Knowl Inf Syst (2015) 45:159–190
DOI 10.1007/s10115-014-0804-5

REGULAR PAPER

Sliding windows over uncertain data streams

Michele Dallachiesa · Gabriela Jacques-Silva · Buğra Gedik · Kun-Lung Wu · Themis Palpanas

Received: 10 July 2013 / Revised: 31 July 2014 / Accepted: 3 November 2014 /
Published online: 25 December 2014

Abstract 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 <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 easily 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

M. Dallachiesa (✉) · T. Palpanas
University of Trento, Trento, Italy
e-mail: michele.dallachiesa@gmail.com

M. Dallachiesa · G. Jacques-Silva · K.-L. Wu
IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
e-mail: g.jacques@us.ibm.com

K.-L. Wu
e-mail: klwu@us.ibm.com

B. Gedik
Bilkent University, Ankara, Turkey
e-mail: bgedik@cs.bilkent.edu.tr

T. Palpanas
Paris Descartes University, Paris, France
e-mail: themis@mi.parisdescartes.fr

used to maintain uncertain sliding windows can efficiently operate 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.

Keywords Data stream processing · Sliding windows · Uncertainty management

1 Introduction

The strong demand for applications that continuously monitor the occurrence of interesting events (e.g., road-tunnel management [39] and health monitoring [43]) has driven the research in data stream processing systems [1,14,21,48]. In many of these application domains, the data sources available for processing can be considered *uncertain*, because of the imprecisions that arise from the inherent inaccuracy of sensor devices, or of external data manipulations like privacy-preserving data transformations [19].

The uncertainty of a stream data item (or tuple) can be twofold: (i) *value uncertainty*, and (ii) *existential uncertainty*. A tuple has value uncertainty when its value is represented by either a Probability Density Function (PDF) [49] or by discrete samples [4]. In the latter case, each sample is called a *possible value* and has an *existential probability* associated with it (indicating the chance that the tuple assumes the associated possible value). In this work, we represent value uncertainty with discrete samples and their respective occurrence probabilities. A tuple has existential uncertainty when the sum of the existential probabilities of its possible values is <1.

Modeling tuples with value and existential uncertainty has several advantages. From an engineering perspective, a programmer can feed uncertain data directly into the system, without explicitly preprocessing data and forcing data approximations. From an application requirements perspective, maintaining possible values allows the application to provide results with confidence intervals. Simply averaging values and eliminating the uncertainty may lead to misleading results, as this technique does not take into account the distribution of the data.

Monitoring of offshore drilling operations is an example application where data sources are uncertain and the accuracy of the results is crucial [38]. 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. This confidence can be seen as the existential uncertainty associated with the event.

Another example application where result accuracy is key is the 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 [8], the exact location of a car in a highly urbanized area is uncertain, as GPS provides inaccurate data in such scenarios. As a result, the position of a car can be modeled as a set of possible locations with attached probabilities (i.e., a value uncertain tuple).

This set can be obtained by correlating GPS data with road network data. Similarly, particle filters have been used by prior studies [35] and [34] to derive the location of moving objects based on the GPS signal. The particle filters can be used as weighted samples to represent the distribution of the object location. The set can then be used to estimate many possible distance measures to cars nearby. By filtering samples in which the distance between cars is above the safety limit, we obtain a stream of tuples that is existentially uncertain. Discarding value and existential uncertainty can lead to the following two possible outcomes: (i) tagging safe drivers as unsafe, which results in the insurance company increasing the premium unfairly and a significant risk of losing clients; (ii) unsafe drivers tagged as safe, resulting in the insurance company decreasing the premium and risking its own profit model.

Current research in processing uncertain data streams focuses mostly on the development of specific stream operators (e.g., joins [30,33] and aggregates [26]) and specific queries (e.g., top-k [27,51] and clustering [3]) 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 [1,24,37]. 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 [1,20,27,33]. 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, and larger window size may lead to over-estimates of the desired window size, eventually causing undesired and unexpected effects. In this study, we investigate this problem.

1.1 Contributions

In this paper, 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 toward developing applications via operator composition.

