging 4% to 15%, respectively, higher than DUST, and maintaining the same level of performance across all error distributions. Overall the F_1 score of UEMA is 4% higher than that of UMA.

The above results are very interesting: the intuitive and simple UMA and UEMA techniques outperform DUST, a complex method that requires much more a priori knowledge on the data distributions. Instead, these experiments indicate that much of the knowledge is conveyed in the error standard deviation, and in the distribution of the neighboring points. UMA and UEMA are the best performers, because they do *not* assume that data points are independent, a simplifying, yet unrealistic assumption made by the techniques previously proposed in the literature.

Note that UMA and UEMA are also computationally efficient, requiring almost the same time as Euclidean, and significantly less time than DUST, PROUD, and MUNICH. All the above observations indicate that UEMA is the method of choice for similarity matching in uncertain time series, when a probabilistic measure of certainty for the similarity is not required. Even when such a measure is required, UEMA can serve as a baseline for the target performance.

4.5 Discussion

In this work, we reviewed the existing techniques for similarity matching in uncertain time series, and performed analytical and experimental comparisons of the techniques. Based on our evaluation, we can provide some guidelines for the use of these techniques.

MUNICH and PROUD are based on the Euclidean distance, while

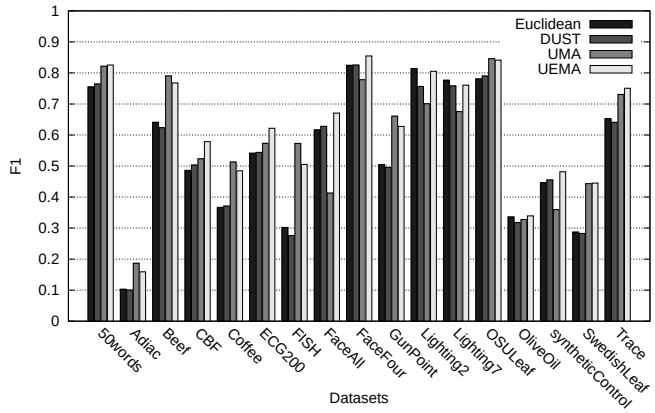


Figure 4.15: F1 score for all datasets and mixed error distribution: uniform with 20% standard deviation 1.0, and 80% standard deviation 0.4.

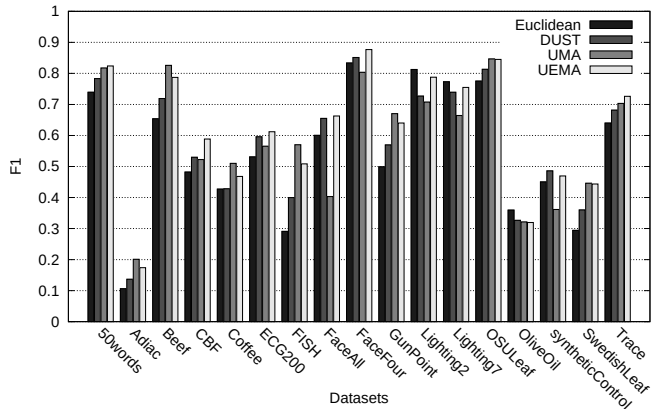


Figure 4.16: F1 score for all datasets and mixed error distribution: normal with 20% standard deviation 1.0, and 80% with standard deviation 0.4.

DUST proposes a new distance measure. Nevertheless, DUST outperforms Euclidean only if the distribution of the observation errors is mixed, and the parameters of this distribution are known.

An important factor for choosing among the available techniques is the information that is available about the distribution of the time series and its errors. When we do not have enough, or accurate information on the distribution of the error, PROUD and DUST do not offer an advantage in terms of accuracy when compared to Euclidean. Nevertheless, Euclidean does not provide quality guarantees while MUNICH and PROUD do.

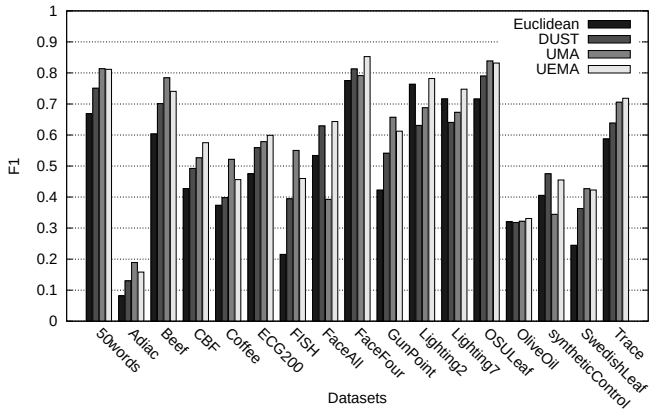


Figure 4.17: F1 score for all datasets and mixed error distribution: exponential with 20% standard deviation 1.0, and 80% with standard deviation 0.4.

The probabilistic threshold τ has a considerable impact on the accuracy of the MUNICH and PROUD techniques. However, it not obvious how to set τ, and no theoretical analysis has been provided on that. The only way to pick the correct value is by experimental evaluation, which can sometimes become cumbersome.

Our experiments showed that MUNICH is applicable only in the cases where the standard deviation of the error is relatively small, and the length of the time series is also small (otherwise the computational cost is prohibitive). However, we note that this may not be a restriction for some real applications. Indeed, MUNICH's high accuracy may be a strong point when deciding the technique to use.

The UMA and UEMA moving average filters proved to be very effective, outperforming the previous techniques in a variety of settings. This surprising result is due to the ability of the moving average to exploit the correlation of neighboring points in a very intuitive and simple manner: it reduces the effect of errors, which the filter levels out. Ignoring the strong correlation exhibited by neighboring points in the time series is not beneficial. Indeed, as our study shows, it is a severe limitation of all the techniques previously proposed in the literature.

However, we should note that the goal of MUNICH and PROUD is to provide an additional probabilistic measure of certainty for the computed similarity value. This is something that we cannot readily get from UMA, UEMA, or DUST, and may be important for certain applications.

Finally, we observe that there exist some datasets for which all techniques perform well (e.g., FaceFour and OSU-Leaf), and others for which accuracy is low (e.g., Adiac and Swedish Leaf). A close look at the characteristics of these datasets revealed that datasets for which the average distance between time series was low led to low accuracy. This is because uncertainty has a significant impact for these datasets, making it hard to distinguish the time series and select a clear winner for the similarity matching problem. On the other hand, the same level of uncertainty does not affect much datasets that have a high average distance among their time series.

4.6 Summary

The emerging area of uncertain time series processing and analysis is increasingly attracting the attention of both the research community and the practitioners in the field, mainly because of the applications and interesting problems it entails.

In this study, we evaluated the state of the art techniques for similarity matching in uncertain time series, as this operation is the basis for more complex algorithms. Apart from the techniques that were previously proposed in the literature, we also evaluated two additional, obvious alternatives that were not studied before.

Our experiments were based on 17 real, diverse datasets, and the results demonstrate that simple measures, based on moving average, outperform the more sophisticated alternatives. These results also suggest that

a promising direction is to develop measures that take into account the sequential correlations inherent in time series.

In the next Chapter we will consider the problem of evaluating top-k nearest neighbor searches revisiting the underlying uncertainty model to use the correlation of the values at neighboring time-stamps.

Chapter 5

Top-k Nearest Neighbor Search for Uncertain Data Series

In the previous chapter, we compared analytically and experimentally different uncertainty models for data series to support similarity search queries. We proceed studying the efficient evaluation of top-k nearest neighbor searches adopting the "possible worlds" model. We use series as possible instantiations to leverage the dependencies between the values at neighboring time-stamps.

Many real applications consume data that is intrinsically uncertain and error-prone. In this study we investigate the problem of finding the top-k nearest neighbors in uncertain data series. An uncertain data series is a series whose point values are uncertain. Typical sources of uncertainty in data series include sensor data, data synopses, privacy-preserving transformations and forecasting models. We introduce different formal definitions of uncertain data series and discuss their properties. We unify and improve prior studies in the field in the *Holistic-PkNN* family of algorithms. Moreover, we consider different strategies to prune the search space in spatial and metric spaces. We experimentally verify of our proposal under a variety of settings using 45 real datasets and synthetic datasets, which illustrate the effectiveness of our approach.

The rest of this chapter is organized as follows. In Section 5.1, we discuss different alternative models of uncertain data series and formally define the problem of top-k nearest neighbor queries in uncertain data series. In Section 5.2, we detail our baseline approach. In Section 5.3, we present our proposal. In Section 5.4, we discuss our adaptation of the M-tree index for uncertain data series. Extensions of our proposal are reported in Section 5.5. In Section 5.6, we present the experimental results. In Section 2.1, we survey prior studies and offer our conclusions in Section 5.7.

5.1 Preliminaries

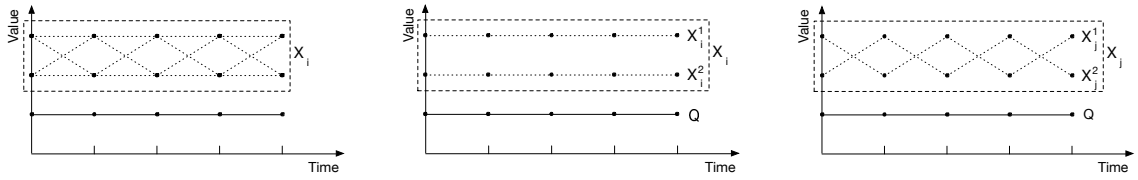
In this section, we formalize the problem after introducing some definitions. Recall that a data series S is an ordered sequence of n real valued numbers $S = S[t]$, $1 \leq t \leq n$.

An uncertain data series X is a data series whose values at each time-stamp are uncertain. We denote the value at time-stamp t of sample j of uncertain series X_i as $X_i^j[t]$. The index i may be omitted for ease of exposition. We formalize the *value-uncertainty* and the *series-uncertainty models* as follows:

Definition 5.1.1 (Value-uncertainty model) *An uncertain series X of length n is represented by m real valued samples at each time-stamp t, $1 \leq t \leq n$. The value distributions at different time-stamps are assumed to be independent. Formally, $X =< \{X^j[t] : 1 \leq j \leq m\}, ... >$, $1 \leq t \leq n$.*

Definition 5.1.2 (Series-uncertainty model) *An uncertain series X of length n is represented by m series samples. A series sample X^j is a sample drawn from the full joint distribution of X. Formally, $X = \{X^j : 1 \leq j \leq m\}$.*

An example of uncertain series using the *value-uncertainty* model is reported in Figure 5.1(a). The same uncertain series represented under the *series-uncertainty* model is shown in Figure 5.1(b). We observe that in contrary to the *series-uncertainty* model, the uncertain series reported in Figure 5.1(b) and Figure 5.1(c) cannot be distinguished under the *value-uncertainty* model.



(a) (b) (c)

Figure 5.1: Graph (a) shows an uncertain series X_i of length 5 and 2 samples at every time-stamp that is represented by the *value-uncertainty* model. Graph (b) shows the uncertain series X_i introduced in graph (a), modeled using the *series-uncertainty* model with samples X_i^1 and X_i^2. Graph (c) shows an uncertain series X_j that is distinguishable from uncertain series X_i under the *series-uncertainty* model but not under the *value-uncertainty* model.

We observe that the number of possible instantiations is exponential to the series length (m^n) under the *value-uncertainty* model and linear to the number of series samples (m) under the *series-uncertainty* model. The *series-uncertainty* model is clearly to be preferred in terms of space and time complexities. We further observe that the *value-uncertainty* model is less accurate then the *series-uncertainty* model in capturing the correlation among the series values at neighboring time-stamps: under the *value-uncertainty* model the sample value at time-stamp t doesn't depend on a single sample value at time-stamp $t' < t$. In contrary, under the *series-uncertainty* model the value at time-stamp t depends on a single series of sample values at time-stamp $t' < t$. Nevertheless, if the raw data is provided by means of multiple measurements at each time-stamp (i.e., as

in the *value-uncertainty* model), there is no statistical difference between the two modes. However, the *series-uncertainty* model is more accurate than the *value-uncertainty* if the raw data is provided by means of multiple series samples. Following the Occamś Razor principle, we conclude that there are no benefits in adopting the *value-uncertainty* model and the *series-uncertainty* model should always be preferred instead.

Definition 5.1.3 (Dataset) *A dataset of uncertain data series D is a set of uncertain data series $D = \{X_1, X_2, ..., X_N\}$ of size $N = |D|$. The uncertain series in dataset D are represented by m samples each, and each series sample has length n. Similarly to prior works [26, 16], we further assume that the uncertain series in D are independent, i.e., samples drawn from uncertain series X_i are independent of samples drawn from uncertain series X_j, $\forall i \neq j$.*

The distance between uncertain series X and query Q is uncertain, and under the *series-uncertainty* model is represented by a set of distance samples obtained evaluating the distance function between Q and each instantiation X^j:

Definition 5.1.4 (Sample distance distribution) *The sample distance distribution between uncertain series X and query series Q is denoted by $Dist(X, Q)$ and is defined as $Dist(X, Q) = \{dist(X^j, Q) : j \in 1, ..., m\}$, where $dist(X^j, Q)$ is the distance measure between Q and the jth instance of X.*

We consider the Euclidean distance as reference implementation of the $dist(\cdot)$ function, however other distance measures [14, 34, 25] can be considered as well.

Table 5.1 summarizes the most important symbols used in the rest of the chapter.

Notation	Description
Q	Query series
k	Result set size
n	Series length
m	Number of uncertain series samples
D	Dataset of uncertain series
N	Number of uncertain series in dataset D
X_i	ith uncertain series
X_i^l	lth sample of ith uncertain series
$X_i^l[t]$	lth sample of X_i at time-stamp t
$B_i = [B_i^{lb}, B_i^{ub}]$	PNN bounds for uncertain series X_i
$S_i = \{S_i^l\}$	distance partition for X_i
S_i^l	lth distance interval in S_i
W_i^l	Weight of lth distance interval in S_i
$I(X)$	1 if X is true, 0 otherwise

Table 5.1: Notation used in this chapter.

Now we are ready to formulate the problem of top-k nearest neighbor search in uncertain series.

5.1.1 Problem Statement

We consider an adaptation of the formulation of the top-k probable nearest neighbors originally proposed in [26] and then used in more recent studies [56, 16, 28].

Let D be a dataset of N uncertain series modeled under the *series-uncertainty* model with series length n and number of samples m. Given a query series Q, the probability of an uncertain data series X_i to be the NN (Nearest Neighbor), denoted by $P_{NN}(Q, X_i)$, is:

$$P_{NN}(Q, X_i) = \int Pr\left(Dist(Q, X_i) = s \wedge \bigwedge_{\forall j \neq i} Dist(X_j, Q) > s \right) ds \tag{5.1}$$

Let $r(i)$ be a rank function s.t. $r(i) \leq r(j)$ iff $P_{NN}(Q, X_i) \geq P_{NN}(X_j, Q)$. We can now introduce the definition of the top-k probable nearest neighbors:

Problem 5.1.1 (Top-k Probable Nearest Neighbors) *Given a dataset D and query Q, the top-k probable nearest neighbor search Top-k-$P_{NN}(D, Q, k)$ returns the k uncertain series X_i with the largest $P_{NN}(Q, X_i)$ probabilities. Formally, the result set is defined as $\{X_{r(1)}, ..., X_{r(k)}\}$.*

5.2 Baseline Algorithm

In this section, we present the baseline algorithm to evaluate top-k probable nearest neighbor queries under the *series-uncertainty* model. Thanks to the independence assumption between the uncertain series $X_i \in D$, Eq. 5.1 can be simplified to:

$$P_{NN}(Q, X_i) = \int Pr(Dist(Q, X_i) = s) \prod_{\forall j \neq i} Pr(Dist(X_j, Q) > s) ds \quad (5.2)$$

where the Probability Density Function (PDF)

$$Pr(Dist(Q, X_i) = s)$$

is essentially used to weight the second term and the inverse of the Cumulative Density Function (CDF)

$$Pr(Dist(X_j, Q) > s)$$

is estimated as the ratio of samples matching the inequality condition:

$$Pr(Dist(X_j, Q) > s) = \frac{1}{|\{r \in Dist(X_j, Q) : r > s\}|} \quad (5.3)$$

For ease of exposition, we introduce the $I(X)$ indicator function: $I(X)$ evaluates to 1 if the condition X holds, 0 otherwise. The evaluation of Eq. 5.2 can then be reduced as follows:

$$P_{NN}(Q, X_i) = \frac{1}{m^N} \sum_{s \in Dist(Q,X_i)} \left[\prod_{\forall j \neq i} \left(\sum_{r \in Dist(X_j,Q)} I(r > s) \right) \right] \tag{5.4}$$

where $1/(m^N)$ results from further simplifications of the $Dist(Q, X_i) = s$ terms in Eq.5.2, and $Dist(\cdot)$ is a set of distance samples previously defined in Definition 5.1.4.

We are now ready to introduce the *PkNN-Selection* algorithm that evaluates Top-k-$P_{NN}(D, Q, k)$ queries in Algorithm 1.

Algorithm 1 PkNN-Selection(D: dataset, Q: query sequence, k: result set size).

1: $B_i \leftarrow P_{NN}(Q, X_i)$ for all $X_i \in D$ using Eq. 5.4
2: $T \leftarrow select(\{B_1, ..., B_N\}, k)$

$P_{NN}(Q, X_i)$ probabilities are evaluated in line 1 and the k uncertain series with the top-k highest NN probabilities are identified in line 2 using an adaptation of the selection algorithm [55]. The selection algorithm is a sort-based algorithm that can be used to identify efficiently the k largest values in an array.

5.2.1 Complexity Analysis

The evaluation of Eq. 5.4 has a CPU cost of $O(Nm^2)$. Considering all candidate uncertain series, line 1 has a CPU cost of $O(m^2N^2)$, where m is the number of samples of each uncertain series and N is the number of uncertain series in the dataset D. The time complexity of the selection algorithm is linear to the size of the array (on average) and is bounded by $O(N)$. We note that the evaluation of $P_{NN}(Q, X_i)$ dominates the CPU cost.

5.3 Proposed Approach

In this section, we present our algorithms for the efficient evaluation of Top-k-$P_{NN}(D, Q, k)$ queries. We start off by presenting an approach that uses coarse representations of the distance samples to obtain an estimate of the PNN probability bounds. A technique based on incremental refinements of the PNN probability bounds is then presented. We conclude discussing different algorithms to prune the search space and to select the best PNN bound refinements in the iterative search.

5.3.1 Bounding the PNN Probability Estimates

Recall that uncertain series X_i are uncertain points in high-dimensional spaces whose dimensions are correlated. The evaluation of the PNN probabilities is based on the distance measurements between the query series Q and the uncertain series X_i. Once the distance samples have been determined, the raw series X_i are not accessed anymore.

Under the *series-uncertainty* model, the distribution between query Q and the uncertain series X_i is represented by a set of distance samples $Dist(Q, X_i)$, previously defined in Eq. 5.1.4. In the following we show how summarizations of the distance samples in $Dist(Q, X_i)$ can be used to bound the PNN probability estimates for candidate X_i.

Definition 5.3.1 (Distance interval) *A distance interval S_i^l is a region in the distance space that represents a subset of the distance samples in $Dist(Q, X_i)$. The lower and upper bounds of S_i^l are denoted by $lb(S_i^l)$ and $ub(S_i^l)$, respectively. The associated weight is denoted by W_i^l and is defined as the ratio of samples in $Dist(Q, X_i)$ falling within the interval bounds. Any distance interval S_i^l represents at least a distance sample, i.e., $W_i^l > 0$. A distance partition S_i is a set of disjoint distance intervals S_i^l that partition the samples in $Dist(Q, X_i)$.*

Given a distance partition S_i, these two properties hold: first, distance intervals S_i^l and S_i^k do not overlap, $\forall l \neq k$. Second, the sum of the weights of the distance intervals in the S_i partition equals to one, i.e. $\sum_l W_i^l = 1$.

Distance partitions at different levels of detail can be instantiated to represent the samples in $Dist(Q, X_i)$. The number of distance intervals (denoted by d) in the partitions ranges between 1 and m. We denote with S_i^{min} the partition instantiation composed by a single distance interval (coarsest level, $d = 1$). S_i^{max} denotes the partition instantiation composed by m distinct distance intervals (finest level, $d = m$). An example of S_i^{min}, S_i^{max} and an intermediate valid distance partition instantiation denoted by S_i^{mid} is reported in Figure 5.2.

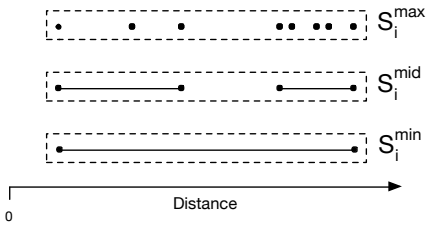


Figure 5.2: Example of valid distance partition instantiations S_i^{min}, S_i^{mid} and S_i^{max} representing the distance samples in $Dist(Q, X_i)$.

Distance partitions can be used to estimate lower- and upper-bounds of the PNN probabilities, denoted by $P_{NN}^{lb}(Q, X_i)$ and $P_{NN}^{ub}(Q, X_i)$, respectively.