The second challenge is to provide an efficient implementation of an uncertain sliding window in terms of both 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 many realistic scenarios.

The main contributions of this paper 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;
- We 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 datasets, and show improvement (on all 17 datasets) over a state-of-the-art approach [33] adapted to handle existential uncertainty.

The rest of this paper is organized as follows. We discuss related work in Sect. 2. Uncertain data streams are introduced in Sect. 3. In Sect. 4, we describe a model that allows for efficient processing of sliding windows with uncertain data. In Sect. 5, we describe how uncertain sliding windows can be used by aggregate and join operators. In Sect. 6, 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 Sect. 7, and in Sect. 8, we discuss some possible extensions. Section 9 concludes the paper.

2 Related work

In the last decade, several database and stream processing systems with support for uncertainty have been proposed [6,13,15,17,27,28,42,45,46], eventually leading to two emerging tuple models.

The *x-tuple* model [6] represents uncertain tuples by multiple alternatives and their respective occurring probabilities. If the sample probabilities do not sum up to one, there exist possible instantiations of the uncertain stream where the tuple does not exist. Uncertain tuples are processed according to the possible world semantics [23].

In the *attribute* model [17,42], uncertainty is more fine-grained, and it refers to single tuple attributes. An uncertain attribute is represented by a random variable whose 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 [42].

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 are often provided by means of discrete samples drawn from unknown distributions. Fourth, possible world 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 [33] 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. [17] 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. [40] 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. [13] perform an extensive experimental and analytical comparison of methods for answering similarity matching queries on uncertain time series.

In [11], an augmented R-tree indexes a dataset of spatial points with existential uncertainty. The authors represent existential uncertainty by independent probability values associated with 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 [27], 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 [32], 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. [52] propose an efficient method to maintain skylines over uncertain data streams. A skyline is a set of items that 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.

CLARO and PODS [26,45] are a probabilistic data stream processing systems that represent continuous-valued attributes using Gaussian mixture models. Formal semantics for

relational processing are presented for operators including joins and aggregates. Exact result distributions for aggregates based on characteristic functions and exact closed forms are presented. The authors acknowledge that these algorithms may be impractical because the time complexity grows exponentially in the number of input tuples, and propose approximated schemes. Joins are evaluated by using cross-product semantics or the novel concept of probabilistic views, that is used to derive closed form join result distributions in the form of Gaussian mixture models. Existential uncertainty of tuples is recognized as an important issue that requires extensions to the current proposal, using expensive computational methods such as Monte Carlo simulations.

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 Sect. 6, previous works on streaming operations with sliding windows can be easily adapted to accommodate our extensions.

Although in this work, we focus on existential uncertainty in data streams, similar forms of structural uncertainty have been investigated also on linked data, where the connections between different entities are uncertain [12,22]. These models can be used for link prediction and collective classification. However, they haven't been designed to estimate the number of existing links, and they cannot be easily applied to our problem definition.

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. [7], which propose an algorithm with time cost $O(n^2)$ based on dynamic programming, and Sun et al. [44], which propose an algorithm based on divide-and-conquer with time cost $O(nlog^2n)$. Other approximation algorithms also exist [9,47].

We use one exact method (RF1) and three approximations (Poisson, Normal, and Refined Normal Approximations), as reviewed in [25]. Independence is a simplifying assumption widely used in prior studies on uncertain data management [2].

We observe that in our data model, the parameters of the Poisson-binomial distribution can be easily derived from the existential probabilities. In particular situations where this information is not available, a simplified data model can be adopted and the distribution parameters must be estimated. In [16], the authors propose an algorithm which learns Poisson-binomial distributions with ϵ-accuracy from input samples.

Hierarchical Markov models and conditional random fields have been used to learn and infer a user's daily movements [35] from noisy sensor measurements. Our proposal can be used in these applications to model more accurately the imprecise location of users by filtering out noise using sliding windows and aggregate operators.