The lower-bound $P_{NN}^{lb}(Q, X_i)$ is determined by using the upper-bounds $ub(S_i^l)$ as representatives of the distance intervals in partition S_i and the lower-bounds $lb(S_j^l)$ as representatives of the distance intervals in the other partitions S_j. The resulting adaptation of Eq.5.4 is:

$$P_{NN}^{lb}(Q, X_i) = \sum_{S_i^a \in S_i} \left[W_i^a \prod_{\forall j \neq i} \left(\sum_{S_j^b \in S_j} W_j^b \cdot I(ub(S_i^a) < lb(S_j^b)) \right) \right] \tag{5.5}$$

Similarly, the upper-bound $P_{NN}^{ub}(Q, X_i)$ is determined by using the lower-bounds $lb(S_i^l)$ as representatives of the distance intervals in partition S_i and the upper-bounds $ub(S_j^l)$ as representatives of the distance intervals in the other partitions S_j. The resulting adaptation of Eq.5.4 is:

$$P_{NN}^{ub}(Q, X_i) = \sum_{S_i^a \in S_i} \left[W_i^a \prod_{\forall j \neq i} \left(\sum_{S_j^b \in S_j} W_j^b \cdot I(lb(S_i^a) < ub(S_j^b)) \right) \right] \tag{5.6}$$

We denote the probability interval identified by the lower- and upper-bounds of the PNN probability estimates by B_i:

$$B_i = [P_{NN}^{lb}(Q, X_i), P_{NN}^{ub}(Q, X_i)] \tag{5.7}$$

We observe that if we use the finest representation of the distance samples S_i^{max} for all uncertain series $X_i \in D$, then $P_{NN}^{lb}(Q, X_i)$ and $P_{NN}^{ub}(Q, X_i)$ can be reduced to Eq.5.4 and degenerate to the same value, $P_{NN}(Q, X_i)$.

The CPU cost of determining the PNN probability bounds using Eq.5.5 and Eq.5.6 is bounded by the number of distance intervals in each partition. A low number of distance intervals in each partition is to be preferred. However, PNN bounds might not be tight enough to discriminate the answer set and a more fine-grained representation of the distance partitions may be required to improve sufficiently the PNN bounds. In the next section, we introduce the *Holistic-PkNN* algorithm that solves efficiently this problem.

5.3.2 The Holistic-PkNN Algorithm

In this section we present *Holistic-PkNN*, an iterative algorithm to evaluate Top-k-$P_{NN}(D, Q, k)$ queries as defined in Section 5.1.1. The *Holistic-PkNN* algorithm uses the PNN probability bounds B_i as defined in Section 5.3.1 and refines incrementally the distance partitions S_i until convergence at a reduced CPU cost.

Let $top_k(V)$ be the kth largest value in a set V of values, not necessarily distinct. Let B^{lb} and B^{ub} be the set of PNN probability lower-bounds and the set of PNN probability upper-bounds, respectively. The critical region $[c, d]$ is defined as follows:

Definition 5.3.2 (Critical region) *Let $c = top_k(B^{lb})$ and $d = top_{k+1}(B^{ub})$. The critical region R is defined as the probability interval $R = [c, d]$. The critical region is empty if $c > d$.*

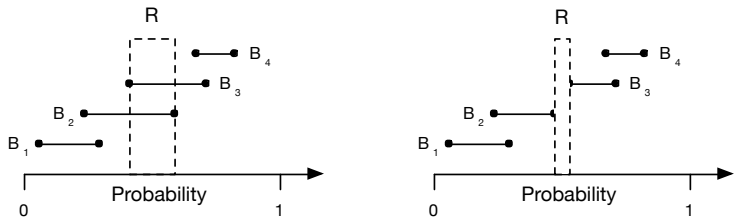
Uncertain series X_i whose PNN probability upper bound $P^{ub}_{NN}(Q, X_i)$ is lower than the lower bound c of the critical region R have zero probability of being the NN, and can be safely pruned. In contrary, uncertain series X_i whose PNN probability lower bound $P^{lb}_{NN}(Q, X_i)$ is higher than the upper bound d of the critical region R can be safely appended to the result set. We can now formally introduce the set of the active candidates:

Definition 5.3.3 (Active candidates) *Uncertain series X_i is an active candidate iff its PNN probability interval B_i overlaps with the critical region R. We denote with X^* the set of the uncertain series that qualify as active candidates.*

The set of the active candidates X^* identifies the uncertain series X_i that can be eligible to enter the result set but require tighter PNN probability bounds to make a final decision. Note that, if $c > d$, then the critical region R is empty, and the set of active candidates X^* is empty. This condition

is encountered when the PNN probability bounds are sufficiently tight to discriminate the result set without further refinements.

Figure 5.3(a) shows an example of a critical region (R) for $k = 2$. The uncertain series corresponding to PNN interval B_4 is clearly part of the result set since it dominates all the other PNN bounds. However, PNN bounds B_3 and B_2 overlap and we cannot discriminate between them. Figure 5.3(b) shows an example of a critical region for $k = 2$, s.t. PNN probability bounds B_4 and B_3 dominate all the other PNN bounds and there is no uncertainty on the memberships of the result set.



(a) (b)

Figure 5.3: Graph (a) shows an example of critical region R with overlapping PNN probability bounds B_3 and B_2. Graph (b) reports an example of empty critical region R.

PNN probability bounds of uncertain series $X_i \in X^*$ are tightened by refining some of the distance partitions. A partition is refined by increasing the number of its distance intervals. We observe that the underlying relationships between different candidates can lead to tighter PNN bounds for candidate X_i after refining a distance interval in partition S_j, where $i \neq j$.

Let S^* be the set of the distance partitions $S_1, ..., S_N$. Let B^* be the set of the PNN probability bounds $B_1, ..., B_N$. The overall *Holistic-PkNN* procedure that refines selectively the distance partitions until convergence of the result set is illustrated in Algorithm 2.

Algorithm 2 Holistic-PkNN(D: dataset, Q: query sequence, k: result set size).

1: $S_i \leftarrow$ init-distance-partitions(Q, X_i) for all candidates $X_i \in D$
2: $B_i \leftarrow P_{NN}(Q, X_i)$ bounds using Eq. 5.5 and Eq. 5.6 for all candidates $X_i \in D$
3: Update critical region $[c, d]$ using Def. 5.3.2
4: **while** $c \leq d$ **do**
5: $C \leftarrow$ find-critical$(B^*, [c, d])$
6: $R \leftarrow \bigcup_{X_i \in C}$ find-splits(S_i)
7: **for** $S_i^l \in R$ **do**
8: $S_i \leftarrow$dist-refine(S_i^l)
9: **end for**
10: $B_i \leftarrow P_{NN}(Q, X_i)$ bounds using Eq. 5.5 and Eq. 5.6 for all active candidates $X_i \in X^*$
11: Update critical region $[c, d]$ using Def. 5.3.2
12: **end while**
13: $T \leftarrow \{X_i : P_{NN}^{lb}(Q, X_i) > d\}$

Line 1 initializes the distance partitions S_i for all $X_i \in D$ to their respective S_i^{min} instances. Efficient algorithms to determine the S_i^{min} instances are presented in Section 5.3.4. In Line 2, lower- and upper-bounds B_i for all $X_i \in D$ are initialized using Eq. 5.5 and Eq. 5.6. The bounds c, d of the critical region R are updated in Line 3 using Definition 5.3.2. This concludes the initialization of the data structures before the iterative refinement of the PNN probability bounds. The PNN bounds are then tightened iteratively in Lines 4-12 until convergence of the result set, i.e., the termination condition $c > d$ is met. In each iteration a set of PNN bounds to be tightened is selected (set C) and the distance intervals expected to improve the selected PNN bounds are refined (set R). The PNN probability bounds of the candidates $X_i \in X^*$ and the critical region $[c, d]$ are updated at the end of each iteration. Line 5 identifies the uncertain series $X_i \in C$ whose PNN bounds are selected for improvement. Efficient implementations of the *find-critical* procedure are presented in Section 5.3.3. In Line 6 the distance intervals S_i^l to be refined are identified, and then refined in Lines

7-9. The efficient evaluation of the refinements is discussed in Section 5.3.4. Line 10 updates the PNN probability bounds B_i of the active candidates $X_i \in X^*$. The bounds c, d of critical region R are then updated in Line 11. Finally, the result set is constructed in Line 13.

We note that the *find-splits* procedure may return the same distance interval S_i^l to tighten the PNN probability bounds of different candidates. However, the distance interval S_i^l gets refined only once by the *dist-refine* function. Multiple PNN bounds can benefit holistically from the same refinement, keeping the global number of refinements as low as possible (thus making the evaluation of the PNN bounds in Line 10 more efficient).

Lemma 1 (Termination) *Algorithm 2 always terminates after a finite number of partition refinements.*

Proof. Let S^* be the set of distance partitions S_i initialized in Line 1 with their respective S_i^{min} instances. Lines 4-9 ensure that at least one partition S_i is refined at every iteration. In the worst case, all distance partitions $S_i \in S^*$ are refined completely, obtaining their respective S_i^{max} instances. Consequently, the PNN probability bounds B_i converge to the exact probability estimate, i.e. $P_{NN}^{lb}(Q, X_i) = P_{NN}^{ub}(Q, X_i)$. It is then easy to show that $top_{k+1}(B^{ub}) < top_k(B^{lb})$, i.e., $d < c$. We assume that the $P_{NN}(Q, X_i)$ estimates are distinct values, i.e., if $P_{NN}(Q, X_i) = P_{NN}(Q, X_j)$ then it must be that $i = j$.

5.3.3 Tightening the PNN Bounds

The optimal search path to identify the top-k probable nearest neighbors in the *Holistic-PkNN* method (Algorithm 2) minimizes the number of partition refinements (Lines 7-9) and minimizes the number of evaluations of the PNN probability bounds (Line 2,10). Unfortunately, its determination

is computationally prohibitive: Its identification would require an extensive search in the solution space of the possible refinements, altogether with repeated evaluations of the PNN probability bounds. The incurred CPU cost would deny any expected benefit provided by the iterative refinements, and this motivates the introduction of efficient sub-optimal algorithms.

In this section, we discuss efficient implementations of the functions *find-critical* and *find-splits* used in Algorithm 2. While the procedure *find-critical* identifies the PNN bounds B_i to be tightened, procedure *find-splits* finds the refinements in the distance partitions S_i to be applied.

find-critical

We present an adaptation of the *multi-simulation* algorithm firstly introduced in [76]. The *multi-simulation* selection heuristic returns up to two PNN probability bounds B_i to be improved with provable optimally bounds.

The algorithm starts off trying to return an uncertain series $X_i \in X^*$ whose PNN bounds B_i contain the critical region R, i.e. $R \subseteq B_i$ (named double-crosser). The distance partition S_i overlaps with other distance partitions S_j, $j \neq i$ (that identify the c, d boundaries of the R region), and its refinement is expected to provide a significant overall progress. If there is no matching candidate, the algorithm tries to identify a pair of uncertain series $X_i, X_j \in X^*$ whose PNN bounds B_i and B_j overlap with the left boundary (named left-crosser) and the right boundary (named right-crosser) of the critical region R. If there is no pair of matching candidates, the candidate $X_i \in X^*$ whose PNN bounds B_i have the largest overlap with the critical region R and contains all the other B_j s.t. $X_j \in X^*$ is returned. The approach is reported in Algorithm 3.

The procedure returns immediately if no refinement is required (Lines 1-3). First, we return the active candidate $X^i \in X^*$ s.t. the critical region

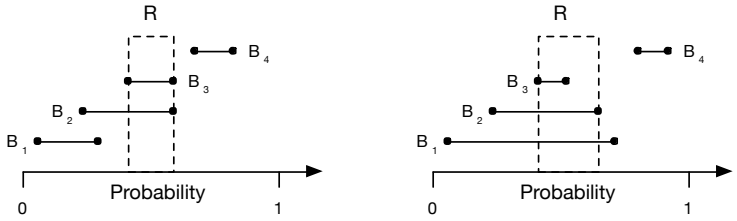
Algorithm 3 Multi-Simulation(D: dataset, $B*$: probability bounds, $[c, d]$: critical region).

1: **if** $c > d$ **then**
2: **return** $\emptyset$
3: **end if**
4: **if** $\exists X_i \in X^*$ s.t. $B_i^{lb} < c \wedge d < B_i^{ub}$ **then**
5: **return** X_i // double-crosser
6: **end if**
7: **if** $\exists X_i, X_j \in X^*$ s.t. $B_i^{lb} < c \wedge d < B_i^{ub}$ **then**
8: **return** X_i and X_j // left- and right-crosser pair
9: **end if**
10: **return** $X_i \in X^*$ s.t. the overlap between B_i and critical region $[c, d]$ is maximal and contains all the other B_j s.t. $X_j \in X^*$ // maximal crosser

is within its PNN bounds (Lines 4-6). Otherwise, we return a pair of active candidates $X_i, X_j \in X^*$ s.t. the lower- and upper-bounds of the critical region overlap respectively with the PNN bounds of X_i and X_j (Lines 7-9). Finally, we return the active candidate $X^i \in X^*$ s.t. it maximizes the overlap of its PNN bounds with the critical region and contains all the other B_j s.t. $X_j \in X^*$ (Line 10).

We report an example in Figure 5.3 and Figure 5.4, that show two possible configurations for the critical region R. In Figure 5.3(a), PNN bounds B_2 and B_3 form a pair of left- and right-crossers. In Figure 5.3(b), R is empty and the algorithm returns immediately. In Figure 5.4(a), PNN bound B_2 is a left crosser and there are no right crossers. In Figure 5.4(b), PNN bound B_1 is a double crosser and B_2 is a left crosser.

Despite the provable optimality bound guarantees of the *multi-simulation* algorithm, the experiments show that is in general more convenient to consider all active candidates X^* in the tightening process rather than selecting up to two among the most promising ones. The same conclusion applies to other heuristics, such as the one considered in [16] where only the the



(a) (b)

Figure 5.4: Graph (a) shows the critical region R, where B_2 is a left crosser. Graph (b) shows the critical region R, where B_1 is a double crosser and B_2 is a left crosser.

uncertain series X_i with the largest PNN upper bound is selected for refinement.

find-splits

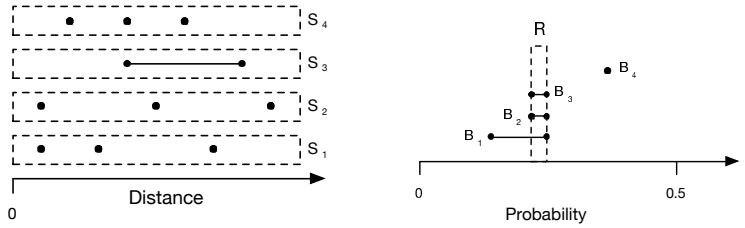
Given a set or uncertain series R, the *find-splits* procedure identifies the best distance intervals S_i^l of the distance partitions S_i to be splitted, i.e., refined. First, we discuss the importance of the dependencies across different distance partitions in the selection of the refinements to apply.

Lemma 2 (Dependencies in Distance Partitions) *Tightening the PNN probability bounds B_i of candidate $X_i \in X^*$ may require some refinements in the distance partition S_j, where $j \neq i$. Uncertain series X_j is not necessarily in the active set, i.e., $X_j \notin X^*$.*

Demonstration by example. Let D be a dataset with $N = 4$ and $m = 3$. Let S_1, S_2, S_3 and S_4 be the instantiated distance partitions: $S_1 = \{[2,2] : 0.33, [4,4] : 0.33, [6,6] : 0.33\}$, $S_2 = \{[4,8] : 1\}$, $S_3 = \{[1,1] : 0.33, [5,5] : 0.33, [9,9] : 0.33\}$ and $S_4 = \{[1,1] : 0.33, [3,3] : 0.33, [7,7] : 0.33\}$.

The PNN probability estimates determined using the Eq.5.5 and Eq.5.6 result in the following B_i bounds: $B_1 = [0.14, 0.25]$, $B_2 = [0.22, 0.25]$, $B_3 = [0.22, 0.25]$ and $B_4 = [0.37, 0.37]$.

We want to identify the top-2 most probable nearest neighbors, i.e., $k = 2$. Figure 5.5(a) and Figure 5.5(b) show the corresponding distance partitions S_i and the PNN probability bounds B_i, respectively.



(a) (b)

Figure 5.5: Graph (a) shows the distance partitions S_i and graph (b) reports their respective PNN bounds B_i.

We observe that the candidate uncertain series X_4 and X_1 can be safely appended to the result set and discarded, respectively. We are unable to discriminate the PNN probabilities of X_2 and X_3. However, distance partitions S_1, S_2 and S_3 have been fully refined to their S_i^{max} instantiations and cannot be refined further. There is no other choice than refining samples in the distance partition S_2, whose respective uncertain series X_2 is not in the set of the active candidates X^* and has zero probability of being part of the result set.

We introduce the *Pair-split* method in Algorithm 4 as baseline implementation of the *find-split* procedure. Let $width(S_i^a)$ be the distance width of the distance interval S_i^a.

The algorithm iterates over all combinations of the (S_i^a, S_j^b) pairs (Lines 3-9). Line 4 defines a score based on the two interval weights and the respective distance widths. Lines 5-8 select the pair (S_i^a, S_j^b) s.t. they overlap with the largest score value.

Algorithm 4 Pair-split(S_i)

1: $R \leftarrow \emptyset$
2: $score_{best} \leftarrow 0$
3: **for** $\forall\ (S_i^a, S_j^b)$ s.t. $S_i^a \in S_i \wedge S_j^b \in S_j \wedge i \neq j$ **do**
4: $\quad score \leftarrow W_i^a \cdot width(S_i^a) + W_j^b \cdot width(S_j^b)$
5: $\quad$ **if** $score > score_{best} \wedge S_i^a$ overlaps with S_j^b **then**
6: $\quad\quad R \leftarrow \{S_i^a, S_j^b\}$
7: $\quad\quad score_{best} \leftarrow score$
8: $\quad$ **end if**
9: **end for**

Intuitively, splitting the pair of overlapping distance intervals (S_i^a, S_j^b) whose weighted width is the largest is expected to improve the PNN probability bounds of the respective PNN probability estimates B_i and B_j, respectively.

We note that the *pair-split* algorithm considers only pair-wise dependencies between distance intervals. However, there may be a distance interval S_i^a that overlaps with a large number of distance intervals S_j^b, whose score is individually too low to get selected. In the next section, we propose heuristics that overcome this limitation.

Uncertainty-aware distance refinements

In this section we discuss how to identify a pair of refinements to tighten the PNN bounds B_i by looking explicitly at the candidate refinements in distance partition S_i and in distance partitions S_j, $\neq i$. The proposed heuristics consider the pair-wise dependencies between different distance partitions S_i and S_j, $i \neq j$.

First, we identify the best distance interval S_i^a to tighten the PNN bounds B_i. We observe that refining S_i^a may be beneficial to tighten B_i only if it overlaps with one or more distance intervals S_j^b, $j \neq i$. Among the S_i^a candidate refinements, we select the S_i^a candidate that maximizes the

weighted sum of the overlapping distance intervals with all other partitions S_j s.t. $j \neq i$. Function *select-inner* implements this strategy:

$$\text{select-inner}(Q, X_i) = \underset{S_i^a \in S_i}{\text{argmax}} \; W_i^a \prod_{\forall j \neq i} \left(\sum_{S_j^b \in S_j} W_j^b \cdot I(lb(S_i^a) < ub(S_j^b) \wedge ub(S_i^a) > lb(S_j^b)) \right) \tag{5.8}$$

Second, we identify the best distance interval S_j^b to tighten the PNN bounds B_i s.t. $j \neq i$. Similarly, we observe that refining S_j^b may be beneficial to tighten B_i only if it overlaps with one or more distance intervals in S_i. Among the S_j^b candidate refinements, we select the S_i^b candidate refinement that maximizes the weighted sum of overlapping distance intervals with the distance intervals in partition S_i. Function *select-outer* implements this strategy:

$$\text{select-outer}(Q, X_i) = \underset{S_j^b \in S_j, \forall j \neq i}{\text{argmax}} \; W_j^b \sum_{S_i^a \in S_i} W_i^a \cdot I(ub(S_i^a) > lb(S_j^b) \wedge lb(S_i^a) < ub(S_j^b)) \tag{5.9}$$

The *find-split* procedure can be implemented by returning the distance intervals identified by the *select-inner* and the *select-outer* heuristics. We observe that the computation of Eq.5.8 can be combined efficiently with the evaluation of the PNN upper-bound in Eq.5.6 because of the similarities in the formulation and in the verified inequalities. Eq. 5.9 cannot be combined in a similar way with the evaluation of the PNN intervals because of the different ordering in the enumeration of the distance interval pairs.

5.3.4 Managing the Distance Partitions

In this section we discuss the efficient implementation of the *init-distance-partitions* procedure to initialize the distance partitions S_i and their incremental refinement required in the *dist-refine* function in Algorithm 2.

Initialization

The *init-distance-partitions* procedure initializes the distance partitions S_i for all candidates $X_i \in D$. We consider two different implementations that can be used to construct the distance partitions using their S_i^{min} instantiations.

First, we present distance bounds inspired by the spatial properties of the uncertain series and prior works on spatial indexes for time series. The value of an uncertain series X_i can be bounded by the minimum and maximum values at each time-stamp across all its instantiations X_i^l, where $1 \leq l \leq m$. The lower-bound series of uncertain series X_i at time-stamp t (denoted by $X_i^{lb}[t]$) is defined as:

$$X_i^{lb}[t] = X_i^l[t] : X_i^l[t] \leq X_i^k[t] \ \forall \ k \in \{1, ..., m\} \tag{5.10}$$

Similarly, the upper-bound series of uncertain series X_i at time-stamp t (denoted by (denoted by $X_i^{ub}[t]$) is defined as:

$$X_i^{ub}[t] = X_i^l[t] : X_i^l[t] \geq X_i^k[t] \ \forall \ k \in \{1, ..., m\} \tag{5.11}$$

Definition 5.3.4 (Uncertain series envelope) *Let X_i^{lb} and X_i^{ub} be the lower-bound and upper-bound series of uncertain series X_i, respectively. The uncertain series envelope of uncertain series X_i is defined as the pair of series $E_i = (X_i^{lb}, X_i^{ub})$.*

Figure 5.6 shows an example of an uncertain series envelope. We observe that the uncertain series envelope E_i identifies the smallest n-dimensional hyper-rectangle enclosing all instantiations of uncertain series X_i.

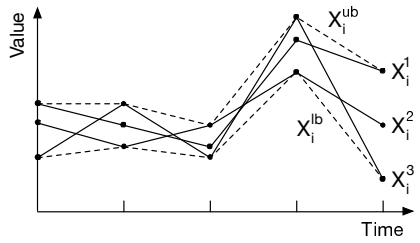


Figure 5.6: Example of uncertain series envelope.