3 Uncertain data streams

3.1 Preliminaries

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

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

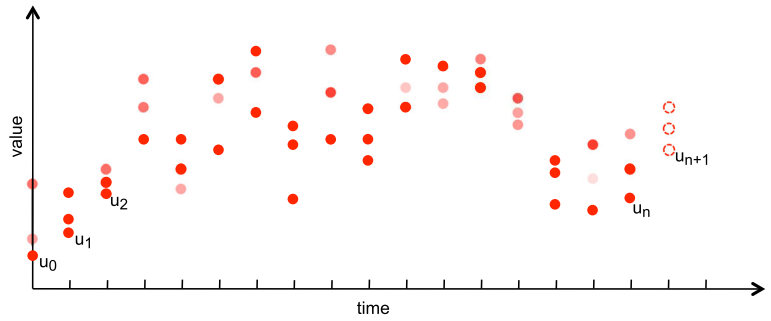

Fig. 1 Example of an uncertain data stream, where uncertainty is modeled by repeated weighted measurements and tuples are one-dimensional points. Weights are encoded using transparency, i.e., *lighter points* occur with lower probability

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

$$Pr(u_i) = \sum_{u_{i,j} \in u_i} Pr(u_{i,j}). \tag{1}$$

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. Figure 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.

3.2 From value to existential uncertainty

We use a *filter* stream operator 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 $Pr(v_k) < 1$.

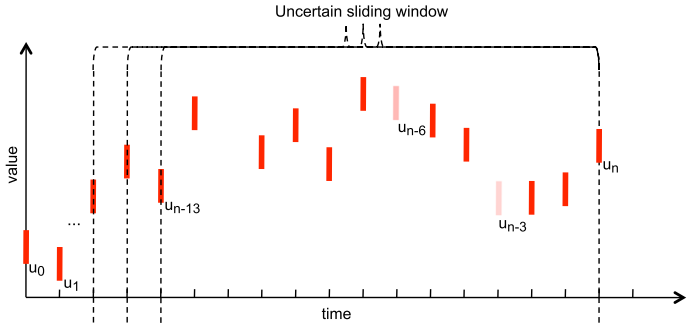

Fig. 2 Example of an uncertain sliding window. Bounding intervals drawn using *dashed lines* represent the sliding window content, whereas *light colored bars* represent existentially uncertain tuples

3.3 From existential to value uncertainty

As described in Sect. 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 $t_\eta = s_\eta + \cdots + s_{\eta-w+1}$.

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 which 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 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 an output tuple that can have up to four different summation values and their corresponding probabilities.

4 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 win-

Table 1 Symbols used in the paper and their explanations

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

dows. 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 world semantics. Intuitively, the number of tuples actually maintained in the sliding window can 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 Sect. 5.

In Table 1, we summarize the most important symbols used in the rest of the paper.

4.1 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 Fig. 2. To model this boundary, we first define $\hat{W}(U, w)$ as the subsequence of tuples in a materialization of $W(U, w)$. 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 Fig. 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:

$$Pr(|\hat{W}(U, w')| = w) = 1. \tag{2}$$

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 to one. This leads to a refinement of the probabilistic condition in Eq. (2), as follows:

$$Pr(|\hat{W}(U, w')| \geq w) = 1 \ \wedge \ w' \text{minimal}. \tag{3}$$

The second problem is that if all tuples in U are existentially uncertain, the value of w' in $\hat{W}(U, w')$ approaches to the total size of U (or infinity) when Eq. (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:

$$Pr(|\hat{W}(U, w')| \geq w) \ \geq \ \alpha \ \wedge \ w' \text{minimal}. \tag{4}$$

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 Eq. (4). Thus, α serves as a probabilistic bound that limits w'.

We note that Eq. 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 [10]. In the following, we will consider this definition to define the probabilistic bounds of uncertain sliding windows.

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

Algorithm 1 *uncert-evict*

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

The algorithm evaluates the probabilistic condition defined in Eq. (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 1, we need a model for the random variable $|\hat{W}(U, w' - 1)|$ in terms of its cumulative distribution function (CDF):

$$Pr(|\hat{W}(U, w' - 1)| \geq w) = 1 - Pr(|\hat{W}(U, w' - 1)| \leq w - 1). \tag{5}$$

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

$$I_i \sim Bernoulli(Pr(u_i)), \tag{6}$$

where $Pr(u_i)$ is the existential probability of tuple u_i as defined in Eq. (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:

$$|\hat{W}(U, w' - 1)| = \sum_{i=\eta-w'+2}^{\eta} I_i. \tag{7}$$

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 [40] 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 [31] as well as in many other application areas [18].