Note that Uncertain series envelopes are equivalent to the concept of Minimum Bounding Rectangles (MBRs), a popular representation of bounding regions in spatial indexes. Envelopes can be pre-computed, since they don't depend on the query series. Given a query Q, an uncertain series envelope E_i can be used to determine the lower-bound (denoted by $spatial_{lb}(Q, X_i)$) of the distance samples in $Dist(Q, X_i)$ as follows:

$$spatial_{lb}(Q, X_i) = \sqrt{\sum_{1 \leq t \leq n} \begin{cases} 0 & \text{if } X_i^{lb}[t] \leq Q[t] \leq X_i^{ub}[t] \\ min((Q[t] - X_i^{lb}[t])^2, & \\ (Q[t] - X_i^{ub}[t])^2) & otherwise \end{cases}} \tag{5.12}$$

Similarly, the upper-bound (denoted by $spatial_{ub}(Q, X_i)$) of the distance samples in $Dist(Q, X_i)$ is defined as:

$$spatial_{ub}(Q, X_i) = \sqrt{\sum_{1 \leq t \leq n} max((Q[t] - X_i^{lb}[t])^2, (Q[t] - X_i^{ub}[t])^2)} \tag{5.13}$$

The distance bounds between uncertain series X_i and query Q are determined by measuring the minimum and maximum distances between the uncertain series envelope E_i and query Q. An equivalent formulation of the distance lower-bound can be found in [93]. On the contrary, the upper-bound $spatial_{ub}(Q, X_i)$ is novel.

We introduce now a different formulation of distance bounds, inspired by the metric properties of the distance function. Let P be a series serving as *pivot* in the metric space. A *pivot* is a series that has been selected as representative series during the distance computations, and its distance to an uncertain series X_i can be used to bound the distance between X_i and the query Q thanks to the triangle inequality property. We note that the triangular inequality holds only in metric spaces, i.e. distance spaces induced by metric distance measures.

Let d_{PX} be the maximum distance between series P and all X^l instantiations, i.e. $d_{PQ} = max(Dist(X_i, P))$. Let d_{PQ} be the distance between P and Q, i.e., $d_{PQ} = dist(P, Q)$. d_{PX} can be pre-computed, since it doesn't depend on the query Q. Figure 5.7 shows an example (projected in two dimensions for ease of exposition).

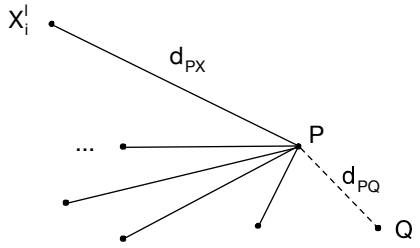


Figure 5.7: Example of metric distance bounds.

Given a query Q, a pivot P and uncertain series X, the distance between X and Q is bounded by the following lower-bound:

$$max(0, d_{PQ} - d_{PX}) \tag{5.14}$$

Similarly, one can define an upper-bound for the distance between X and Q:

$$D_{PQ} + D_{PX} \tag{5.15}$$

These bounds have been widely used in index structures for metric spaces [72, 24]. Multiple pivot series can be combined to increase to improve the distance bounds as follows. Let P^* be a set of pivot series. The best lower-bound (denoted by $metric_{lb}(Q, X_i)$) is determined using the pivot P s.t. it maximizes $max(0, d_{PQ} - d_{PX})$ and the best upper-bound (denoted by $metric_{ub}(Q, X_i)$) is determined using P' s.t. it minimizes $d_{P'Q} + d_{P'X}$:

$$metric_{lb}(Q, X_i) = \max_{P \in P^*} max(0, d_{PQ} - d_{PX}) \tag{5.16}$$

$$metric_{ub}(Q, X_i) = \min_{P \in P^*} (d_{PQ} + d_{PX}) \tag{5.17}$$

The tightness of the metric bounds depends on the quality of the selected pivot series. In the experimental evaluation we consider different strategies that have been proposed in prior works.

The distance partitions S_i for all uncertain series X_i are initialized using their spatial or metric bounds in the *Holistic-PkNN* algorithm using a linear scan on the dataset D. In the following, we discuss how distance partitions are refined incrementally.

Refinement

Let S_i be a distance partition to be refined and let S_i^l be the distance interval that has been previously selected for refinement by the *find-splits*

procedure as presented in Section 5.3.3.

If the bounds of the distance interval S_i^l have been determined using metric or spatial distance bounds (Section 5.3.4), then its bounds $lb(S_i^l)$ and $ub(S_i^l)$ are lower- and upper-estimates of $min(Dist(Q, X_i))$ and $max(Dist(Q, X_i))$, respectively. We substitute S_i^l with a new distance interval S_i^k s.t. $lb(S_i^k) = min(Dist(Q, X_i))$ and $ub(S_i^k) = max(Dist(Q, X_i))$. Please note that, after the substitution, the distance partition S_i is a new instantiation of the distance partition S_i^{min}. Distance samples $Dist(Q, X_i)$ are maintained in a sorted array. The optimal distance bounds $min(Dist(Q, X_i))$ and $max(Dist(Q, X_i))$ of the new instantiation of S_i^{min} serve as first refinement for S_i.

In case of subsequent refinements, the refinement of the distance interval S_i^l is performed by partitioning the region covered by S_i^l into two new distance intervals. Let p_{lb}, p_{ub} be a pair of pointers to the sub-array in the $Dist(Q, X_i)$ sorted array that identifies the lowest and highest distance sample in S_i^l, and let $dist_{mid}$ be the value that partitions the region defined by S_i^l into two equi-width regions. Then, the closest sample to $dist_{mid}$ in the sub-array identified by pointers p_{lb}, p_{ub} is used as largest sample in the new left partition, and its immediate next sample in the sorted array is used as lowest sample in the new right partition.

While the first refinement is very expensive because it requires the evaluation of the distance samples in $Dist(Q, X_i)$, the CPU cost of subsequent refinements is negligible, since its cost is bounded by the cost of evaluating a binary search on a subset of the distance samples, i.e., $O(log(m))$. Recall that m is the number of X_i instantiations.

5.4 Indexing Uncertain Data Series

An M-tree is a tree whose nodes represent regions in a metric space and can be used to index objects in metric spaces [72]. In this section we present our adaptation of the M-tree to index uncertain series.

We denote with $tree_i$ the sub-tree rooted at node $node_i$. Let

$$(ptr_i, count_i, pivot_i, radius_i, dist_{ij})$$

be a tuple associated to node $node_i$ defined as follows: ptr_i is a pointer to the list of children of $node_i$, $count_i$ is the count of leaves in the sub-tree $tree_i$, $pivot_i$ is a pivot series that represents the sub-tree $tree_i$, $radius_i$ is the maximum distance between $pivot_i$ and leaves in $tree_i$, and $dist_{ij}$ is the distance between $pivot_i$ and $pivot_j$ where $node_j$ is the parent node of $node_i$. IF $node_i$ is the tree root then $dist_j = 0$.

In contrast to the original M-tree where a leaf node is used to represent a set of indexed objects stored on disk, each leaf node indexes an uncertain series. Let P_i be a pivot series for uncertain series X_i. The leaf node that represents uncertain series X_i is defined by the following associated tuple:

$$leaf(X_i) = (empty, 1, P_i, max(Dist(X_i, P_i)), dist_{ij}) \tag{5.18}$$

We note that ptr_i is empty since the sub-tree contains only the leaf itself. The number of uncertain series referenced by $leaf(X_i)$ is one and P_i is a pivot series selected to represent uncertain series X_i. Finally, $max(Dist(X_i, P_i))$ is used as covering radius and $dist_{ij}$ is initialized during the construction of the tree. We can now detail the construction of the M-tree.

5.4.1 Bulk-loading Algorithm

We adapted the base version of the bulk loading algorithm presented in [29]. The method is illustrated in Algorithm 5.

Algorithm 5 M-Tree-BulkLoad(D: dataset)

1: $A \leftarrow \{leaf(X_i) : X_i \in D\}$
2: **while** $|A| > 1$ **do**
3: $S \leftarrow s$ random samples from A
4: **for** $node_i \in A - S$ **do**
5: append $node_i$ as child of $node_j \in S$ s.t. $pivot_j$ closest to pivot $node_i$ among pivots of nodes in S
6: **end for**
7: $A \leftarrow S$
8: **end while**

Line 1 initializes the leaves corresponding to the uncertain series $X_i \in D$. A set S of s random nodes is selected at Line 3 and the remaining nodes are associated to their closest nodes in S (in respect to their representative pivot series), forming a set of s clusters (Lines 4-6). The algorithm iterates until set A contains only one node, i.e. the root node of the M-tree.

Random projections and early abandoning of the Euclidean distance have been considered in our implementation to reduce the CPU cost of the pairwise distance computations between pivot series in Line 5. These optimizations are novel to M-tree implementations and reduce significantly the processing time.

5.4.2 Pruning the Search Space

In this section, we discuss how our adaptation of M-trees can be used to avoid the complete linear scan on the dataset D to initialize the distance partitions S_i. We note that if the lower-bound of the distance partition S_i is higher than the upper-bound of the distance partitions whose lower-

bound is among the lowest k lower-bounds, then candidate X_i can be safely pruned and its distance partition doesn't need to be refined further.

Intuitively, we want to identify the candidates X_i that are part of the result set or that can be eligible to enter the result set (i.e., $X_i \in X^*$). The algorithm to identify these candidates is based on an adaptation of the top-k nearest neighbor search algorithm for M-trees [29].

We note that the algorithm may return more than k candidates. This is due to the tightness of the metric bounds during the search: if they are not enough tight, then we may not be able to impose a total order on the candidates. This problem is encountered at the leaf level. In such cases, the leaves that cannot be safely pruned are either added to the result set or added to the set of the active candidates X^* and their distance bounds are used to initialize the respective distance partitions, S_i. The remaining candidates can be safely discarded.

Similarly to top-k nearest neighbor searches in certain data, the search procedure maintains a dynamic search radius d_k s.t. at least k indexed uncertain series are at a distance lower than d_k.

The sub-trees that index the leaves whose distance to Q is larger than d_k can be safely pruned. A sub-tree $tree_i$ can be pruned if this condition holds:

$$d_k < dist(pivot_i, Q) - radius_i \tag{5.19}$$

The pruning strategy can be further extended by using the pre-computed distances $dist_{ij}$ between pivot series in children and their parents.

5.5 Extensions

In this section we discuss how the algorithms presented in this study can be adapted to evaluate different formulations of top-k probable nearest

neighbor searches and to evaluate probabilistic similarity search queries.

Top-k probable nearest neighbor formulations differ in how the results are aggregated across different possible worlds or in how uncertainty is modeled. We note that the incremental refinement of the distance partitions is a general approach that can be easily adapted to other top-k probable nearest neighbor queries whose formulation uses a "possible worlds" model.

Definition 5.5.1 (Probabilistic similarity search) *Given query Q, dataset D, distance threshold ϵ and probabilistic threshold α, a probabilistic similarity search returns the uncertain series $X_i \in D$ s.t. their probability to be at a distance lower than ϵ is at least α, i.e., $Pr(dist(Q, X_i) \leq \epsilon) \geq \alpha$.*

Probabilistic similarity search queries are intrinsically easier to evaluate than top-k probable nearest neighbor queries because the eligibility of uncertain series $X_i \in D$ to be part of the result set does not depend from the other candidates. We note that, if at least αm distance samples in $Dist(Q, X_i)$ have a value lower than the distance threshold ϵ, then candidate X_i can be appended to the result set. Despite this, the M-tree adaptation for uncertain data can be extended to evaluate similarity queries on uncertain series.

5.6 Experimental results

In this section we empirically evaluate our proposal under a variety of settings, assessing time performance and accuracy. We implemented all techniques in C++ using the Standard Template Library (STL) and Boost libraries, and ran the experiments on a Linux machine equipped with eight Intel Xeon 1.80GHz processor cores and 64GB of RAM. For all results, we report the averages of the measurements obtained from 10 independent

runs, as well as the 95% confidence intervals. Confidence intervals are reported when the y-axis is not in log scale for clearness.

5.6.1 Datasets

In this study, we consider real and synthetic datasets.

A synthetic dataset is a set of independent uncertain series X_i, $1 \leq i \leq N$. Uncertain series X_i is constructed as follows. Let $walk_i$ be a random walk of length n where the value difference between neighboring time-stamps is a normally distributed random variable with zero mean and unit standard deviation. The series $walk_i$ is then normalized [74], obtaining a new series denoted by $seed_i$. Normalization is a pre-processing step that removes value shifts and amplitude changes from the series, properties usually regarded to as noise rather than signal. The lth sample of uncertain series X_i, X_i^l, is obtained by adding samples drawn from a normally distributed random variable $N(0, \sigma)$ to the $seed_i$ values. Finally, the samples are normalized again. The procedure is repeated independently for each uncertain series X_i. Please note that the seed series $seed_i$ used to generate the samples of uncertain series X_i is different from the other seed series $seed_j$, $j \neq i$. Random queries are generated similarly (and independently) to the X_i series samples.

We additionally consider 45 real datasets published in the UCR time series collection [1]. Each dataset is divided into two sets of series, *train* and *test*, and each series is associated with a class label. We assume that these raw series are samples drawn from unknown distributions that may be noisy and error-prone. Similarly to prior works on uncertain data processing, we further assume that uncertain series can be modeled as a multivariate normal distribution centered on the real samples [16, 28, 26, 62, 95, 11, 79, 31]. Uncertain series are constructed from series in the *train* sets similarly to uncertain series in the synthetic datasets: first, normalized real series are

used as seed series $seed_i$ instead of random walks. Second, the perturbation standard deviation σ is used to generate positive samples of the normal distribution, then used independently as standard deviation σ' for each sample. The resulting uncertain series is composed by a set of samples whose perturbation standard deviation varies and is controlled by σ. We adopted this perturbation model with real datasets after verifying experimentally that other perturbation models such as the one used for synthetic datasets lead to nearly the same results in terms of accuracy for all considered methods. The series in the *test* sets are used as queries.

5.6.2 Evaluation Methodology

In this section, we describe how accuracy and time performance have been assessed.

Accuracy

We verify the accuracy of our proposal on the classification task using the real UCR datasets. Given a random query series Q drawn from the *test* set, the class label associated with the uncertain series identified as the NN in the *train* set is assigned as label.

For each method we compute the confusion matrix M where M_{ij} is the count of query series assigned to class i whose ground truth class is j. Accuracy is defined as the ratio of true positives for all class labels:

$$Accuracy = \frac{1}{|test|} \sum M_{ii}, \tag{5.20}$$

where $|test|$ is the size of the *test* set. If all queries are labeled correctly, M is a diagonal matrix. The experiment is repeated several times on each dataset (*train* and *test* pair sets) to get statistically significant results.

We note that the very same results can be obtained using different algorithms that implement the top-k probable nearest neighbor queries as formulated in Section 5.1.1.

We compare the Top-k-$P_{NN}(D, Q, k)$ formulation of Top-1 probable nearest neighbors (defined in Section 5.1.1) to the NN classifier based on the *Euclidean-AVG* method. The *Euclidean-AVG* algorithm averages all samples to obtain their average series, that is then used as representative to determine a distance measure between the uncertain series and the query. The NN classifier based on these distance measures is used to label the query.

Time performance

We compare the performance of our proposal to two algorithms which are the *Baseline* algorithm (Algorithm 1) and an adaptation of the *Find-TopK-PNN* method [16]. The algorithm *Find-TopK-PNN* has been designed to minimize the number of I/O operations: uncertain series are retrieved in min-distance order to the query. The PNN probability upper-bound of a virtual object is used to bound the PNN upper-bound probabilities of all non-retrieved objects, and it is used to control the retrieval of new uncertain series until convergence is reached. Our adaptation, named *Holistic-PkNN-Virtual*, considers our best-performing combination of the procedures for the initialization and the refinement of the distance partitions and for the evaluation of the PNN probability bounds.

Our proposal *Holistic-PkNN* is further decomposed in four distinct versions, which identify the different combinations of the *find-critical* and *find-splits* implementations: The *find-critical* function can be implemented using the *multi-simulation* procedure [76] (Algorithm 3, denoted by the suffix *M*) or enumerating all candidates whose PNN probability bounds overlap with the critical region (denoted by the suffix *H*). The *find-splits*

function can be implemented using the *pair-split* method (Algorithm 4, denoted by the suffix *P*) or using the *select-inner* and *select-outer* procedures (Eq. 5.8 and Eq. 5.9, denoted by the suffix *S*). From our experiments, the optimal configuration is *Holistic-PkNN-HS*, also denoted in short by *Holistic-PkNN*.

We stress that the algorithms *Baseline*, *Holistic-PkNN-Virtual* and *Holistic-PkNN* are different algorithms that return the very same result set as defined by the formulation of top-k probable nearest neighbor queries defined in Section 5.1.1.

The parameters considered in the experiments are summarized in Table 5.2. When not explicitly stated, we use the default configuration value (highlighted in bold).

Parameter	Range
No. of samples (m)	$[100, ..., \mathbf{500}, ..., 1000]$
No. of uncertain series (N)	$[100, ..., \mathbf{1000}, ..., 100000]$
Standard deviation (σ)	$[0.1, ..., \mathbf{0.5}, ..., 1]$
Series length (n)	$[100, ..., \mathbf{256}, ..., 1000]$
Result set size (k)	$[\mathbf{1}, ..., 16]$

Table 5.2: Experiment parameter configuration ranges. Default values are indicated in bold.

5.6.3 Quality Results

In this section, we report our results on the classification accuracy defined in Eq. 5.20 using 45 real datasets. The number of samples representing each uncertain series is $m = 100$. The experiment shows the accuracy by varying the perturbation standard deviation σ for the NN classifier defined using the *Euclidean-Avg* procedure and the Top-k-$P_{NN}(D, Q, k)$ formulation of top-1 probable nearest neighbors. The results are reported in Figure 5.8. We observe that, as the perturbation standard deviation

increases, the top-1 probable nearest neighbor formulation is up to 6.3% more accurate than *Euclidean-Avg*.

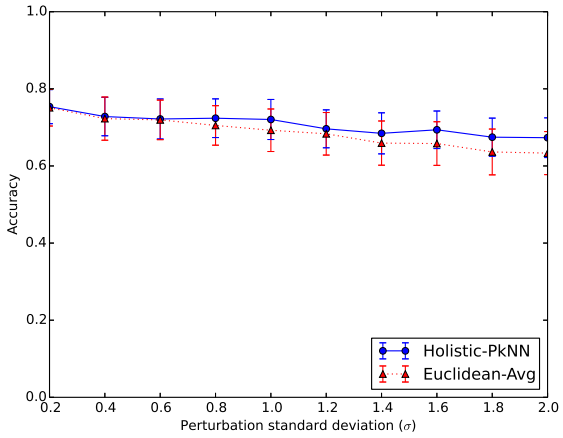


Figure 5.8: Accuracy when varying the perturbation standard deviation σ using the NN classifiers based on the Top-k-$P_{NN}(D, Q, k)$ and *Euclidean-Avg* algorithms.

5.6.4 Time performance

In this section, we evaluate the running time efficiency on a variety of settings using synthetic datasets.

Effectiveness of Pruning Strategies

The tightness of the distance bounds during the initialization of the distance partitions as detailed in Section 5.3.4 can greatly reduce the processing time, thanks to the early pruning of a large fraction of the candidate uncertain series. In the first set of experiments, we measure the ratio of candidates that cannot be pruned immediately after the initialization of their distance partitions for the *spatial*, *metric* and *exact* distance bounds. Recall that the *spatial* and *metric* distance bounds can be estimated efficiently (Section 5.3.4), while *exact* distance bounds require the evaluation

of the exact distance samples in $Dist(Q, X_i)$. We stress that the *spatial* and *metric* distance bounds lower- and upper-bound the *exact* distance bounds and the final produced result is exactly the same.

In Figure 5.9 we report the ratio of the retained uncertain series after pruning the candidates based on the distance partitions initialized using the *exact*, *metric* and *spatial* distance bounds with perturbation standard deviation $sigma = 0.2$ when varying the number of samples, m. The *exact* distance bounds are the most accurate, followed by the *metric* and then the *spatial* distance bounds. The average ratio of the retained candidates is 0.7%, 19% and 57% when the distance partitions are initialized using the *exact*, *metric* and *spatial* distance bounds, respectively. We observe a slight increase in the ratio of retained candidates as the number of samples m increases. Recall that the m samples are drawn from a normal distribution. As the number of samples m increases, the probability that at least a few samples deviate significantly from the mean value increases. Although the distance bounds are affected by outlier samples, the overall sample distribution is stable and does not depend on the number of samples.

In the next experiment, we vary the perturbation standard deviation σ and report the ratio of retained uncertain series after pruning the candidates based on their initialization of the distance partitions for the *spatial*, *metric* and *exact* distance bounds. The results are reported in Figure 5.10. The *exact* distance bounds are the most accurate, followed by the *metric* and then the *spatial* distance bounds. As the perturbation standard deviation σ increases, the pruning power of the different initializations for the distance partition is reduced. With *exact* bounds, 20% of the candidates is retained when σ approaches 1.0. On the contrary, 100% of the candidates are retained when σ approaches 0.4 and 0.8 for the *spatial* and *metric* bounds, respectively.

In Figure 5.11 we report the ratio of retained uncertain series after prun-

ing the candidates based on their initialization of the distance partitions for the *spatial*, *metric* and *exact* distance bounds with perturbation standard deviation $\sigma = 0.2$ when varying the length of the uncertain series, n. Similarly to the prior experiments, *exact* distance bounds are the most accurate, followed by the *metric* and then the *spatial* distance bounds. The series length n doesn't affect significantly the performance. We note that normally distributed samples along the candidate series result in a normally distributed distance between the candidate and the query. The same applies for other distributions, independently from the series length n that does not affect the effectiveness of the pruning strategies.

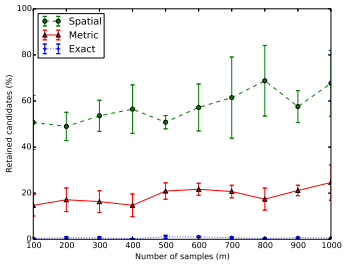


Figure 5.9: Ratio of retained candidates when varying the number of uncertain series samples m for distance partitions initialized with *spatial*, *metric* and *exact* distance bounds, respectively.

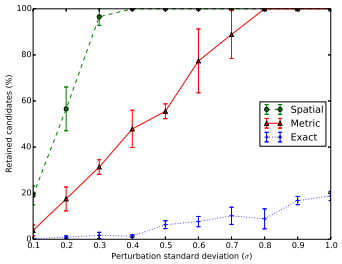


Figure 5.10: Ratio of retained candidates when varying the perturbation standard deviation σ for distance partitions initialized with *spatial*, *metric* and *exact* distance bounds, respectively.

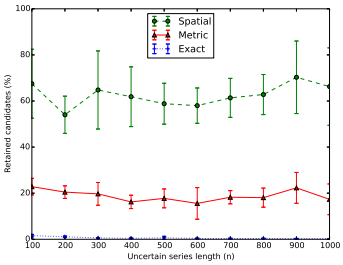


Figure 5.11: Ratio of retained candidates when varying the uncertain series length n for distance partitions initialized with *spatial*, *metric* and *exact* distance bounds, respectively.

Figure 5.12 reports the ratio of retained uncertain series using the *metric* distance bounds when pruning the candidates varying the number of pivots for different pivot selection strategies: *random*, *max-dist* and *k-means*. The *random* strategy selects k random samples X_i^l from the X_i instantiations as pivot series. The *max-dist* approach selects the k samples from the X_i instantiations that maximize their distance to k random samples from

the X_i instantiations. The *k-means* method uses the average series of the k clusters of X_i instantiations, identified using the k-means algorithm. The *k-means* strategy is the most effective, followed by the *random* and *max-dist* methods. Increasing the number of pivots reduces slightly the ratio of retained candidates for the *max-dist* and *random* methods. On the contrary, the number of pivots does not affect significantly the performance of the *k-means* method.

We report the results on time performance when pruning the candidates with the *metric* distance bounds using a linear scan over all candidates (denoted by Metric-LSCAN) and our adaptation of the M-tree index (denoted by Metric-M-tree) in Figure 5.13. The graph shows the time performance using the two techniques when varying the perturbation standard deviation σ. We observe that the M-tree index is the best performing when the perturbation standard deviation σ is lower than 0.18. For larger values of σ, the linear scan is more efficient. Recall that each leaf in the M-tree represents a distinct uncertain series. As σ increases, the regions represented by the internal nodes overlap with higher probability and the number of pruned sub-trees is reduced. The incurred cost of visiting the tree and the evaluation of the pruning conditions nullifies the benefits of using an index. In the rest of the experiments, we use the Metric-LSCAN approach.

In the next set of experiments, we compare the time performance using the different methods to initialize the distance partitions S_i to evaluate Top-k-$P_{NN}(D, Q, k)$ queries. Figure 5.14 reports the time performance when varying the number of samples m for the *spatial*, *metric* and *exact* distance bounds. The *metric* distance bounds are the best performing, followed by the *spatial* and then the *exact* distance bounds. We observe that the evaluation of Top-k-$P_{NN}(D, Q, k)$ queries using the *metric* distance bounds can be up to 18% faster than using *spatial* distance bounds. This is due to the tighter distance bounds when using the *spatial* bounds. It is

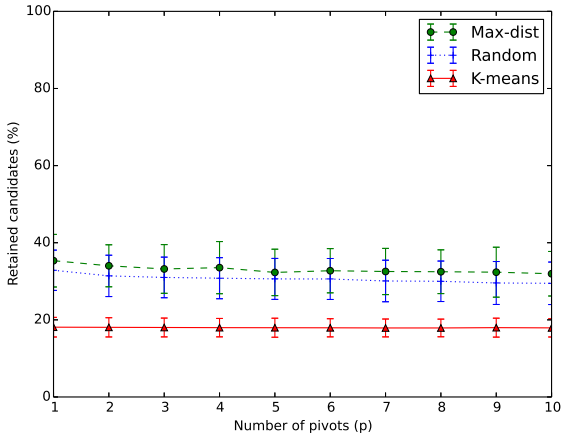


Figure 5.12: Ratio of retained candidates when varying the number of pivots for the *metric* distance bounds using different pivot selection strategies *random*, *max-dist* and *k-means*.

worth noting that the *exact* distance bounds are even tighter, but the high incurred CPU cost for the evaluation of the exact distance samples leads to inferior overall performance.

We report the time performance when varying the perturbation standard deviation σ with distance partitions initialized using *spatial*, *metric* and *exact* distance bounds in Figure 5.15. The *metric* distance bounds are the best performing, followed by the *spatial* and then the *exact* distance bounds. As the perturbation standard deviation σ increases, all methods approach the same time performance and perform nearly the same. This is due to the increasing probability of overlaps between the distance distributions. As this probability increases, the CPU cost is mainly driven by the repeated evaluation of the PNN bounds and the iterative refinements of the distance partitions.

The time performance with varying number of uncertain series N is illustrated in Figure 5.16. The distance partitions are initialized using the *spatial*, *metric* and *exact* distance bounds. Similarly to the previous

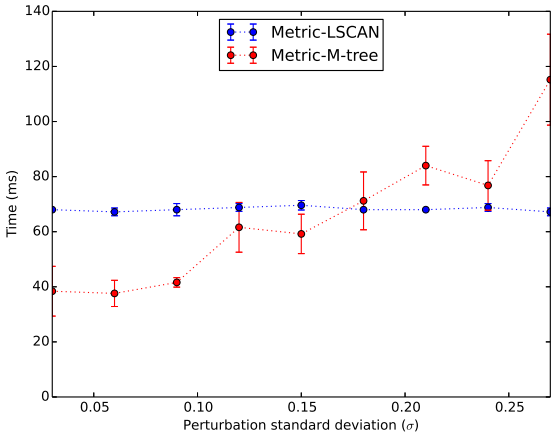


Figure 5.13: Time performance when varying perturbation standard deviation for *linear-scan* and *M-tree* techniques for distance partitions initialized using *metric* distance bounds.

experiments, the *metric* distance bounds are the best performing, followed by the *spatial* and then the *exact* distance bounds. As expected, the CPU cost increases linearly to the dataset size, N.

In conclusion, the distance partitions initialized using the *metric* distance bounds using one pivot series are the best performing. The M-tree index proved to be competitive to a linear scan when the perturbation standard deviation σ is low. However, a linear scan is to be preferred in general. In the rest of this study we initialize the distance partitions S_i using the *metric* distance bounds, initialized through a linear scan over the dataset.

Comparison to Prior Approaches

In the next series of experiments, we compare the time performance of the algorithms *Baseline* and the different versions of the *Holistic-PkNN* algorithm. We may omit the prefix *Holistic* for ease of presentation in some of the figures. Recall that the *Holistic-PkNN-Virtual* technique is our adaptation of the *Find-TopK-PNN* algorithm [16] (see Section 5.6.2

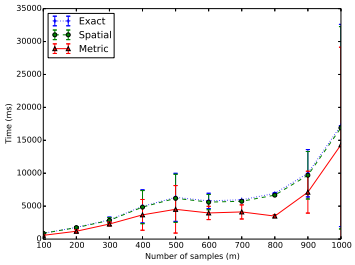


Figure 5.14: Time performance when varying the number of samples m for distance partitions initialized with *spatial*, *metric* and *exact* distance bounds, respectively.

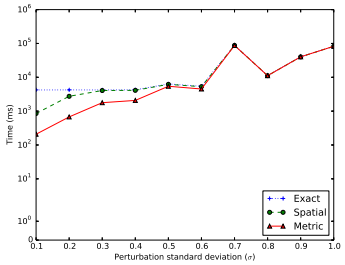


Figure 5.15: Time performance when varying the perturbation standard deviation σ for distance partitions initialized with *spatial*, *metric* and *exact* distance bounds, respectively.

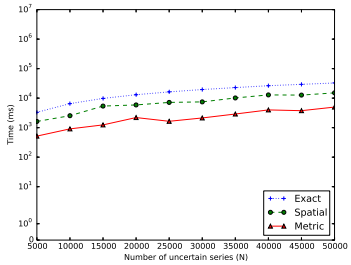


Figure 5.16: Time performance when varying the number of uncertain series N for distance partitions initialized with *spatial*, *metric* and *exact* distance bounds, respectively.

for a detailed presentation).

We report the time performance when varying the number of samples m for the *Baseline* and *Holistic-PkNN* algorithms in Figure 5.17. The graph shows that *Holistic-PkNN-HS* is the best performing, followed by *Holistic-PkNN-MS*, *Holistic-PkNN-HP* and *Holistic-PkNN-Virtual* variants of the *Holistic-PkNN* algorithm. The *Baseline* approach is the worst performing. We note that *Holistic-PkNN-HS* is up to two orders of magnitude faster then the *Holistic-PkNN-Virtual* algorithm. This is due to the larger number of iterations of the *Holistic-PkNN-Virtual* algorithm, that continuously increases the number of candidates until convergence. The larger number of iterations is associated with a larger, very CPU intensive, number of evaluations of the PNN probability bounds. The *Baseline* approach doesn't rely on pruning strategies to reduce the number of candidates. The distinct distance samples in $Dist(Q, X_i)$ are used in the pairwise comparisons in Algorithm 1. The incurred CPU cost is up to three orders of magnitude higher than the CPU cost of the algorithms in the *Holistic-PkNN* family.

In the next experiment, we assess the time performance when vary-

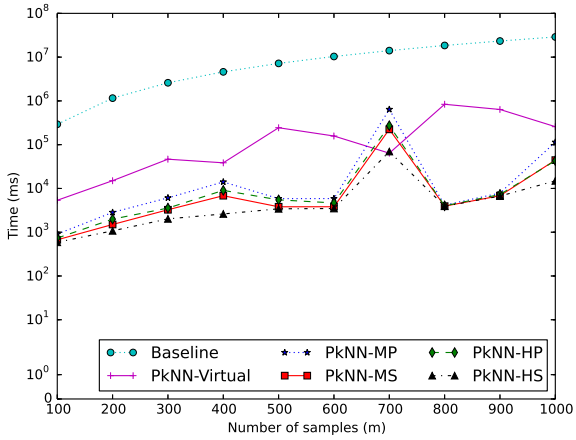


Figure 5.17: Time performance when varying number of samples m for *Baseline* and *Holistic-PkNN* algorithms.

ing the perturbation standard deviation σ for the *Baseline* and *Holistic-PkNN* algorithms. The results are reported in Figure 5.18 and show that *Holistic-PkNN-HS* is in general the best performing, followed by *Holistic-PkNN-MS*, *Holistic-PkNN-HP* and *Holistic-PkNN-Virtual* variants of the *Holistic-PkNN* algorithm. The *Baseline* approach exhibits again the worst performance. The CPU cost of the *Baseline* algorithm is not affected by the properties of the perturbation, and is constant. We observe that the *Holistic-PkNN-Virtual* algorithm is the best performing when the perturbation standard deviation σ is lower than 0.2. An in-depth analysis of the execution revealed that a lower number of candidates is retrieved by the *Holistic-PkNN-Virtual* when the perturbation is sufficiently low. The reduced number of candidates is limiting the number of evaluations of the PNN probability bounds across all iterations in contrast to the *Holistic-PkNN-HS* algorithm.

We report our results on performance when varying the number k of retrieved uncertain series for the *Baseline*, *Holistic-PkNN* and *Holistic-PkNN-Virtual* algorithms in Figure 5.19. *Holistic-PkNN* is the best per-

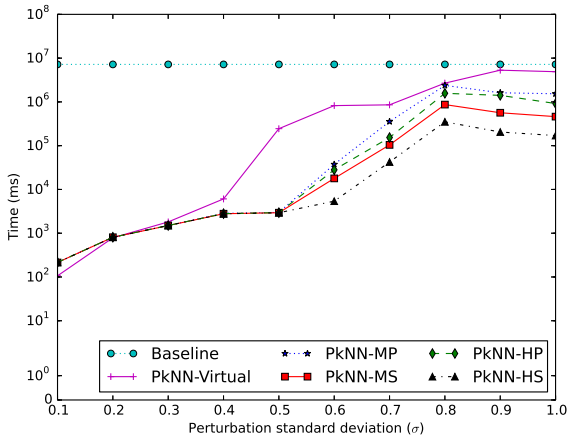


Figure 5.18: Time performance when varying the perturbation standard deviation σ for *Baseline* and *Holistic-PkNN* algorithms.

forming, followed by the *Holistic-PkNN-Virtual* algorithm and the *Baseline* method. We observe that the *Baseline* approach determines the exact PNN probability estimates for all candidates and the CPU cost incurred in the identification of the top-k result set is negligible if compared to the computational cost of evaluating the PNN estimates. The *Holistic-PkNN* algorithm is more than one order of magnitude faster than the *Holistic-PkNN-Virtual* method when k is low.

We conclude reporting the time performance when varying the uncertain series length n for the *Baseline*, *Holistic-PkNN* and *Holistic-PkNN-Virtual* algorithms in Figure 5.20. The *Holistic-PkNN* algorithm is the best performing, followed by the *Holistic-PkNN-Virtual* and *Baseline* methods. We observe that the incurred CPU time cost does not increase steadily as the series length n increases. An in-depth analysis of the execution traces revealed that, as n increases, the difference between the closest and the farthest distance samples increases. In other words, as the dimensionality (n) increases the distance samples are spreaded in a wider region. This results in a lower probability of overlaps between the distance partitions, eventu-

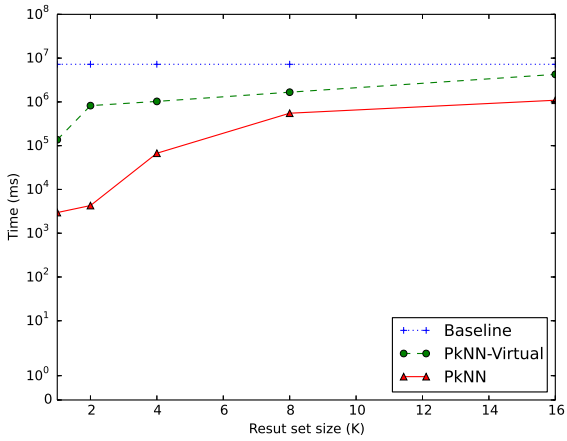


Figure 5.19: Time performance when varying the number k of retrieved uncertain series for the *Baseline*, *Holistic-PkNN* and *Holistic-PkNN-Virtual* algorithms.

ally resulting in less active candidates in X^* as reported in Figure 5.21. Fewer active candidates in X^* translate to fewer PNN probability bound estimates and fewer refinements of the distance partitions. We further note that the time required to determine the distance samples increases linearly to n. The resulting time performance is a combination of these two characteristics of the distance distributions.

In summary, the *Holistic-PkNN-Virtual* proved to be the best performing when the perturbation standard deviation is below $\sigma = 0.2$. Following the results presented in Section 5.6.4, we conclude that the *Holistic-PkNN-Virtual* algorithm combined with M-trees is expected to be the best performing when the perturbation standard deviation is sufficiently low, i.e., $\sigma = 0.2$. However, in general the *Holistic-PkNN-HS* algorithm using a linear scan to initialize the distance partitions S_i with *metric* distance bounds is to be preferred.

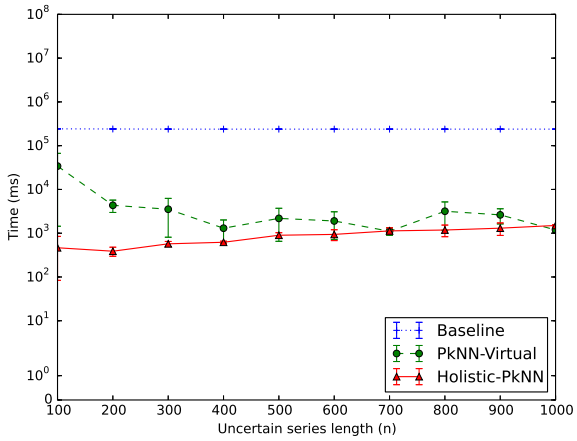


Figure 5.20: Time performance when varying the uncertain series length n for the *Baseline*, *Holistic-PkNN* and *Holistic-PkNN-Virtual* algorithms.

Scalability

Finally, we report our results on scalability. Figure 5.22 shows the time performance for different levels of perturbation standard deviation σ when varying the number of uncertain series N using the *Holistic-PkNN* algorithm. We observe that the incurred CPU cost for $N = 10000$ and perturbation standard deviation $\sigma = 0.3$ is similar to the time performance for $N = 100000$ and perturbation standard deviation $\sigma = 0.1$. The size of the dataset N and the perturbation standard deviation σ are two parameters that affect significantly the time performance of the algorithm, and even relatively small datasets are challenging when a large fraction of the distance distributions overlap (i.e., when σ is large).

Figure 5.23 shows the time performance for different configurations of the number of uncertain series samples m when varying the number of uncertain series N using the *Holistic-PkNN* algorithm. The experiments on scalability show that the *Holistic-PkNN-HS* algorithm can scale to large datasets. The number of samples m and the perturbation standard devia-

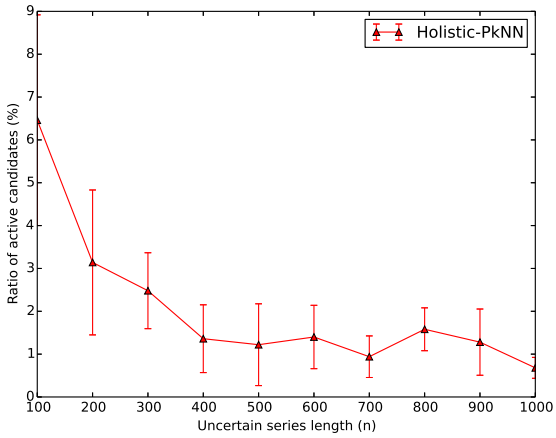


Figure 5.21: Ratio of candidates in the active set X^* when varying the uncertain series length n for the *Holistic-PkNN* algorithm.

tion σ affect significantly the CPU time performance.

5.7 Summary

Uncertain data series can be used to represent data series whose values are imprecise or inherently uncertain. In this study we formalize different models of uncertain data series, presenting and discussing their properties. We define the top-k nearest neighbor problem in uncertain data series and propose a variety of methods that unify and extend prior studies in the field in the *Holistic-PkNN* algorithm. We further investigate the adoption of pruning techniques based on spatial and metric bounds. We evaluate our proposal under a variety of settings using 45 real datasets from diverse domains and synthetic datasets. The results show that modeling uncertainty with probabilistic models can lead to more accurate results. Our proposal proved to be up to orders of magnitude more efficient than previously proposed techniques.

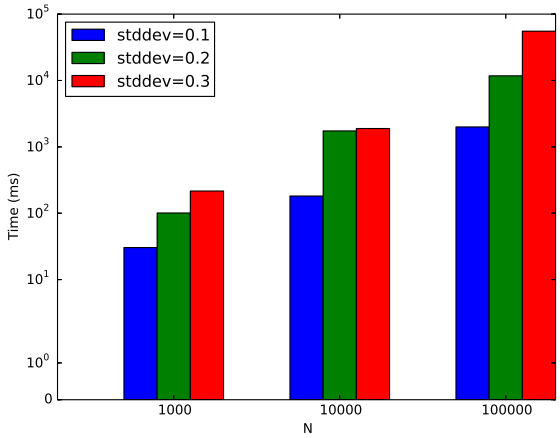


Figure 5.22: Time performance for different levels of perturbation standard deviation σ when varying the number of uncertain series N using the *Holistic-PkNN* algorithm.

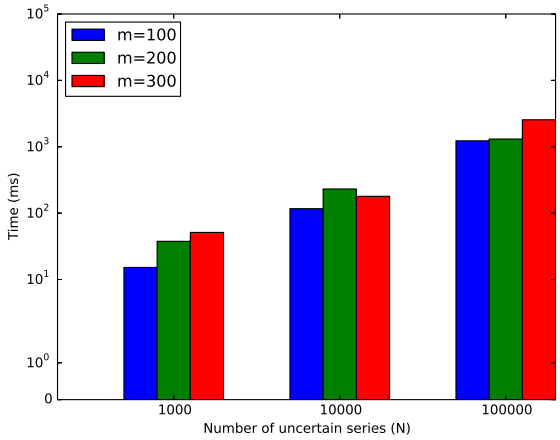


Figure 5.23: Time performance for different configurations of the number of uncertain series samples m when varying the number of uncertain series N using the *Holistic-PkNN* algorithm.

Chapter 6

Sliding Windows over Uncertain Data Streams

Uncertain data streams can have tuples with both value and existential uncertainty. A tuple has value uncertainty when it can assume multiple possible values. A tuple is existentially uncertain when the sum of the probabilities of its possible values is less than 1. A situation where existential uncertainty can arise is when applying relational operators to streams with value uncertainty. Several prior works have focused on querying and mining data streams with both value and existential uncertainty. However, none of them have studied, in depth, the implications of existential uncertainty on sliding window processing, even though it naturally arises when processing uncertain data. In this work, we study the challenges arising from existential uncertainty, more specifically the management of count-based sliding windows, which are a basic building block of stream processing applications. We extend the semantics of sliding window to define the novel concept of *uncertain sliding windows*, and provide both exact and approximate algorithms for managing windows under existential uncertainty. We also show how current state-of-the-art techniques for answering similarity join queries can be adapted to be used with uncertain sliding windows. We evaluate our proposed techniques under a variety of

configurations using real data. The results show that the algorithms used to maintain uncertain sliding windows can operate efficiently while providing a high quality approximation in query answering. In addition, we show that sort-based similarity join algorithms can perform better than index-based techniques (on 17 real datasets) when the number of possible values per tuple is low, as in many real-world applications.