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

4.3 The Poisson-binomial distribution

We first look at computing the exact CDF. Let $I_1, \ldots, I_n$ be n independent Bernoulli random variables with success probabilities $p_1, \ldots, p_n$. 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:

$$Pr(N = k) = \sum_{A \in F_k} \prod_{i \in A} p_i \prod_{i \in A^c} (1 - p_i), \tag{8}$$

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

$$Pr(N \leq k) = \sum_{i=0}^{k} Pr(N = i). \tag{9}$$

Since F_k in Eq. (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 [25], 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:

$$Pr(X_j = l) = (1 - p_j) \cdot Pr(X_{j-1} = l) \; + p_j \cdot Pr(X_{j-1} = l - 1), \tag{10}$$

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 Sect. 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 1. Experiments in Sect. 7.2 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.

4.4 Efficient approximations of the Poisson-binomial distribution

Hong [25] 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:

$$\mu = E(N) = sum_{\mu}, \tag{11}$$

$$\sigma = \sqrt{Var(N)} = \sqrt{sum_{\sigma}}, \tag{12}$$

$$\gamma = Skewness(N) = \frac{1}{\sigma^3} sum_{\gamma}, \tag{13}$$

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 Sects. 4.2 and 4.3, 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 [25]:

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

$$Pr(N \leq k) \approx \sum_{i=1}^{k} \frac{\mu^k \exp(-\mu)}{k!}. \tag{14}$$

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

$$Pr(N \leq k) \approx \Phi\left(\frac{k + 0.5 - \mu}{\sigma}\right), \tag{15}$$

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:

$$Pr(N \leq k) \approx G\left(\frac{k + 0.5 - \mu}{\sigma}\right), \tag{16}$$

where

$$G(x) = \Phi(x) + \frac{\gamma(1 - x^2)\phi(x)}{6}, \tag{17}$$

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

5 Adapting stream operators to handle data uncertainty

Windowed stream operators reviewed in Sect. 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 [33].

The similarity join operator correlates similar tuples from two input data streams. When the operator receives a new tuple, it evaluates whether 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:

$$T' = \{(s_i, t_j) \mid Dist(s_i, t_j) \leq \epsilon \wedge t_j \in W(T, w)\}, \tag{18}$$

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 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

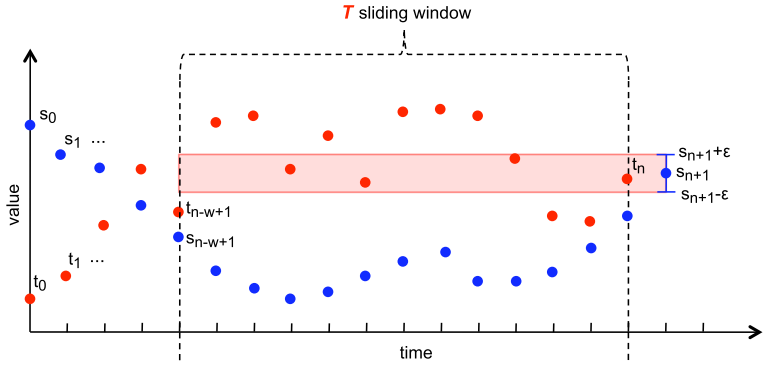

Fig. 3 Example of a similarity join between certain data streams. *Interval bar* displays tuples in $W(T, w)$ that are similar to $s_{\eta+1}$ based on the distance threshold ϵ. *Blue* (*dark*) and *red* (*light*) *dots* represent the values of the two streams to be joined

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:

$$V' = \{(u_i, v_j)_{\bowtie} \text{ s.t. } v_j \in W(V, w, \alpha) \wedge Pr(match(u_i, v_j)) \geq \beta\}, \tag{19}$$