In this study, we tackle three main challenges emerging from developing applications that process uncertain data streams. The first is to model existential uncertainty in order to support operator composition in the presence of value uncertainty. We address this challenge by considering existential uncertainty in our stream processing model and by extending the definition of sliding windows to take into account its uncertain boundaries. We consider this to be a first step towards developing applications via operator composition.

The second challenge is to provide an efficient implementation of an uncertain sliding window both in terms of memory space and computational time required, so that it can be used in streaming applications with stringent performance requirements. To this effect, we provide an algorithm for managing count-based sliding windows by modeling its size as a discrete random variable that has a Poisson binomial distribution, which we then use to obtain an estimate of the window size based on the current contents of the window.

The third challenge is to have streaming operators that are efficient in the presence of both value and existential uncertainty. As an example, we adapt a state-of-the-art similarity join technique to uncertain sliding windows. In addition, we introduce a simple sort-based join algorithm that is competitive in scenarios where the number of samples is small.

The rest of this chapter is organized as follows. Uncertain data streams are introduced in Section 6.1. In Section 6.2, we describe a model that

allows for efficient processing of sliding windows with uncertain data. In Section 6.3, we describe how uncertain sliding windows can be used by aggregate and join operators. In Section 6.4, we describe efficient join algorithms for uncertain data streams, including a sort-based algorithm specifically designed for similarity matching of uncertain data. Our experimental evaluation is presented in Section 6.5 and in Section 6.6 we discuss some possible extensions. Section 6.7 concludes the chapter.

6.1 Uncertain data streams

6.1.1 Preliminaries

A data stream S is a sequence of tuples s_i, where $0 \leq i \leq \eta$ and $\eta \in \mathbb{N}$ is the index of the last appended tuple. We refer to i as the index of a tuple in a stream. Without loss of generality, a tuple s_i is a $d-$dimensional real-valued point [1]. We define a subsequence of stream S as $S_{[i,j]} = \langle s_i, \ldots, s_j \rangle$. We define a count-based sliding window $W(S, w)$ as the subsequence $S_{[\eta-w+1,\eta]}$, where η is index of the most recent tuple received from stream S and $w \in \mathbb{N}$ indicates the size of the window. When not implicit from the context, we refer to data streams without uncertainty as *certain* data streams.

An uncertain data stream U is a sequence of uncertain tuples u_i, where $0 \leq i \leq \eta$ and $\eta \in \mathbb{N}$. Tuple u_i is represented by a set of l possible materializations, i.e., $u_i = \{u_{i,1}, \ldots, u_{i,l}\}$. If $|u_i| > 1$, then the tuple has value uncertainty. A sample materialization $u_{i,j} \in u_i$ occurs with a given probability $Pr(u_{i,j})$. The existential probability $Pr(u_i)$ of tuple u_i is defined as

$$Pr(u_i) = \sum_{u_{i,j} \in u_i} Pr(u_{i,j}). \tag{6.1}$$

[1] Each dimension can be considered as an attribute.

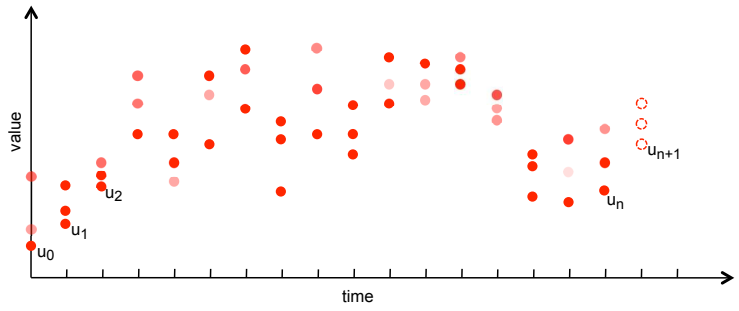


Figure 6.1: Example of an uncertain data stream, where uncertainty is modeled by repeated weighted measurements and tuples are 1-dimensional points. Weights are encoded using transparency, i.e., lighter points occur with lower probability.

Tuple u_i is said to exist in stream U if $Pr(u_i) = 1$. If $Pr(u_i,) < 1$, tuple u_i is considered existentially uncertain. Figure 6.1 shows an example of an uncertain data stream, where each tuple is represented by three weighted samples.

In the rest of this section we show that applying commonly used stream transformations to uncertain data streams can (i) introduce existential uncertainty from value uncertainty, and (ii) introduce value uncertainty from existential uncertainty.

6.1.2 From value to existential uncertainty

We use a stream operator *filter* to illustrate how value uncertainty may cause existential uncertainty. Filter operators are widely deployed to discard non-interesting data, noisy tuples, and outliers.

Given a *certain* data stream S, a filter operator $f_c(S)$ accepts an input stream S and produces an output stream T s.t. $s_i \in T$ iff s_i meets the user-defined condition c. In particular, we have $T \subseteq S$.

With uncertain data streams, a filter operator must consider that a tuple may assume multiple values. When an input tuple u_i from an uncertain data stream U gets processed, the *filter* operator $f_c(U)$ produces an output stream V. An output tuple $v_k \subseteq u_i$ s.t. the samples $u_{i,j}$ meeting the user-

defined condition c are retained, while all other samples are dropped (i.e., filtered out). For ease of exposition, we assume that tuples u_i exhibit value uncertainty only and thus $Pr(u_i) = 1$. If a subset of possible assignments for tuple u_i is filtered out, the produced output tuple v_k exhibits existential uncertainty, since $Pr(v_k) < 1$.

6.1.3 From existential to value uncertainty

As described in Chapter 1, operators that use sliding windows in their logic are influenced by existential uncertainty. This is because the sliding window boundary becomes uncertain, thus leading to uncertain output values. To illustrate this problem, we consider a *sliding-window aggregate* operator performing a summation.

Given a *certain* data stream S and a sliding window $W(S, w)$, an aggregate produces a new stream data item t_η by summing up the attribute values of the last w incoming tuples from stream S. Given that the incoming tuple is s_η, the resulting tuple t_η is defined as $t_\eta = s_\eta + ... + s_{\eta-w+1}$.

In the presence of uncertain input data, the aggregate must consider the uncertainty of sliding windows. Given an uncertain input stream U, an aggregate operator processes incoming uncertain tuples through sliding window $W(U, w)$. Assuming that there is at least one tuple u_i that is existentially uncertain ($Pr(u_i) < 1$), there is a set of possible worlds for the content of the sliding window $W(U, w)$. For example, if one tuple within the last w tuples does not exist, then we must account for it by including one more tuple from U to the window content. If there is a second tuple within the last w tuples that is existentially uncertain, then there is a window that considers the possible world with two more tuples from U's history. Note that there are multiple possible summations for the same sliding window. This means that the stream generated by the aggregate operator has value uncertainty.

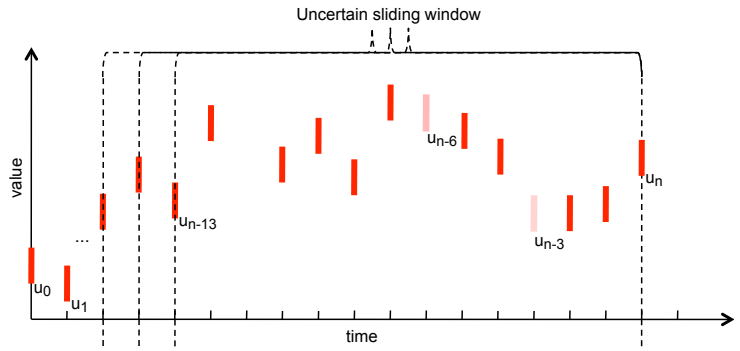


Figure 6.2: Example of an uncertain sliding window. Bounding intervals drawn using dashed lines represent the sliding window content, whereas light colored bars represent existentially uncertain tuples.

Figure 6.2 shows an example of the content of an uncertain sliding window of size 13 in an aggregate operator. We represent each tuple in the uncertain data stream as a bar, which indicates the minimum and maximum values of the tuple attribute. The window contains two tuples that are existentially uncertain ($u_{\eta-3}$ and $u_{\eta-6}$). In this example, the sliding window has four different materializations. The bounding intervals in the figure represent three different window boundaries corresponding to these materializations. This results in four different combinations of tuples whose values lead to multiple materializations.

6.2 Uncertain Sliding Windows

In this section, we formalize the semantics for count-based *uncertain sliding windows*. We stress that in past studies uncertain data streams are processed through regular sliding windows. In our study, we investigate the implications of the marriage between sliding window processing and existential uncertainty. The user-defined window size refers to the number of truly existing points according to the possible worlds semantics. Intuitively, the number of tuples actually maintained in the sliding window can

Symbol	Description
U	data stream
u_i	i_{th} tuple in U
η	index of most recent tuple in U
$W(U, w)$	sliding window over data stream U of size w
$\hat{W}(U, w)$	distribution of sliding window $W(U, w)$
$\lvert\hat{W}(U, w)\rvert$	count of existing tuples in $\hat{W}(U, w)$
α	probabilistic threshold
β	similarity threshold

Table 6.1: Symbols used in the chapter and their explanations.

overflow the user-defined window size due to the existential uncertainty of some tuples.

Uncertain sliding windows can be used as building blocks for common streaming operators, such as *joins*, as we will show later in Section 6.3.

In Table 6.1, we summarize the most important symbols used in the rest of the chapter.

6.2.1 Modeling uncertain sliding windows

Given an uncertain data stream U, a windowed stream operator processes incoming tuples through sliding window $W(U, w)$ where w is the window size. When all tuples in U are existentially certain, the sliding window boundaries are managed in a straightforward manner, i.e., when the operator inserts a new tuple into a full window, it also evicts the oldest tuple from the window.

When some tuples in U are existentially uncertain, the boundaries of the sliding window become uncertain, as shown in the example in Figure 6.2. To model this boundary, we first define $\hat{W}(U, w)$ as the subsequence of tuples existing within $W(U, w)$. This subsequence can be considered as a random variable whose sample space is the set of all possible window

materializations corresponding to $W(U, w)$. We denote this subsequence's size as $|\hat{W}(U, w)|$, which is a discrete random variable.

When a stream operator processes uncertain tuples through a sliding window of length w, the number of tuples in some materializations of the window may not reach the window length w, i.e., $Pr(|\hat{W}(U, w)| = w) < 1$. Considering the sliding window semantics and the uncertainty model with possible world semantics, more tuples from the history of U must be included into the sliding window to account for existential uncertainty. More formally, exactly w existentially certain tuples (i.e., $u_i \in U$ s.t. $Pr(u_i) = 1$) must be kept inside the sliding window. As an example, in Figure 6.2, two tuples in $W(U, w)$ are existentially uncertain. As a result, two more existentially certain tuples are included in the sliding window ($u_{\eta-14}$ and $u_{\eta-15}$). Now the window contains at least w tuples, regardless of the existence of the uncertain ones ($u_{\eta-6}$ and $u_{\eta-3}$).

Intuitively, we want to substitute the sliding window $\hat{W}(U, w)$ with $\hat{W}(U, w')$, where $w' \geq w$ represents the number of tuples kept in the window $W(U, w)$ and the following holds:

$$Pr(|\hat{W}(U, w')| = w) = 1. \tag{6.2}$$

This equation has two problems. First, each possible materialization of $\hat{W}(U, w')$ may have a different number of tuples in it. Thus, the probability that the number of tuples existing in the window is exactly w is not guaranteed to reach one. Instead, we need to make sure that each possible materialization has *at least* w tuples. We observe that with increasing values of w' the probability $Pr(|\hat{W}(U, w')| \geq w)$ approaches one. This leads to a refinement of the probabilistic condition in Equation (6.2), as follows:

$$Pr(|\hat{W}(U, w')| \geq w) = 1 \ \wedge \ w' \text{minimal}. \tag{6.3}$$

We observe that if all tuples in U are existentially uncertain, the value of w' in $\hat{W}(U, w')$ approaches the total size of U (or infinity) when Equa-

tion (6.3) must hold. Thus, our definition of an *uncertain sliding window*, denoted as $W(U, w, \alpha)$, bounds the number of tuples to be kept in a window (that is w') by introducing a probabilistic threshold α, as follows:

$$Pr(|\hat{W}(U, w')| \geq w) \geq \alpha \ \wedge \ w' \text{minimal.} \tag{6.4}$$

As the number of tuples kept in the window increases, the probability that less than w tuples exist within $\hat{W}(U, w')$ approaches to zero. When this probability reaches $1 - \alpha$, we do not need to keep any additional tuples in the window, according to Equation (6.4). Thus, α serves as a probabilistic bound that limits w'.

We note that Eq. 6.4 can be used to define a sliding window whose number of tuples is w with a known level of confidence, α. Similar formulations of probabilistic thresholds to bound uncertainty have been proposed in prior studied, such as for range queries and nearest neighbor searches in [26]. In the following, we will consider this definition to define the probabilistic bounds of uncertain sliding windows.

6.2.2 Processing uncertain sliding windows

Given a *certain* data stream S and sliding window $W(S, w)$, new tuples are processed as follows. Whenever a new tuple s_i comes in, (i) the operator *adds* s_i to the content of sliding window W and (ii) if $|W| > w$, then the operator *evicts* tuple s_j from window W, where $\forall_{s_k \in W} j \leq k$, i.e., s_j is the oldest tuple in W. The eviction policy is deterministic. Once W reaches the desired user-defined length w, the operator evicts exactly one tuple every time a new tuple comes in.

With uncertain data streams, we substitute regular sliding windows with uncertain sliding windows. Given an uncertain data stream U, an operator processes an uncertain sliding window $W(U, w, \alpha)$, as defined in

Algorithm 6. The key point here is the eviction procedure, which may evict more than one tuple at a time.

Algorithm 6 *uncert-evict*

Input: U, w, α

Output: $W(U, w, \alpha)$

```
1: W(U, w, α) ← ∅
2: loop
3:    if new tuple u from U then
4:       W(U, w, α) ← W(U, w, α) ∪ {u}
5:       while Pr(|Ŵ(U, w' − 1)| ≥ w) ≥ α do
6:          W(U, w, α) ← W(U, w, α) \ {u'} s.t. u' is the oldest tuple in W(U, w, α)
7:       end while
8:    end if
9: end loop
```

The algorithm evaluates the probabilistic condition defined in Equation (6.4) on the window content without the oldest tuple, that is using $w' - 1$ rather than w' in $|\hat{W}(U, w')|$, where w' is the number of tuples currently kept in the window $W(U, w, \alpha)$. If the condition is met, the algorithm evicts the oldest tuple, since the window has sufficient content without it. The test is iterated, evicting as many tuples as possible. This ensures that the resulting window is minimal.

To evaluate $Pr(|\hat{W}(U, w'-1)| \geq w)$ in Algorithm 6, we need a model for the random variable $|\hat{W}(U, w' - 1)|$ in terms of its cumulative distribution function (CDF):

$$Pr(|\hat{W}(U, w' - 1)| \geq w) = 1 - Pr(|\hat{W}(U, w' - 1)| \leq w - 1). \qquad (6.5)$$

The random variable $|\hat{W}(U, w' - 1)|$ can be seen as the sum of independent Bernoulli trials, where the success probabilities of the trials are mapped to the existential probabilities of the tuples. Formally, let I_i be a random indicator associated with tuple u_i of stream U, where $0 \leq i \leq \eta$

and η is the most recent tuple index. We have

$$I_i \sim Bernoulli(Pr(u_i)), \tag{6.6}$$

where $Pr(u_i)$ is the existential probability of tuple u_i as defined in Equation (6.1). As a simplifying assumption, we assume that random indicators I_i are independent. The distribution of $|\hat{W}(U, w'-1)|$ is known as *Poisson-binomial* and is defined as follows:

$$|\hat{W}(U, w'-1)| = \sum_{i=\eta-w'+2}^{\eta} I_i. \tag{6.7}$$

In some real-world scenarios existential probabilities $Pr(u_i)$ may not be independent, and could be seen as observations from an unknown Markovian process. For example, bursts of missing tuples can be described using this model. However, many times, stream operators don't have direct access to tuple correlation information [77] and process new tuples independently as they come in. In this work, we assume that windowed operators consider each tuple independently, and, as such, window sizes can be modeled as a Poisson-binomial distribution. The Poisson-binomial distribution has been used for modeling purposes with similar assumptions in reliability theory and fault tolerance [59] as well as in many other application areas [39].

In the subsequent sections, we describe algorithms and efficient online approximation schemes to compute the CDF of $|\hat{W}(U, w')|$.

6.2.3 The Poisson-binomial distribution

We first look at computing the exact CDF. Let $I_1, \cdots, I_n$ be n independent Bernoulli random variables with success probabilities $p_1, \cdots, p_n$. Then the random variable $N = \sum_{i=1}^{n} I_i$ is *Poisson-binomial* distributed. The

probability mass function (PMF) $Pr(N = k)$ is defined as:

$$Pr(N = k) = \sum_{A \in F_k} \prod_{i \in A} p_i \prod_{i \in A^c} (1 - p_i), \tag{6.8}$$

where F_k is the set of all subsets of k integers that can be selected from $\{1, \cdots, n\}$ and $A^c = \{1, \cdots, n\} \backslash A$. The CDF $Pr(N \leq k)$ is defined as follows:

$$Pr(N \leq k) = \sum_{i=0}^{k} Pr(N = i). \tag{6.9}$$

Since F_k in Equation (6.8) contains $\binom{n}{k} = n!/((n-k)! \cdot k!)$ elements, its enumeration becomes unfeasible as n increases. Hence, we need efficient techniques for computing the CDF of a *Poisson-binomial* random variable.

We consider the *RF1* recursive formulation, as reviewed in [47], to compute the exact PMF $Pr(N = k)$. Given $X_j = \sum_{i=1}^{j} I_i$, $Pr(N = k) = Pr(X_n = k)$ can be reformulated using the following decomposition:

$$Pr(X_j = l) = (1 - p_j) \cdot Pr(X_{j-1} = l) \ + p_j \cdot Pr(X_{j-1} = l - 1), \tag{6.10}$$

with boundary conditions $\forall_{k \geq l > 0}, Pr(X_0 = l) = 0$, and $\forall_{n \geq j \geq 0}, Pr(X_j = 0) = \prod_{i=1}^{j}(1 - p_i)$. If the jth Bernoulli trial is a success, we need $l - 1$ successes from the remaining $l - 1$ trials to reach l successes in total. Otherwise, we need l successes from the remaining trials.

The *RF1* algorithm can be implemented efficiently by determining the values $M_{j,l} = Pr(X_j = l)$ of matrix M in a bottom-up manner. Similarly, one can compute the CDF $Pr(N \leq k)$ by summing up the relevant cells of the matrix M, that is $Pr(N \leq k) = \sum_{l=0}^{k} M_{n,l}$.

More efficient exact algorithms (as reported in Section 2.2) have computational time cost of $O(n)$, where n is the number of tuples currently maintained in the sliding window (where $n >> k$). However they remain computationally expensive, given that the CDF must be evaluated several times within Algorithm 6. Experiments in Section 6.5.2 show that the loss

in accuracy due to the approximated estimations of the Poisson-binomial distribution CDF is negligible. We use *RF1* as a baseline to assess the performance of approximated schemes, which are briefly reviewed in the rest of this section.

6.2.4 Efficient approximations of the Poisson-binomial distribution

Hong [47] reviews some approximations for the *Poisson-binomial* distribution N, namely *Poisson*, *normal*, and *refined normal*. These approximations are obtained by combining the *Poisson* and *Normal* distributions with statistics such as mean (μ), standard deviation (σ), and skewness (γ). These statistics are defined as follows:

$$\mu = E(N) = sum_{\mu}, \tag{6.11}$$

$$\sigma = \sqrt{Var(N)} = \sqrt{sum_{\sigma}}, \tag{6.12}$$

$$\gamma = Skewness(N) = \frac{1}{\sigma^3} sum_{\gamma}, \tag{6.13}$$

where $sum_{\mu} = \sum_{i=1}^{n} p_i$, $sum_{\sigma} = \sum_{i=1}^{n} p_i \cdot (1 - p_i)$ and $sum_{\gamma} = \sum_{i=1}^{n} p_i \cdot (1 - p_i) \cdot (1 - 2p_i)$.

As described in Sections 6.2.2 and 6.2.3, we use the *Poisson-binomial* distribution to model $|\hat{W}(U, w')|$. Whenever an operator appends new tuples or evicts old tuples from sliding window $W(U, w, \alpha)$, this distribution changes. We observe that statistics μ, σ, and γ can be efficiently maintained over time by adding and removing components from the sums sum_{μ}, sum_{σ}, and sum_{γ} at the cost of simple additions and subtractions. In particular, when a new tuple is appended to the stream the computational time cost of updating these statistics is $O(k)$ where k is the number of evicted tuples. This is a key characteristic of these approximations, which allows their efficient use in streaming algorithms.

For completeness, we briefly cover these approximations [47]:

Poisson Approximation . The *Poisson-binomial* distribution is approximated with the *Poisson* distribution as $N \approx Poisson(\mu)$. Consequently,

$$Pr(N \leq k) \approx \sum_{i=1}^{k} \frac{\mu^k \exp(-\mu)}{k!}. \tag{6.14}$$

Normal Approximation . The *Poisson-binomial* distribution is approximated with the *Normal* distribution, thanks to the central limit theorem, as follows:

$$Pr(N \leq k) \approx \Phi\left(\frac{k + 0.5 - \mu}{\sigma}\right), \tag{6.15}$$

where $\Phi(x)$ is the CDF of the *standard normal* distribution.

Refined Normal Approximation . The *Poisson-binomial* distribution is approximated again via the *Normal* distribution, but this time the skewness is taken into account to improve the approximation accuracy. The CDF for the refined normal approximation is given as follows:

$$Pr(N \leq k) \approx G\left(\frac{k + 0.5 - \mu}{\sigma}\right), \tag{6.16}$$

where

$$G(x) = \Phi(x) + \frac{\gamma(1 - x^2)\phi(x)}{6}, \tag{6.17}$$

where $\Phi(x)$ and $\phi(x)$ are, respectively, the PDF and the CDF of the *standard normal* distribution.