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:

$$(u_i, v_j)_{\bowtie} = \{(u_{i,k}, v_{j,l}) \text{ s.t. } dist(u_{i,k}, v_{j,l}) \leq \epsilon\}. \tag{20}$$

To evaluate the match probability, we first evaluate whether 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:

$$Pr(match_s(u_i, v_j)) = \sum_{(u_{i,k}, v_{j,l}) \in (u_i, t_j)_{\bowtie}} Pr(u_{i,k}) \cdot Pr(v_{j,l}). \tag{21}$$

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

$$Pr(match(u_i, v_j)) = Pr(v_j \in \hat{W}_{[w]}(V, w') \wedge match_s(u_i, v_j)), \tag{22}$$

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

$$\begin{aligned} Pr(match(u_i, v_j)) = {} & Pr(v_j \in \hat{W}_{[w]}(V, w')) \cdot \\ & Pr(match_s(u_i, v_j) \mid v_j \in \hat{W}_{[w]}(V, w')). \end{aligned} \tag{23}$$

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

$$\begin{aligned} Pr(match(u_i, v_j)) = {} & Pr(v_j \in \hat{W}_{[w]}(V, w')) \cdot \\ & Pr(match_s(u_i, v_j))/Pr(v_j). \end{aligned} \tag{24}$$

In Eq. (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:

$$Pr(v_j \in \hat{W}_{[w]}(V, w')) = Pr(v_j) \cdot Pr(|\hat{W}(V, \eta - j)| \leq w - 1), \tag{25}$$

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

$$\begin{aligned} Pr(match(u_i, v_j)) = \; & Pr(|\hat{W}(V, \eta - j)| \leq w - 1) \cdot \\ & Pr(match_s(u_i, v_j)) \end{aligned} \tag{26}$$

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 Sect. 4.3.

6 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 [33] 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.

6.1 Upper-bounding the match probability

As described in Sect. 5, 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:

$$\lim_{\alpha \to 1} Pr(u_{\eta - w' + 1} \in \hat{W}(U, w')) = 0. \tag{27}$$

From Eq. (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 1 for maintaining *uncertain sliding windows* such that it also takes β into account.

From Eq. (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 2 shows the updated window eviction policy. This policy results in smaller uncertain sliding windows and an overall performance improvement.

Algorithm 2 *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**

In Algorithm 2, we use the following derivation to bring the eviction conditions into the same form and avoid repeated computation:

$$\begin{aligned} Pr(|\hat{W}(U, w' - 1)| < w) &= 1 - Pr(|\hat{W}(U, w' - 1)| \geq w) \\ Pr(|\hat{W}(U, w' - 1)| < w) < \beta &\equiv Pr(|\hat{W}(U, w' - 1)| \geq w) \geq 1 - \beta. \end{aligned} \tag{28}$$

If the oldest tuple in the uncertain window exists 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 1, here, we consider the α and β probabilistic constraints together, using a single formula (see Algorithm 2, line 5).

6.2 Pruning the similarity search space

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

6.2.1 Index-based pruning

Lian et al. [33] 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 with 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 2D 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 Eqs. (20) and (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 is close by, then the pruning is ineffective.

Multi-dimensional data is supported in a straightforward manner for low number of dimensions [33].

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 Sect. 5. 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 (Sect. 6.1). In the rest of the paper, we refer to our adaptation of methods in [33] as *Index-Match*.

6.2.2 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 redblack trees. A redblack 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 redblack 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 redblack 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 redblack 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 whether it satisfies the β condition, as discussed in Sect. 5. 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 [36].

7 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.

Table 2 Experiment parameter configuration ranges. Default values are indicated in bold

Parameter	Range
No. of samples per tuple	[5, ..., **10**, ..., 50]
Sliding window size (w)	[100, ..., **500**, ..., 1, 000]
Existential uncert. σ	[0.025, ..., **0.1**, ..., 0.25]
Value uncert. σ	[0.1, ..., **0.5**, ..., 1]
Stream length	**2000**
α Probabilistic threshold	[0.5, ..., **0.95**, ..., 1]
β Probabilistic threshold	[0.1, ..., **0.5**, ..., 0.9]
ϵ Distance threshold	Selectivity close to 0.05 %

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 2. When not explicitly stated, we use the default configuration value (shown in bold).