6.3 Adapting stream operators to handle data uncertainty

Windowed stream operators reviewed in Section 2.2 do support uncertain data streams. However, they operate using sliding windows as defined over regular data streams. In this section, we discuss how they can be adapted

to use *uncertain sliding windows*, investigating the implications on operator semantics. As a driving example, we consider the problem of answering similarity join queries over uncertain data streams [61].

The similarity join operator correlates similar tuples from two input data streams. When the operator receives a new tuple, it evaluates if the tuple is similar to any of the other tuples residing in the sliding window of the opposing stream. Similarity joins are used in many applications, including detection of duplicates in web pages, data integration, and pattern recognition.

More formally, the similarity join between two certain data streams S and T is denoted by $S \bowtie_{\epsilon,w} T$. Two tuples $s_i \in S$ and $t_j \in T$ are similar if their distance is less than or equal to the user-defined distance threshold ϵ. Tuples from S and T are maintained by sliding windows $W(S, w)$ and $W(T, w)$. Whenever the similarity join operator receives a new tuple s_i from stream S, it appends the following sequence of tuples T' to the output stream:

$$T' = \{(s_i, t_j) \,|\, Dist(s_i, t_j) \leq \epsilon \wedge t_j \in W(T, w)\}, \tag{6.18}$$

where t_j is any tuple in $W(T, w)$ that meets the similarity condition. New tuples received from stream T are processed similarly. Figure 6.3 shows an example of a similarity join operator.

The similarity join operator between uncertain data streams U and V is denoted by $U \bowtie_{\epsilon,w,\alpha,\beta} V$, where ϵ and w are the match distance threshold and the sliding window size, respectively. Parameters α and β are the *probabilistic sliding window bound* and the *match probability threshold*, respectively. Given an uncertain sliding window $W(V, w, \alpha)$, whenever a new point $u_i \in U$ comes in, the join operator appends to the output stream the sequence of uncertain points V' defined as follows:

$$V' = \{(u_i, v_j)_{\bowtie} \text{ s.t. } v_j \in W(V, w, \alpha) \wedge Pr(match(u_i, v_j)) \geq \beta\}, \tag{6.19}$$

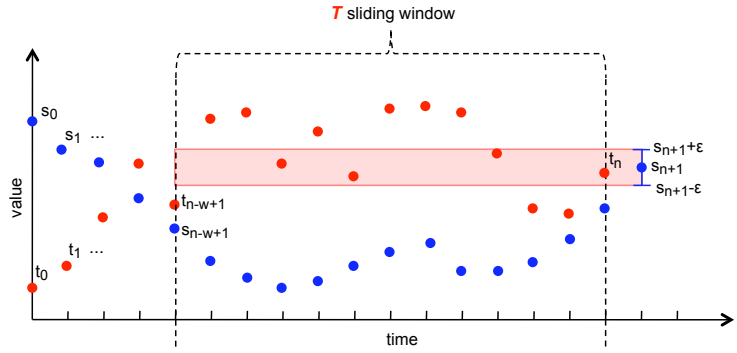


Figure 6.3: Example of a similarity join between certain data streams. Interval bar displays tuples in $W(T, w)$ that are similar to $s_{\eta+1}$ based on the distance threshold ϵ. Blue (dark) and red (light) dots represent the values of the two streams to be joined.

where v_j is any tuple in $W(V, w, \alpha)$ that meets the similarity condition $match(u_i, v_j)$ with sufficient probability.

The operator constructs the candidate output tuple $(u_i, v_j)_{\bowtie}$ by pairing all matching samples $(u_{i,k}, v_{j,l})$ as:

$$(u_i, v_j)_{\bowtie} = \{(u_{i,k}, v_{j,l}) \text{ s.t. } dist(u_{i,k}, v_{j,l}) \leq \epsilon\}. \tag{6.20}$$

To evaluate the match probability, we first evaluate if v_j is existentially certain. If so, then the match probability $Pr(match(u_i, v_j))$ is equal to the probability of the matching samples, namely $Pr(match_s(u_i, v_j))$, which is calculated as follows:

$$Pr(match_s(u_i, v_j)) = \sum_{(u_{i,k}, v_{j,l}) \in (u_i, t_j)_{\bowtie}} Pr(u_{i,k}) \cdot Pr(v_{j,l}). \tag{6.21}$$

When tuple v_j is existentially uncertain, then the match probability is computed as follows:

$$Pr(match(u_i, v_j)) = Pr(v_j \in \hat{W}_{[w]}(V, w') \wedge match_s(u_i, v_j)), \tag{6.22}$$

where $\hat{W}_{[w]}(V, w')$ is the subsequence of *most recent* w tuples within $\hat{W}(V, w')$. This leads to the following:

$$\begin{aligned} Pr(match(u_i, v_j)) = & Pr(v_j \in \hat{W}_{[w]}(V, w')) \cdot \\ & Pr(match_s(u_i, v_j) \mid v_j \in \hat{W}_{[w]}(V, w')). \end{aligned} \tag{6.23}$$

With the simplifying assumption that existential uncertainty and tuple values are independent, we have:

$$\begin{aligned} Pr(match(u_i, v_j)) =& Pr(v_j \in \hat{W}_{[w]}(V, w')) \cdot \\ & Pr(match_s(u_i, v_j))/Pr(v_j). \end{aligned} \tag{6.24}$$

In Equation (6.24), tuple v_j exists within $\hat{W}_{[w]}(V, w')$ iff it exists in V and less than w tuples exist within the sequence of tuples $v_{j+1}, ..., v_\eta$ that are more recent than v_j. Formally, we have:

$$Pr(v_j \in \hat{W}_{[w]}(V, w')) = Pr(v_j) \cdot Pr(|\hat{W}(V, \eta - j)| \leq w - 1), \tag{6.25}$$

where $\eta - j$ is the number of tuples in the window that are more recent than v_j. Finally, we have:

$$\begin{aligned} Pr(match(u_i, v_j)) =& Pr(|\hat{W}(V, \eta - j)| \leq w - 1) \cdot \\ & Pr(match_s(u_i, v_j)) \end{aligned} \tag{6.26}$$

Note that $Pr(|\hat{W}(V, \eta - j)| \leq w - 1)$ is the CDF of the *Poisson-binomial* distribution. Efficient methods for its evaluation have been discussed in Section 6.2.3.

6.4 Efficient similarity join processing

The performance of similarity joins using uncertain sliding windows can be improved by combining the probabilistic thresholds on the window size and on the match probability. We present a novel upper-bound of the match probability based on this idea. Besides, we discuss an adaptation of state-of-the-art similarity join methods [61] to uncertain sliding windows. Finally, we conclude presenting a simple yet effective sort-based similarity join algorithm that can be competitive in real-world scenarios.

6.4.1 Upper-bounding the match probability

As described in Section 6.3, we denote a similarity join operator for uncertain data streams U and V as $U \bowtie_{\epsilon,w,\alpha,\beta} V$, where ϵ is the match distance threshold, w is the sliding window size, α is the probabilistic threshold on the sliding window bound, and β is the match probability threshold. Whenever the operator receives a new tuple $v \in V$, it matches v against the uncertain sliding window $W(U, w, \alpha)$. If a matching pair exists with probability higher than or equal to β, the operator appends the tuple to its output stream.

We observe that if α approaches 1 and all tuples in U exhibit existential uncertainty, then the probability that the oldest tuple in sliding window $W(U, w, \alpha)$ exists in a materialization of the window approaches to zero:

$$\lim_{\alpha \to 1} Pr(u_{\eta - w' + 1} \in \hat{W}(U, w')) = 0. \tag{6.27}$$

From Equation (6.19), we conclude that $W(U, w, \alpha)$ tends to be oversized if β is large, since the older tuples in the window are not likely to produce any matches with high probability. This motivates a revision of the eviction policy as presented in Algorithm 6 for maintaining *uncertain sliding windows* such that it also takes β into account.

From Equation (6.26), we have $Pr(|\hat{W}(U, w' - 1)| < w)$ as an upper-bound for the match probability $Pr(match(v, u'))$, where u' is the oldest tuple in $W(U, w, \alpha, \beta)$ and $v \in V$ is the tuple we are currently processing against the window defined on U. As a result, $Pr(|\hat{W}(U, w' - 1)| < w) < \beta$ can be used as a secondary eviction condition for discarding tuples from the window, in place of $Pr(match(v, u')) < \beta$. Algorithm 7 shows the updated window eviction policy. This policy results in smaller uncertain sliding windows and an overall performance improvement.

In Algorithm 7, we use the following derivation to bring the eviction

Algorithm 7 *uncert-evict-beta*

Input: U, w, α, β

Output: $W(U, w, \alpha, \beta)$

1: $W(U, w, \alpha, \beta) \leftarrow \emptyset$
2: **loop**
3: **if** new tuple u from U **then**
4: $W(U, w, \alpha, \beta) \leftarrow W(U, w, \alpha, \beta) \cup \{u\}$
5: **while** $Pr(|\hat{W}(U, w' - 1)| \geq w) \geq min(\alpha, 1 - \beta)$ **do**
6: $W(U, w, \alpha, \beta) \leftarrow W(U, w, \alpha, \beta) \setminus \{u'\}$ s.t. u' is the oldest tuple in $W(U, w, \alpha, \beta)$
7: **end while**
8: **end if**
9: **end loop**

conditions into the same form and avoid repeated computation:

$$\begin{aligned} Pr(|\hat{W}(U, w' - 1)| < w) &= 1 - Pr(|\hat{W}(U, w' - 1)| \geq w) \\ Pr(|\hat{W}(U, w' - 1)| < w) < \beta &\equiv Pr(|\hat{W}(U, w' - 1)| \geq w) \geq 1 - \beta. \end{aligned} \tag{6.28}$$

If the oldest tuple in the uncertain window exist in materializations of the window among the first w tuples with insufficient probability, then it cannot result in a match with tuples from the opposing stream. And thus, it can be discarded from the window. β serves as a lower bound for the aforementioned sufficient probability. Note that in contrast to Algorithm 6, here we consider the α and β probabilistic constraints together, using a single formula (see Algorithm 7, line 5).

6.4.2 Pruning the similarity search space

In the following, we present different strategies to prune the search space.

Index-based pruning

Lian et al. [61] propose pruning methods for similarity join operators that process value-uncertain data streams by creating bounding regions based on the samples available in each tuple. In their method, uncertain tuples

u_i are summarized by hyper-spherical bounding regions o_i. Hypersphere o_i for tuple u_i is an approximated minimum enclosing ball of a subset of its samples. Bounding regions o_i are then indexed in a grid index that reflects the sliding window content.

A grid index is a spatial index data structure that partitions the space into a regular grid. An object to be indexed is associated to the partition in the grid whose region overlaps with the spatial coordinates of the object. A search in the grid index identifies the partitions that overlap with the search region and returns the objects associated with the matching partitions.

In the context of a spatial index, a grid (a.k.a. "mesh", also "global grid" if it covers the entire surface of the globe) is a regular tessellation of a manifold or 2-D surface that divides it into a series of contiguous cells, which can then be assigned unique identifiers and used for spatial indexing purposes. A wide variety of such grids have been proposed or are currently in use, including grids based on "square" or "rectangular" cells, triangular grids or meshes, hexagonal grids and grids based on diamond-shaped cells.

Given uncertain input streams U and V, two grid indexes G_U and G_V are maintained over time. Whenever a new tuple u_i comes in, the operator matches it against the tuples indexed in G_V. The algorithm safely prunes tuples v_j s.t. $Dist(o_i, o_j) > \epsilon$ since they cannot produce any match. The operator then processes the retained tuples as in Equations (6.20) and (6.21) to produce output matches.

A grid index is used to quickly discard a large fraction of candidate tuples. The effectiveness of grid indexing depends on the sparseness of the data. If all pairs of tuple samples are, on average, far away from each other, the bounding regions tend to be distant and the pruning strategy works well. Conversely, when at least one pair of samples are close by, then the pruning is ineffective.

Multidimensional data is supported in a straightforward manner for low

number of dimensions [61].

Although the methods proposed by Lian et al. have not been designed to be used with uncertain sliding windows, they can be adapted into the similarity join operator as presented in Section 6.3. In particular, uncertain sliding windows are used instead of regular sliding windows and candidate matches are also filtered according to the upper-bound match probability presented above (Section 6.4.1). In the rest of the chapter, we refer to our adaptation of methods in [61] as *Index-Match*.

Sort-based pruning

As an alternative to spatial pruning based on a grid index, we propose a simple yet effective pruning strategy based on sorting, called *Sort-Match*. The key advantage of sort-join algorithms with uncertain data is that they are less sensitive to the presence of one or only a few matching tuple samples for a given tuple pair.

The *Sort-Match* algorithm relies on red-black trees. A red-black tree is a binary search tree with one extra attribute for each node: the color, which is either red or black. The assigned colors satisfy certain properties that force the tree to be balanced. When new nodes are inserted or removed in the tree, the tree nodes are rearranged to satisfy the conditions. Redblack trees offer worst-case guarantees for insertion time, deletion time, and search time.

Whenever the join operator receives a new tuple $u_i \in U$, it inserts the tuple into $W(U, w, \alpha, \beta)$ and inserts the tuple samples $u_{i,k} \in u_i$ into a red-black tree RB_U. When the operator evicts tuple u_i from $W(U, w, \alpha, \beta)$, it removes the tuple samples $u_{i,k} \in u_i$ from RB_U.

By maintaining one red-black tree per sliding window, the join operator can efficiently identify which tuples in the sliding window are a match to the incoming tuple. Whenever the operator receives tuple $u_i \in U$, it searches

the red-black tree RB_V of the opposing stream for all tuples with values in the interval $[u_{i,j} - \epsilon, u_{i,j} + \epsilon]$, for each sample $u_{i,j} \in u_i$. Note that the samples in RB_V represent the content of sliding window $W(V, w, \alpha, \beta)$. Thus, all matching samples lie between search interval bounds. Once all samples are identified, the operator groups the samples by their tuple indices. After that, the operator computes the matching probability of each tuple and evaluates if it satisfies the β condition, as discussed in Section 6.3. The operator outputs all tuples satisfying the distance and probabilistic constraints.

The *Sort-Match* algorithm cannot be easily adapted to multi-dimensional data. One can overcome this limitation by using linear mapping transformations such as the z-curve or the Hilbert space filling curve [67].

6.5 Experimental evaluation

In this section, we compare how well the various approximations work for modeling uncertain sliding windows under different settings, in terms of both accuracy and performance. Furthermore, we experimentally compare the efficiency of different pruning approaches for implementing a similarity join operator that processes data streams with value and existential uncertainties.

We implemented all techniques in C++, and ran the experiments on a Linux machine equipped with an Intel Xeon 2.13GHz processor and 4GB of RAM. For all results, we report the averages of the measurements obtained from 15 independent runs, as well as the 95% confidence intervals.

For all experiments, we use the parameter configurations described in Table 6.2. When not explicitly stated, we use the default configuration value (shown in bold).

Parameter	Range
No. of samples per tuple	$[5, ..., \mathbf{10}, ..., 50]$
Sliding window size (w)	$[100, ..., \mathbf{500}, ..., 1000]$
Existential uncert. σ	$[0.025, ..., \mathbf{0.1}, ..., 0.25]$
Value uncert. σ	$[0.1, ..., \mathbf{0.5}, ..., 1]$
Stream length	**2000**
α probabilistic threshold	$[0.5, ..., \mathbf{0.95}, ..., 1]$
β probabilistic threshold	$[0.1, ..., \mathbf{0.5}, ..., 0.9]$
ϵ distance threshold	selectivity close to 0.05%

Table 6.2: Experiment parameter configuration ranges. Default values are indicated in bold.

6.5.1 Datasets

In our experiments, we generate uncertain data streams by using time series datasets that contain certain tuples (i.e., one exact value per tuple). We introduce uncertainty through perturbation, similar to prior work [61, 95, 11, 79, 31]. We introduce value uncertainty by considering *uniform*, *normal*, and *exponential* error distributions with zero mean and varying standard deviation within $[0.1, 1.0]$. We introduce existential uncertainty by sampling from *uniform*, *normal*, and *exponential* distributions with varying standard deviation within $[0, 0.25]$. Since existential uncertainty may range within interval $(0, 1)$, we restrict these distributions to this range[2]. Intuitively, the higher the standard deviation, the higher the probability of having tuples with low probability of existence. Samples outside the required range are discarded (rejection sampling).

We use 17 real time series datasets from the UCR classification [1], which represent a wide range of application domains. These are 50words, Adiac, Beef, CBF, Coffee, ECG200, FISH, FaceAll, FaceFour, Gun_Point, Lighting2, Lighting7, OSULeaf, OliveOil, SwedishLeaf, Trace, and syn-

[2]The uniform distribution over $[0, x]$ has a fixed standard deviation that is only dependent on x. To vary the standard deviation, we adapt the value of x (for $\sigma = 0.25$, $x \approx 0.87$).

thetic_control. We generate streams by sampling random subsequences from all datasets. By sampling subsequences we capture the temporal correlation that may appear across neighboring points.

6.5.2 Poisson-binomial distribution approximations

In this section, we compare how the different approximations of the Poisson-binomial distribution (Section 6.2.4) can affect the content of the uncertain sliding window. These experiments only consider the existential uncertainty of the tuples, since their results do not depend on the actual tuple values.

Accuracy

This experiment evaluates the accuracy of the three approximations of the Poisson-binomial distribution, namely Poisson, Normal, and Refined Normal. This helps us to evaluate the error that each approximation can yield when calculating the CDF in Equation 6.26.

We measure the accuracy in terms of the Root Mean Square Error (RMSE), as follows:

$$RMSE(n) = \sqrt{\frac{\sum_{k=0}^{n-1}(cdf_N(k) - cdf'_N(k))^2}{n}}, \tag{6.29}$$

where n is the number of Bernoulli random variables in the Poisson-binomial distribution $N = \sum_{i=1}^{n} X_i$, and $cdf_N(k)$, $cdf'_N(k)$ are, respectively, the exact and approximated CDFs of N. Note that the value of n represents the number of tuples kept in the window (w'), and $cdf_N(k)$ is proportional to the probability that the $k+1^{th}$ most recent tuple (say u_i, where $i = \eta - k$) exists in a window of size w, i.e., $Pr(u_i \in \hat{W}_{[w]}(U, w'))$.

Figure 6.4 shows the RMSE results (y-axis) when applying the different approximations for different window sizes (x-axis). Each graph displays the

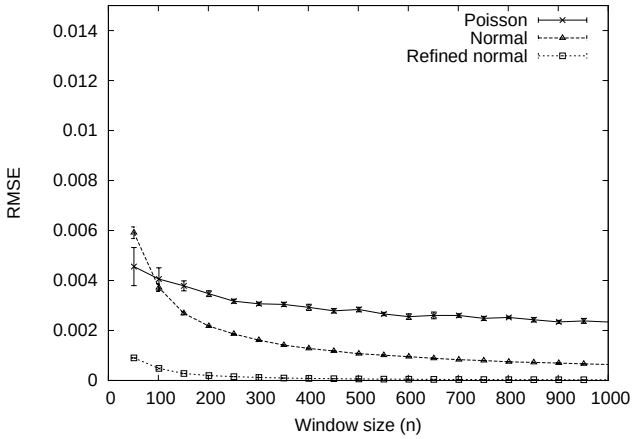


Figure 6.4: RMSE of different Poisson-binomial approximations for different window sizes and with existential uncertainty distribution standard deviation set to 0.1 (Normal distribution). The approximation with lowest error is the Refined Normal, independent of the distribution used for assigning tuple existential uncertainties. The figure also shows the low precision of the approximations for small window sizes.

RMSE results when sampling the existential uncertainty values for each tuple from the normal distribution. Results for uniform and exponential distributions are very similar, and omitted for brevity. The graph also shows the confidence interval for each measurement.

From Figure 6.4, we can see that the Refined Normal approximation provides the lowest RMSE independent of the distribution used to assign existential uncertainty values. We also notice that all approximations exhibit lower quality when the window size is small ($w < 100$). This is expected behavior according to the central limit theorem. In conclusion, the exact computation of the CDF (RF1) should be preferred if the window size (w) is below 100, otherwise the Refined Normal approximation provides the best accuracy compromise (RMSE < 0.002).

Performance

This experiment compares the performance of the different methods for obtaining the Poisson-binomial CDF. We evaluate the computational cost

of the exact algorithm (RF1) and the three approximations (Poisson, Normal, and Refined Normal). Computing the CDF efficiently is critical for a performant implementation of uncertain sliding windows. This is because there are multiple CDF computations on the critical path of the operator logic.

Figure 6.5 shows the time consumed per CDF computation (y-axis) under different window sizes (x-axis). The figure shows the results when we sample the tuple existential uncertainty values from a uniform distribution with standard deviation of 0.1. We observe that while the time required by the RF1 algorithm increases quadratically as window sizes increase, the time consumed by approximated schemes increase linearly. The Poisson approximation is the most computationally intensive among the approximations. The time consumed by the Normal and the Refined Normal are almost indistinguishable from each other. Note that the time consumed for RF1 is small for small window sizes, which indicates that an exact CDF solution can be used for small windows ($w < 100$) to achieve accurate probability computations with low performance cost. This is especially important when considering that for small windows, approximation techniques provide poor accuracy (Figure 6.4). Similar trends have been obtained when using different statistical distributions (normal and exponential) and different standard deviations for the existential uncertainty. We omit these results for brevity.

We observe that all methods require an absolute time below 3 milliseconds to evaluate the CDF function for window sizes up to 1000. However, the data throughput supported by each technique varies considerably. For example, the Refined Normal method, when compared to RF1, provides nearly 100 times better performance.