7.1 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 [5,13,33,41,49]. 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 [29], 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, Swedish-Leaf, Trace, and synthetic_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.

7.2 Poisson-binomial distribution approximations

In this section, we compare how the different approximations of the Poisson-binomial distribution (Sect. 4.4) 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.

[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 $\sigma = 0.25$, $x \approx 0.87$).

7.2.1 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 Eq. 26.

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

$$RMSE(n) = \sqrt{\frac{\sum_{k=0}^{n-1}(cdf_N(k) - cdf'_N(k))^2}{n}}, \tag{29}$$

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

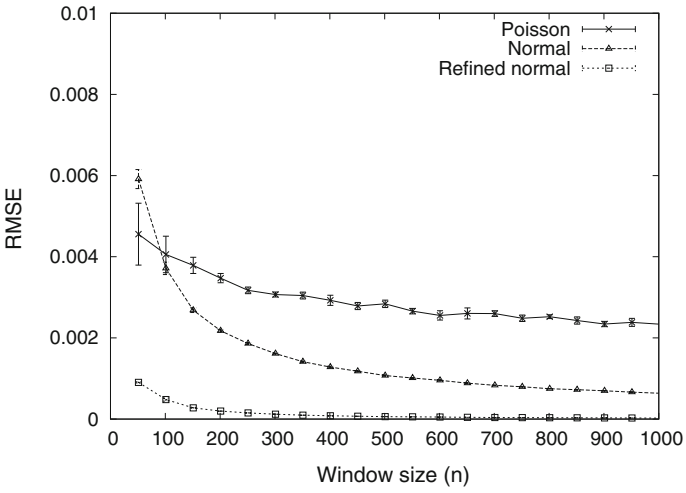

Fig. 4 RMSE of different Poisson-binomial approximations for different window sizes, and with existential uncertainty, distribution standard deviation set to 0.1 (Normal distribution). The approximation with lowest error is the Refined Normal, independent of the distribution used for assigning tuple existential uncertainties. The figure also shows the low precision of the approximations for small window sizes

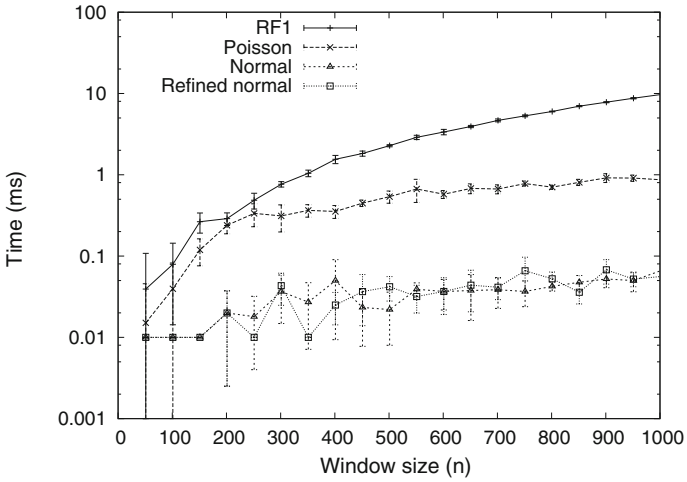

Fig. 5 Time consumed by each CDF computation under different window sizes. Refined Normal and Normal approximations provide the lowest cost

7.2.2 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 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 (Fig. 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 1,000. 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.

7.3 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 Fig. 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 window size (w) between 200 and 1,000. Given that sum_i is the value of the sum obtained using the regular sliding window and that 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 overestimating the window size, not considering the possibility that some data values do not exist in the window, which is exactly what the uncertain sliding window 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 1,800 % in the values of the output stream. For sufficiently small windows, such as $w = 200$, the sums are affected more by changes in the stream tuple values. In contrast, 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.

7.4 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.