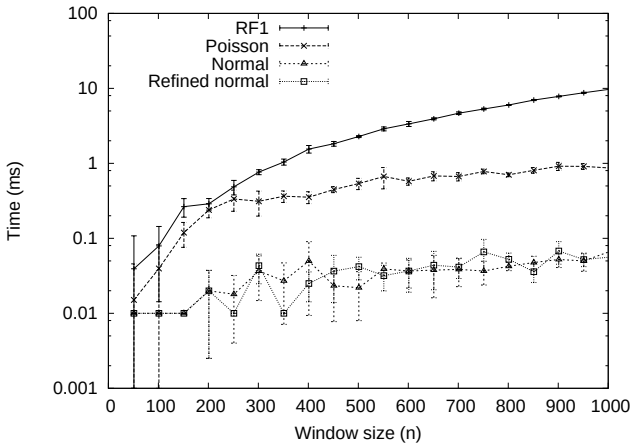


Figure 6.5: Time consumed by each CDF computation under different window sizes. Refined Normal and Normal approximations provide the lowest cost.

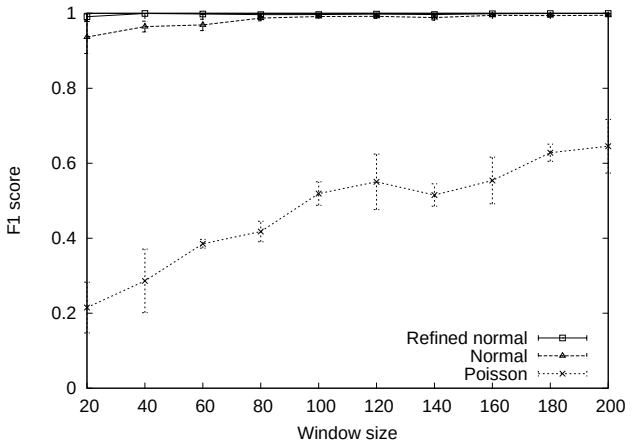


Figure 6.6: F1 score for the similarity join operator when comparing the use of CDF approximations. Join using Normal & Refined Normal approximations provide results very similar to an exact solution.

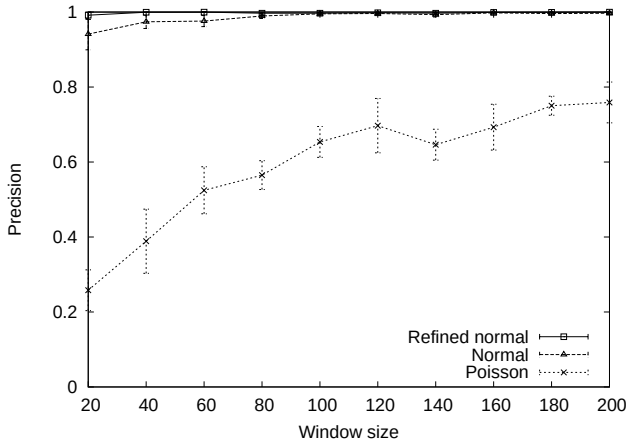


Figure 6.7: Precision for the similarity join operator when using CDF approximations.

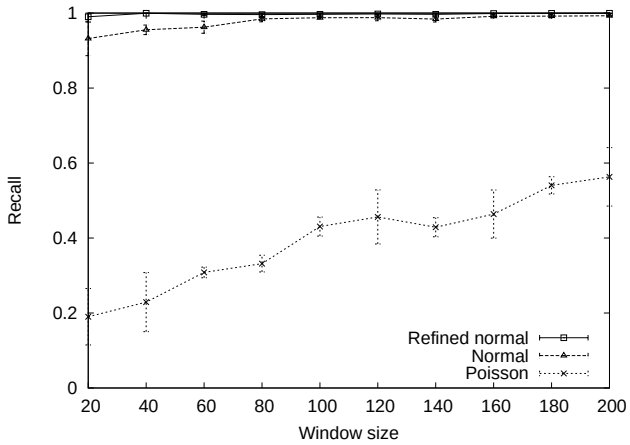


Figure 6.8: Recall for the similarity join operator when using CDF approximations.

6.5.3 Uncertain sliding windows for sum aggregation

In this section, we present our results on the evaluation of the sum aggregation on uncertain data streams. Given an uncertain sliding window $W(V, w, \alpha)$, the sum operator is defined as follows. Whenever a new point $u_i \in U$ comes in, a new tuple v_j is appended to the output stream. Tuple v_j is represented by a single instantiation, whose value is defined as the sum of the average values of the tuples within the window boundaries.

In Figure 6.9, we compare the output stream of the sum aggregation operator when using an uncertain sliding window $W(V, w, \alpha)$ and a regular sliding window $W(V, w)$ on the *Coffee* dataset. Similar results have been obtained with the other datasets, and are omitted for brevity. The experiment uses a standard deviation for existential uncertainty of 0.1, and a standard deviation for value uncertainty of 0.5. We fix the number of samples per tuple to 10, and vary the α probabilistic threshold between 0.5 and 1. We report the average absolute percentage change of the output tuple values ranging the the window size (w) between 200 and 1000. Given that sum_i is the value of the sum obtained using the regular sliding window and sum_j is the value of the sum obtained using the uncertain sliding window, the absolute percentage change between sum_i and sum_j is defined as $|sum_i - sum_j|/|sum_i|$, then multiplied by 100. The reported value is obtained by averaging the absolute percentage change across all the window shifts. The results show that the regular sliding windows are constantly over-estimating the window size, not considering the possibility that some data values do not exist in the window, which is exactly what the uncertain sliding windows model accounts for. The value of the tuples in the stream may be negative, this is why sums don't always get larger as we consider more tuples. We observe that with window size $w = 200$, there are very large differences, with differences of up to 1800% in the values of

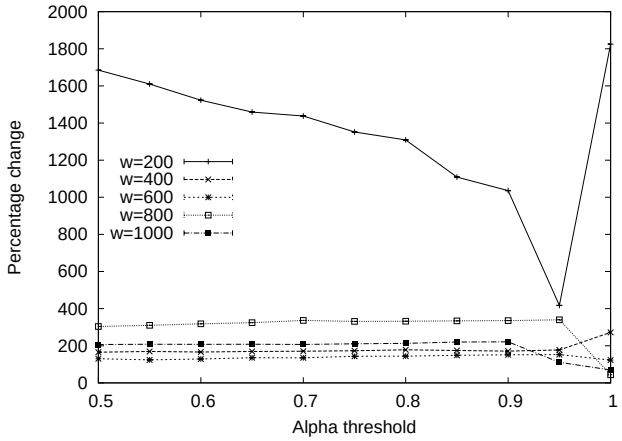


Figure 6.9: Absolute percentage change of the output tuple values when substituting a regular sliding window $W(V, w)$ with an uncertain sliding window $W(V, w, \alpha)$ for different configurations of window size w when varying the α probabilistic threshold.

the output stream. For sufficiently small windows, such as $w = 200$, the sums are affected more by changes in the stream tuple values. In constrast, on larger windows the sums tend to be more stable as positive and negative tuple values balance each other. We further note that tuning the probabilistic threshold *alpha* is a critical choice and depends on the particular application scenario. For example, in case of sum aggregations, the produced values may deviate significantly, and a large value of *alpha* is recommended.

6.5.4 Uncertain sliding windows for similarity join

In this section, we report our results on maintaining uncertain sliding windows within a similarity join operator. We evaluate our approach in terms of accuracy, performance, and memory footprint. We also report the efficiency of the pruning techniques for the join operator.

Accuracy

As shown in Figure 6.5, the performance for computing approximated results of the Poisson-binomial CDF is significantly superior to the performance of calculating an exact solution, suggesting that approximations should almost always be favored in comparison to the exact solution. As a result, we must understand how much the approximations may affect the output of a given operator when compared to the exact solution. In case of window management, approximations may result in a tuple being improperly included or excluded from the sliding window. The effect of these two situations on the join operator is that it may lead to the generation of an output tuple that should not be in the result (false positive), or to the failure of generating an output tuple that should be in the result (false negative). The approximations can also introduce errors in the existential uncertainty values of the output tuples.

To evaluate the effect of the CDF approximation in the results, we use the F1 score, which is an accuracy measure based on the *precision* and *recall* measures. Precision is defined as the percentage of uncertain tuples generated by the join which are truly matching. Recall is defined as the percentage of the truly matching uncertain tuples found by the join using approximate CDF computation.

We compute precision and recall whenever the join operator processes a new input tuple. The computation weighs the contribution to precision and recall of each output tuple $(u_i, v_j)_{\bowtie}$ by its probabilistic distance to the exact answer as follows:

$$1 - |Pr((u_i, v_j)_{\bowtie}) - Pr'((u_i, v_j)_{\bowtie})|, \tag{6.30}$$

where $Pr((u_i, v_j)_{\bowtie})$ and $Pr'((u_i, v_j)_{\bowtie})$ are the existential probabilities of the output tuple on the exact and on the approximate answers, respectively. Intuitively, the loss in accuracy of the probabilities of existence in output

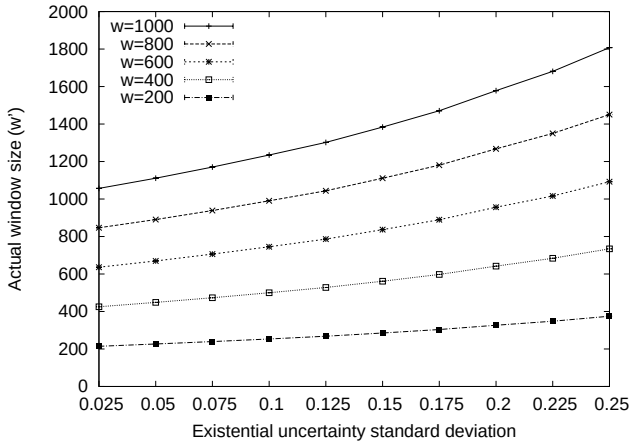


Figure 6.10: Actual size of uncertain sliding windows when varying the existential uncertainty standard deviations (σ). Memory footprint increases as the existential uncertainty standard deviation increases.

tuples impacts the precision and recall metrics. We report the average and 0.95 confidence intervals on the F1 score, precision, and recall.

Figure 6.6 shows the $F1$ score when the join operator uses the three different CDF approximations with varying window sizes (w). This experiment shows the results for data streams exhibiting uniform existential uncertainty with standard deviation $\sigma = 0.1$. The graph shows that the results of the join operator when using the Refined Normal and the Normal approximation methods are nearly the same as the ones provided by the exact solution when the window size is bigger than 80. The average $F1$ scores for the Refined Normal, Normal, and Poisson approximations are respectively 0.99, 0.98, and 0.47. As expected from the previous experiments (Figure 6.4), the Poisson approximation has very inaccurate output and should not be used for a join computation. We obtained similar trends when measuring the $F1$ score using normal and exponential distributions for existential uncertainty. In addition, we observed that the amount of existential uncertainty (varied by increasing the standard deviation for all distributions) does not affect the $F1$ score when the window size is larger than 80 (similar to Figure 6.6). This means that the proposed uncertain

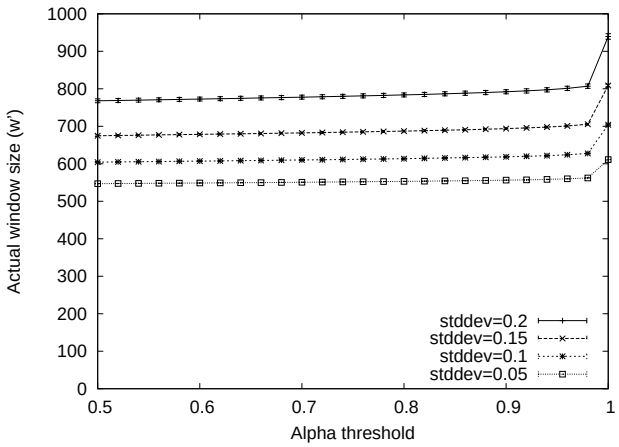


Figure 6.11: Actual size of uncertain sliding windows when varying the probabilistic threshold α and the existential uncertainty σ. Window size is more sensitive to σ than α. It also presents a steep increase as α approaches to 1.

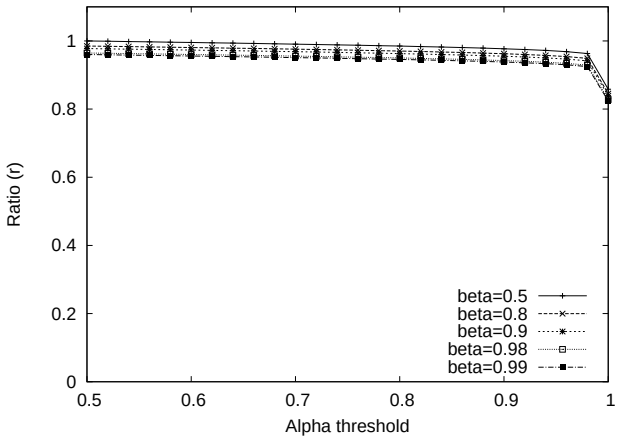


Figure 6.12: Ratio of uncertain sliding window lengths maintained by eviction policies *uncert-evict-beta* and *uncert-evict* when varying the α probabilistic threshold. *uncert-evict-beta* policy maintains windows that are up to 18% smaller.

sliding window is robust to changes in the distribution of the existential uncertainty. The graphs for the last two observations are not shown for brevity.

Figure 6.7 and 6.8 report precision and recall for the same experiment, respectively. The figures show that both the Normal and Refined Normal approximations have a small false positive and false negative rates when windows are bigger than 80. The results also show that while recall and precision measurements are very close for the Normal and Refined Normal approximations, the precision for the Poisson approximation is up to 20% higher than recall.

In conclusion, the Refined Normal method provides the highest accuracy among the approximate schemes. We use it in all of the following experiments.

We note that, in case of similarity joins or filter operators, an user may prefer to have a large value for alpha to reduce the probability of false negatives. e.g., with $alpha = 0.95$, the probability to miss a matching tuple is reduced to less than 0.05%.

Memory footprint

Memory usage for uncertain sliding windows can be measured in terms of the actual number of tuples maintained over time (w'). In Figure 6.10, we report the actual sliding window sizes (y-axis) when processing uncertain data streams that have existential uncertainty values sampled from a uniform distribution with standard deviation varying within $[0.025, 0.25]$. The figure includes results for different uncertain sliding window logical sizes (i.e., w). The results show that the actual size of the sliding window increases as the standard deviation increases. This is because there is more variability in the existential uncertainty values, leading the algorithm to maintain bigger window sizes to maintain the desired α threshold. The

results also show that the memory overhead is, on average, 82.97% when the standard deviation is 0.25 and 6.12% when it is 0.025.

Figure 6.11 reports the actual sliding window size values (w') when varying the α probabilistic threshold and the logical window size (w) is 500. The figure shows the results when the tuple existential uncertainty is drawn from a uniform distribution with standard deviations of $0.05, 0.1, 0.15$, and 0.2. Similar to Figure 6.10, we observe that the actual size of the sliding window increases as the standard deviation increases. We also observe that the window size is not that sensitive to the α value when $\alpha \in [0.5, 0.98]$, since the window size increases, on average, only 4.83% when comparing the window size at $\alpha = 0.5$ and $\alpha = 0.98$. The actual window size has a steep increase when $\alpha = 1.0$. At this point, the uncertain sliding window must have at least w tuples in it that are existentially certain. Assuming a window size of 500 (default value), the window must have at least 500 tuples that are existentially certain. Since in our experiments the standard deviation of the existential uncertainty is always above zero, we expect that the sliding window will grow, in the worst case approaching the full stream history. In practice, the probability that 500 tuples exist is reached before including the complete stream history because of the numerical imprecision in the computation of the CDF of the normal distribution in the Refined Normal approximation method.

Figure 6.12 shows the results when comparing the eviction policies *uncert-evict* and *uncert-evict-beta* reported in Algorithms 6 and 7. The sliding window size w is fixed to 500 and the parameter α varies in the range $[0.5, 1]$ (x-axis). The graph y-axis shows the sliding window ratio $r = w'_{ue_\beta}/w'_{ue}$, where w'_{ue_β} and w'_{ue} are the the number of tuples maintained in the uncertain sliding windows by the *uncert-evict-beta* and *uncert-evict* eviction policies, respectively. The same experiment has been repeated for $\beta \in [0.5, 0.99]$.

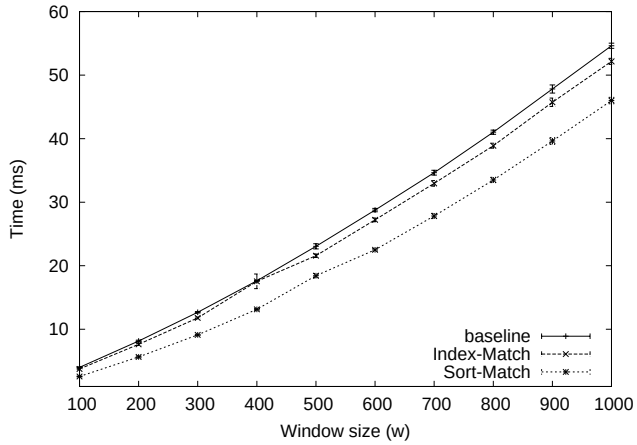


Figure 6.13: Performance of pruning strategies when varying the sliding window size. *Sort-Match* outperforms *Index-Match* for different window sizes.

We observe that uncertain sliding windows maintained by the *uncert-evict-beta* eviction policy are up to 18% smaller than those maintained by the *uncert-evict* eviction policy. The *uncert-evict-beta* algorithm shows more benefit when α has larger values. When α is close to one, a larger number of tuples are maintained in the uncertain sliding window. However, their probability of being within the sliding window boundary is very low, and below β. These results hold when varying the β probabilistic threshold.

These results show that the two key factors that impact memory footprint are (i) the amount of existential uncertainty in the input tuples, and (ii) the α threshold. As expected, larger actual sliding window sizes result in operators that are computationally more expensive.

Performance of pruning strategies

This section reports the performance of the spatial pruning technique *Index-Match* and the proposed sort-based pruning *Sort-Match*. We compare the running time of both techniques to the naive solution (labeled *baseline*), which searches for matching tuples exhaustively (i.e., does not prune the search space).

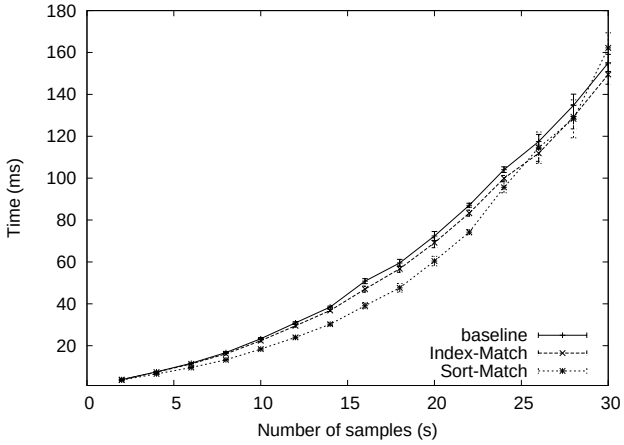


Figure 6.14: Performance of pruning strategies when varying the number of samples.

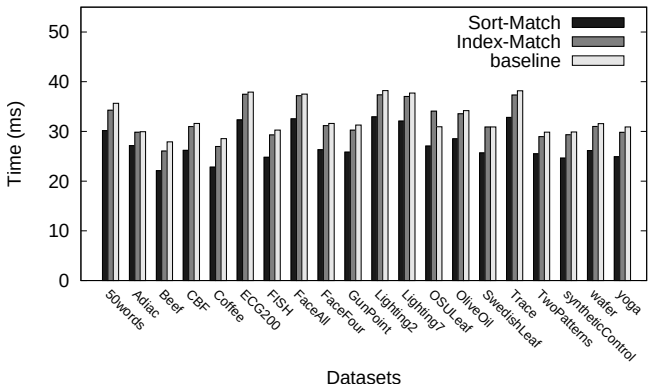


Figure 6.15: Average time performance of different pruning strategies when processing different datasets.

Figure 6.13 shows the results for our first experiment, in which we compare the processing time per tuple of the three algorithm on the *Coffee* dataset. The experiment uses a standard deviation for existential uncertainty of 0.1, and a standard deviation for value uncertainty of 0.5. We fix the number of samples per tuple to 10, and vary the sliding window size (w) between 100 and 1000. The results show that the *baseline* algorithm has the worst performance, followed by the *Index-Match* and *Sort-Match* methods.

We observe that *Index-Match* behaves as the *baseline* when tuples cannot be pruned. On the other hand, *Sort-Match* never behaves as the *baseline*, since it focuses on matching samples without enumerating all possible sample pair combinations. We observed similar trends with other datasets and omit these results for brevity.

Figure 6.14 shows the processing time per tuple of the three algorithms on the *Coffee* dataset when varying the number of samples between 2 and 30. The experiment uses a standard deviation for existential uncertainty of 0.1, and a standard deviation for value uncertainty of 0.5. The window size (w) is fixed to 500.

We observe that *Sort-Match* performs better than *Index-Match* when the number of samples is low — up to 20% when the number of samples is 16. In many real-world applications the number of available samples is rather limited, ranging between $4 - 12$ (e.g., in WiFi-based localization services [96], multiple reader RFID systems [100], and wireless sensor deployments [75]). For applications like the ones mentioned above, the low number of samples is dictated by the installations and the hardware used in these installations. In these applications, *Sort-Match* is a promising and suitable solution. When the number of samples is very large, sort-based similarity joins cannot compete with similarity joins based on indexing data structures, such as *Index-Match*. For such cases, we recommend the

use of *Index-Match*.

Figure 6.15 reports the processing time per tuple when using a sliding window of size $w = 500$ for all datasets. On average, the time per processed tuple in ms is 32.72 for *baseline*, 32.13 for *Index-Match*, and 27.51 for *Sort-Match*. The results show that *Sort-Match* consistently performs better than *Index-Match* for all datasets. In this setup, *Sort-Match* provides an average performance improvement of 16% over *Index-Match*.

6.6 Extensions

In this section, we briefly discuss the implications of existential and value uncertainty on time-based and attribute-delta-based sliding windows, as well implementation considerations for integrating the techniques introduced in this paper into a stream processing engine.

6.6.1 Other sliding window policies

A time-based sliding window, denoted by $W_{time}(S, t)$, keeps the last t seconds worth of tuples. Since tuple timestamps are certain, existential uncertainty does not affect time-based sliding windows.

An attribute-based sliding window, denoted by $W^{a}_{delta}(S, d)$, keeps the most recent tuples such that the difference between the attribute a value of the oldest and the newest tuple is not more than d (*the delta invariant*). In the case of attribute-delta sliding windows, to decide whether the oldest tuple needs to be evicted or not, we need to compute the probability that it breaks the delta invariant. This probability is $1 - \prod_{y \in Y}(1 - Pr(y))$, where Y is the set of tuples that cause violating the invariant with respect to the oldest tuple. It is straightforward to add value uncertainty into the picture.

6.6.2 Integration into System S

We are working on integrating uncertain data streams, as defined in Section 6.1, into System S [43] — an industrial-strength data stream processing engine. This involves three key changes. First, the tuple model is being updated to introduce the notion of value and existential uncertainty. Second, the windowing library is being updated to manage uncertain boundaries. And finally, the relational operators toolkit is being enhanced with operators that can work in the presence of value and existential uncertainty.

6.7 Summary

The problem of processing uncertain data streams has attracted lots of attention in the past years and has found many interesting applications across diverse domains. In many of these applications the uncertainty arises from the value uncertainty present in the data sources. However, as we have shown in this paper, there is a tight relationship between value uncertainty and existential uncertainty when composing stream operators, one inducing the other based on the topology at hand.

In this study, we investigated the implications of existential uncertainty on managing sliding windows. In past studies the window size was taken fixed and it did not depend on data uncertainty. We extended the semantics of sliding window processing by modeling the window size as the number of truly existing tuples with probabilistic guarantees. To the best of our knowledge, this problem has not been addressed before.

Interestingly, previous works on stream operators that can handle value uncertainty are mostly orthogonal to our contributions, and can easily be adapted to use our extensions. To illustrate this, we discussed the adaptation of a state-of-the-art similarity join algorithm to use uncertain sliding windows. We also presented a novel pruning strategy that can be

used to efficiently maintain uncertain sliding windows.

We evaluated the performance of the proposed techniques on many real data streams. The results show that the algorithms used to maintain uncertain sliding windows can efficiently operate while providing a high quality approximation in query answering. Based on our results, *Sort-Match* provides better time performance than *Index-Match*, when the number of tuple samples is low, as is the case for many real-world applications.

Chapter 7

Conclusions and Future Work

The management of uncertainty in data series and data streams is of great interest to many applications that process noisy, inherently uncertain and error-prone measurements. In uncertain data series, the high dimensionality and the highly correlated dimensions pose significant new challenges to the formulation and the efficient evaluation of similarity queries. In uncertain data streams, processing paradigms such as sliding windows must be adapted to accommodate uncertainty.

Similarity search queries are the basis for more complex algorithms. In this work, we compared analytically and experimentally prior studies in the field, and proposed two additional alternatives based on the moving average that were not considered before. Our experiments were based on 17 real, diverse datasets, and the results demonstrate that simple measures, based on moving average, outperform the more sophisticated alternatives. These results also suggest that a promising direction is to develop measures that take into account the sequential correlations inherent in time series.

The efficient evaluation of top-k queries is a well recognized problem. In this study, we investigated the challenges in evaluating top-k queries in uncertain data series. We formalized different models of uncertain data series, presenting and discussing their properties. We defined the top-k

nearest neighbor problem in uncertain data series and propose a variety of methods that assess and extend prior studies in the field. We further investigated the adoption of pruning techniques based on spatial and metric bounds. We evaluated our proposal under a variety of settings using 45 real datasets from diverse domains and synthetic datasets. The results show that modeling uncertainty with probabilistic models can lead to more accurate results. Our proposal proved to be up to orders of magnitude more efficient than previously proposed techniques.

We investigated the implications of existential uncertainty in managing sliding windows. In past studies the window size was fixed and it did not depend on data uncertainty. We extended the semantics of sliding window processing by modeling the window size as the number of truly existing tuples with probabilistic guarantees. Previous works on stream operators that can handle value uncertainty are mostly orthogonal to our contributions, and can easily be adapted to use our extensions. To illustrate this, we discussed the adaptation of a state-of-the-art similarity join algorithm to use uncertain sliding windows. We also presented a novel pruning strategy that can be used to efficiently maintain uncertain sliding windows. We evaluated the performance of the proposed techniques on real data streams. The results show that the algorithms used to maintain uncertain sliding windows can efficiently operate while providing a high quality approximation in query answering.

7.1 Future Directions

The "possible world" semantics are a convenient model for uncertain data series. The experiments show that representing the possible instantiations using a set of series to represent the full-joint distribution of uncertain data series can lead to more accurate results and computationally effi-

cient algorithms. As verified in the experiments, the moving average can leverage the correlation of the values at neighboring time-stamps to reduce the uncertainty along the series. In this thesis we haven't experimented with the combination of these two interesting results. In particular, the *series-uncertainty* model presented in Chapter 5 can be combined with traditional machine learning models for time series such as Markov models and Bayesian networks. For example, an instantiation of the uncertain series can be modeled by a hidden Markov model. Consequently, an uncertain series can be modeled by a set of hidden Markov models. The resulting hidden Markov models can be further unified in a more general model, that describes the complex underlying dynamics of the series and its uncertainty.

The top-k nearest neighbor formulation considered in Chapter 5 may produce inaccurate results if used as a NN classifier. Accordingly to Definition 5.1.1), the nearest neighbor is the uncertain series whose samples are among the closest to the query. However, if one sample from candidate i is the closest to the query and the next k closest samples belong to candidate $j \neq i$, which is the correct uncertain series to be returned as nearest neighbor? The query may be a legitimate sample of candidate i but uncertain series j would be incorrectly returned as nearest neighbor. The algorithm *Holistic-kNN* proposed in Chapter 5 can be revised to consider quantiles of the distance distributions to mitigate the problem. We note that the distance samples are maintained in sorted lists in our proposal, a convenient representation to determine the quantiles efficiently.

In data stream applications, uncertainty can propagate through the different stream operators. Adapting sliding windows to support uncertain data is a significant step toward an uncertainty-aware data stream processing system. However, little work has been done to study the implications of uncertainty on the composition of multiple stream operators. An ex-

haustive enumeration of all the possible combinations under the "possible world" model may not be a practical solution in many scenarios. A statistical approach based on sampling and Monte Carlo simulations can be an interesting direction to investigate further.

Bibliography

[1] Keogh, E., Xi, X., Wei, L. & Ratanamahatana, C. A. (2006). The UCR Time Series Classification/Clustering Homepage: www.cs.ucr.edu/~eamonn/ time_series_data/. Accessed on 17 May 2011.

[2] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Çetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stanley B. Zdonik. The design of the borealis stream processing engine. In *Conference on Innovative Data Systems Research (CIDR)*, pages 277–289, January 2005.

[3] Charu C. Aggarwal. On unifying privacy and uncertain data models. In *ICDE*, pages 386–395, 2008.

[4] Charu C. Aggarwal. *Managing and Mining Uncertain Data*, volume 35 of *Advances in Database Systems*. Kluwer, 2009.

[5] Charu C. Aggarwal and Philip S. Yu. A framework for clustering uncertain data streams. In *IEEE ICDE*, 2008.

[6] Charu C. Aggarwal and Philip S. Yu. A survey of uncertain data algorithms and applications. *IEEE Trans. Knowl. Data Eng.*, 21(5):609–623, 2009.

[7] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, pages 1151–1154, 2006.

[8] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. *Foundations of Data Organization and Algorithms*, 1993.

[9] Fabrizio Angiulli and Fabio Fassetti. Indexing uncertain data in general metric spaces. *IEEE Trans. Knowl. Data Eng.*, 24(9):1640–1657, 2012.

[10] Lyublena Antova, Christoph Koch, and Dan Olteanu. Query language support for incomplete information in the maybms system. In *VLDB*, pages 1422–1425, 2007.

[11] Johannes Aßfalg, Hans-Peter Kriegel, Peer Kröger, and Matthias Renz. Probabilistic similarity search for uncertain time series. In *SSDBM*, pages 435–443, 2009.

[12] O. Benjelloun, A.D. Sarma, A. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, 2006.

[13] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop*, pages 359–370, 1994.

[14] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, 1994.

[15] Thomas Bernecker, Hans-Peter Kriegel, Matthias Renz, Florian Verhein, and Andreas Züfle. Probabilistic frequent itemset mining in uncertain databases. In *KDD*, pages 119–128, 2009.

[16] George Beskales, Mohamed A. Soliman, and Ihab F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. *PVLDB*, 1(1):326–339, 2008.

[17] Alain Biem, Eric Bouillet, Hanhua Feng, Anand Ranganathan, Anton Riabov, Olivier Verscheure, Haris Koutsopoulos, and Carlos Moran. IBM Infosphere Streams for scalable, real-time, intelligent transportation services. In *ACM SIGMOD*, 2010.

[18] C. Bohm, A. Pryakhin, and M. Schubert. The gauss-tree: Efficient object identification in databases of probabilistic feature vectors. In *IEEE ICDE*, 2006.

[19] Toon Calders, Calin Garboni, and Bart Goethals. Approximation of frequentness probability of itemsets in uncertain data. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 749–754. IEEE, 2010.

[20] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. isax 2.0: Indexing and mining one billion time series. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, 2010.

[21] Matteo Ceriotti, Michele Corra, Leandro D'Orazio, Roberto Doriguzzi, Daniele Facchin, Stefan Guna, Gian Paolo Jesi, Renato Lo Cigno, Luca Mottola, Amy L. Murphy, Massimo Pescalli, Gian Pietro Picco, Denis Pregnolato, and Carloalberto Torghele. Is There Light at the Ends of the Tunnel? Wireless Sensor Networks for Adaptive Lighting in Road Tunnels. In *International Conference on Information Processing in Sensor Networks (IPSN)*, pages 187–198, 2011.

[22] Kin-Pong Chan and Ada Wai-Chee Fu. Efficient time series matching by wavelets. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 126–133. IEEE, 1999.

[23] K.P. Chan and A.W.C. Fu. Efficient time series matching by wavelets. In *ICDE*, pages 126–133. IEEE, 2002.

[24] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Computing Surveys (CSUR)*, 33(3):273–321, 2001.

[25] Lei Chen, M Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2005.

[26] 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, September 2004.

[27] 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 Thirtieth international conference on Very large data bases-Volume 30*, pages 876–887. VLDB Endowment, 2004.

[28] Reynold Cheng, Jinchuan Chen, Mohamed F. Mokbel, and Chi-Yin Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *ICDE*, pages 973–982, 2008.

[29] Paolo Ciaccia and Marco Patella. Bulk loading the m-tree. In *Proceedings of the 9th Australasian Database Conference (ADC98)*, pages 15–26, 1998.

[30] X. Dai, M. Yiu, N. Mamoulis, Y. Tao, and M. Vaitis. Probabilistic spatial queries on existentially uncertain data. In *SSTD*, 2005.

[31] Michele Dallachiesa, Besmira Nushi, Katsiaryna Mirylenka, and Themis Palpanas. Uncertain time-series similarity: Return to the basics. *PVLDB*, 5(11):1662–1673, 2012.

[32] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[33] Nilesh N. Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.

[34] Gautam Das, Dimitrios Gunopulos, and Heikki Mannila. Finding similar time series. *Principles of Data Mining and Knowledge Discovery*, pages 88–100, 1997.

[35] Constantinos Daskalakis, Ilias Diakonikolas, and Rocco A Servedio. Learning poisson binomial distributions. In *Proceedings of the 44th symposium on Theory of Computing*, pages 709–728. ACM, 2012.

[36] Yanlei Diao, Boduo Li, Anna Liu, Liping Peng, Charles Sutton, Thanh T. L. Tran, and Michael Zink. Capturing data uncertainty in high-volume stream processing. In *CIDR*, 2009.

[37] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB*, 2008.

[38] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Conference*, 23(2):419–429, 1994.

[39] M. Fernandez and S. Williams. Closed-form expression for the poisson-binomial probability density function. *IEEE Transactions on Aerospace and Electronic Systems*, 46(2), 2010.

[40] Benjamin C. M. Fung, Ke Wang, Rui Chen, and Philip S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, 42(4), 2010.

[41] Ariel Fuxman, Elham Fazli, and Renée J Miller. Conquer: Efficient management of inconsistent databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 155–166. ACM, 2005.

[42] B. Gedik. A windowing library for extensible stream processing systems. *Software: Practice & Experience*, 2013.

[43] Bugra Gedik and Henrique Andrade. A model-based framework for building extensible, high performance stream processing middleware and programming language for IBM InfoSphere Streams. *Software - Practice and Experience Journal, Wiley*, 42(11), 2012.

[44] J.Y. Halpern. *Reasoning about uncertainty*. MIT Press, 2003.

[45] M.P. Hamilton, E.A. Graham, P.W. Rundel, M.F. Allen, W. Kaiser, M.H. Hansen, and D.L. Estrin. New Approaches in Embedded Networked Sensing for Terrestrial Ecological Observatories. *Environmental Engineering Science*, 24(2), 2007.

[46] Martin Hirzel, Henrique Andrade, Bugra Gedik, Vibhore Kumar, Giuliano Losa, Mark Mendell, Howard Nasgaard, Robert Soulé, and Kun-Lung Wu. SPL stream processing language specification. Technical Report RC24897, IBM Research, 2009.

[47] Y. Hong. On computing the distribution function for the sum of independent and non-identical random indicators. Technical report, Department of Statistics, Virginia Tech, 2011.

[48] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

[49] T. S. Jayram, Andrew McGregor, S. Muthukrishnan, and Erik Vee. Estimating statistical aggregates on probabilistic data streams. In *Proceedings of the 26th ACM Symposium on Principles of Database Systems (PODS'07)*, pages 243–252, New York, NY, USA, 2007. ACM.

[50] Jeffrey Jestes, Graham Cormode, Feifei Li, and Ke Yi. Semantics of ranking queries for probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 23(12):1903–1917, 2011.

[51] Cheqing Jin, Ke Yi, Lei Chen, Jeffrey Xu Yu, and Xuemin Lin. Sliding-window top-k queries on uncertain streams. *Proceedings of the VLDB Endowment*, 1(1):301–312, 2008.

[52] D.V. Kalashnikov, Y. Ma, S. Mehrotra, and R. Hariharan. Index for fast retrieval of uncertain spatial point data. In *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pages 195–202. ACM, 2006.

[53] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *IEEE ICDE*, 2008.

[54] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286, 2001.

[55] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

[56] Hans-Peter Kriegel, Peter Kunath, and Matthias Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA*, pages 337–348, 2007.

[57] H.P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Probabilistic similarity join on uncertain data. In *Database Systems for Advanced Applications*, pages 295–309. Springer, 2006.

[58] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *Embedded networked sensor systems*, pages 64–75. ACM, 2005.

[59] W. Kuo and M.J. Zuo. *Optimal reliability modeling: principles and applications*. John Wiley & Sons, 2003.

[60] Carson Kai-Sang Leung and Boyu Hao. Mining of frequent itemsets from streams of uncertain data. In *IEEE ICDE*, 2009.

[61] X. Lian and L. Chen. Similarity join processing on uncertain data streams. *IEEE Transactions on Knowledge and Data Engineering*, 2010.

[62] Xiang Lian and Lei Chen. Probabilistic ranked queries in uncertain databases. In *EDBT*, pages 511–522, 2008.

[63] Xiang Lian and Lei Chen. Efficient join processing on uncertain data streams. In *CIKM*, pages 857–866, 2009.

[64] J. Lin, E. 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.

[65] V. Ljosa and A.K. Singh. APLA: Indexing arbitrary probability distributions. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 946–955. IEEE, 2007.

[66] Chunyang Ma, Hua Lu, Lidan Shou, Gang Chen, and Shujie Chen. Top-k similarity search on uncertain trajectories. In *SSDBM*, pages 589–591, 2011.

[67] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE TKDE*, 13(1), 2001.

[68] Y.S. Moon, K.Y. Whang, and W.S. Han. General match: a subsequence matching method in time-series databases based on generalized windows. In *SIGMOD Conference*, pages 382–393. ACM, 2002.

[69] Y.S. Moon, K.Y. Whang, and W.K. Loh. Duality-based subsequence matching in time-series databases. In *ICDE*, pages 263–272. IEEE, 2002.

[70] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In *International Workshop on Knowledge Discovery Using Cloud and Distributed Computing Platforms (KDCloud 2010)*, pages 170–177, December 2010.

[71] Roar Nybø. Time series opportunities in the petroleum industry. In *ESTSP 08, European Symposium on Time Series Prediction, Porvoo, Finland.*, 2008.

[72] Marco Patella Paolo Ciaccia and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, pages 426–435, 1997.

[73] Spiros Papadimitriou, Feifei Li, George Kollios, and Philip S. Yu. Time series compressibility and privacy. In *VLDB*, 2007.

[74] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM, 2012.

[75] Usman Raza, Alessandro Camerra, Amy L. Murphy, Themis Palpanas, and Gian Pietro Picco. What does model-driven data acquisition really achieve in wireless sensor networks? In *PERCOM*, 2012.

[76] Christopher Re, Nilesh Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 886–895. IEEE, 2007.

[77] Christopher Ré, Julie Letchner, Magdalena Balazinska, and Dan Suciu. Event queries on correlated probabilistic streams. In *ACM SIGMOD*, 2008.

[78] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD Conference*, pages 71–79. ACM, 1995.

[79] S.R. Sarangi and K. Murthy. DUST: a generalized notion of similarity between uncertain time series. In *SIGKDD*, pages 383–392. ACM, 2010.

[80] Jin Shieh and Eamonn J. Keogh. *i*SAX: indexing and mining terabyte sized time series. In *KDD*, pages 623–631, 2008.

[81] Sarvjeet Singh, Chris Mayfield, Sagar Mittal, Sunil Prabhakar, Susanne E. Hambrusch, and Rahul Shah. Orion 2.0: native support for uncertain data. In *SIGMOD Conference*, pages 1239–1242, 2008.

[82] Sarvjeet Singh, Chris Mayfield, Rahul Shah, Sunil Prabhakar, Susanne E. Hambrusch, Jennifer Neville, and Reynold Cheng. Database support for probabilistic attributes and tuples. In *IEEE ICDE*, 2008.

[83] Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. Top-k query processing in uncertain databases. In *ICDE*, pages 896–905, 2007.

[84] Daby Sow, Alain Biem, Marion Blount, Maria Ebling, and Olivier Verscheure. Body sensor data processing using stream computing. In *Proceedings of the International Conference on Multimedia Information Retrieval (MIR'10)*, pages 449–458, New York, NY, USA, 2010. ACM.

[85] Michael Stonebraker, Jacek Becla, David J. DeWitt, Kian-Tat Lim, David Maier, Oliver Ratzesberger, and Stanley B. Zdonik. Requirements for science data bases and scidb. In *CIDR*, 2009.

[86] Dan Suciu, Andrew Connolly, and Bill Howe. Embracing uncertainty in large-scale computational astrophysics. In *MUD*, pages 63–77, 2009.

[87] Liwen Sun, Reynold Cheng, David W Cheung, and Jiefeng Cheng. Mining uncertain data with probabilistic guarantees. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 273–282. ACM, 2010.

[88] 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 *VLDB*, 2005.

[89] Goce Trajcevski, Alok N. Choudhary, Ouri Wolfson, Li Ye, and Gang Li. Uncertain range queries for necklaces. In *Mobile Data Management*, pages 199–208, 2010.

[90] Thanh T Tran, Liping Peng, Yanlei Diao, Andrew McGregor, and Anna Liu. Claro: modeling and processing uncertain data streams. *The VLDB JournalThe International Journal on Very Large Data Bases*, 21(5):651–676, 2012.

[91] Thanh TL Tran, Liping Peng, Boduo Li, Yanlei Diao, and Anna Liu. Pods: a new model and processing algorithms for uncertain data streams. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 159–170. ACM, 2010.

[92] Liang Wang, D Cheung, Reynold Cheng, S Lee, and Xuan Yang. Efficient mining of frequent itemsets on large uncertain databases. 2012.

[93] L. Wei, E. Keogh, H. Van Herle, and A. Mafra-Neto. Atomic wedgie: efficient query filtering for streaming time series. In *Data Mining, Fifth IEEE International Conference on*, 2005.

[94] Kun-Lung Wu, Philip S. Yu, Bugra Gedik, Kirsten Hildrum, Charu C. Aggarwal, Eric Bouillet, Wei Fan, David George, Xiaohui Gu, Gang Luo, and Haixun Wang. Challenges and experience in prototyping a multi-modal stream analytic and monitoring application on System S. In *VLDB*, 2007.

[95] M.Y. Yeh, K.L. Wu, P.S. Yu, and M.S. Chen. PROUD: a probabilistic approach to processing similarity queries over uncertain data streams. In *EDBT*, pages 684–695. ACM, 2009.

[96] M. Youssef, M. Mah, and A. Agrawala. Challenges: device-free passive localization for wireless environments. In *ACM MOBICOM*, 2007.

[97] Qin Zhang, Feifei Li, and Ke Yi. Finding frequent items in probabilistic data. In *Proceedings of the 2008 ACM International Conference on Management of data (SIGMOD08)*, pages 819–832, New York, NY, USA, 2008. ACM.

[98] Wenjie Zhang, Xuemin Lin, Ying Zhang, Wei Wang, and Jeffrey Xu Yu. Probabilistic skyline operator over sliding windows. In *IEEE ICDE*, 2009.

[99] Yuchen Zhao, Charu C. Aggarwal, and Philip S. Yu. On wavelet decomposition of uncertain time series data sets. In *CIKM*, pages 129–138, 2010.

[100] Zongheng Zhou, Himanshu Gupta, Samir R. Das, and Xianjin Zhu. Slotted scheduled tag access in multi-reader rfid systems. In *IEEE ICNP*, 2007.