7.4.1 Accuracy

As shown in Fig. 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 with 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:

$$1 - |Pr((u_i, v_j)_{\bowtie}) - Pr'((u_i, v_j)_{\bowtie})|, \tag{30}$$

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 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 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 (Fig. 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 Fig. 6). This means that the proposed uncertain 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.

Figures 7 and 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.

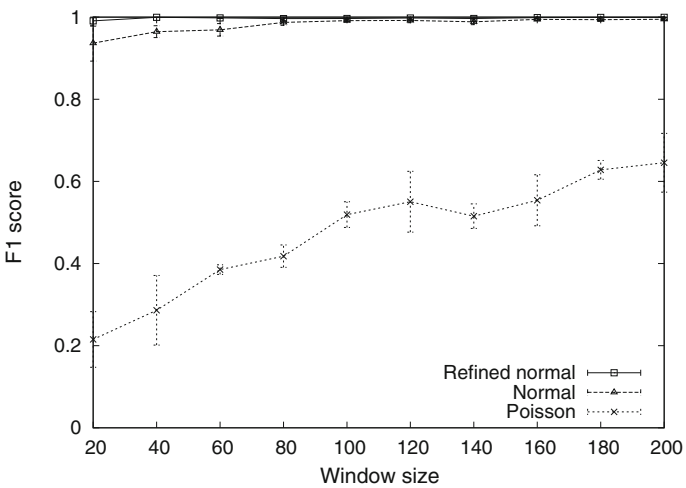

Fig. 6 F1 score for the similarity join operator when comparing the use of CDF approximations. Join using Normal and Refined Normal approximations provide results very similar to an exact solution

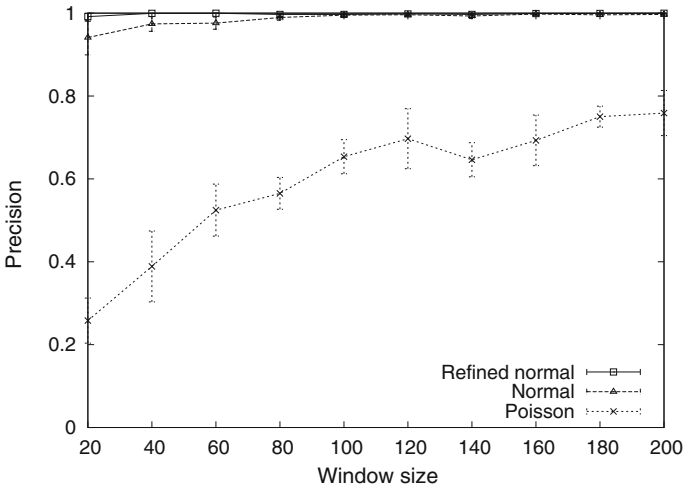

Fig. 7 Precision for the similarity join operator when using CDF approximations

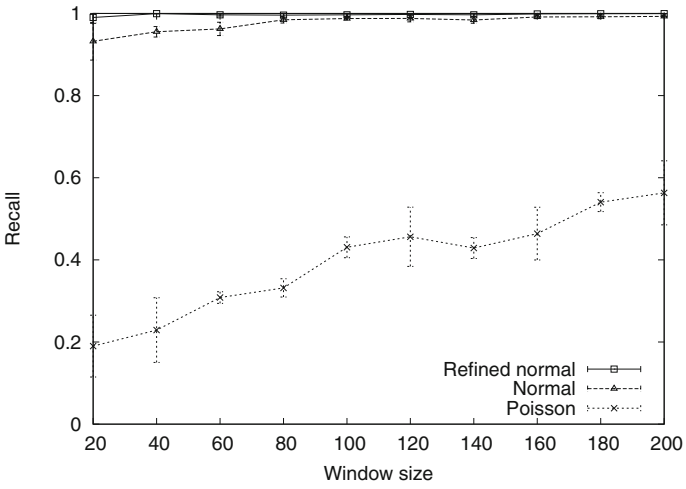

Fig. 8 Recall for the similarity join operator when using CDF approximations

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 $<0.05\,\%$.

7.4.2 Memory footprint

Memory usage for uncertain sliding windows can be measured in terms of the actual number of tuples maintained over time (w'). In Fig. 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

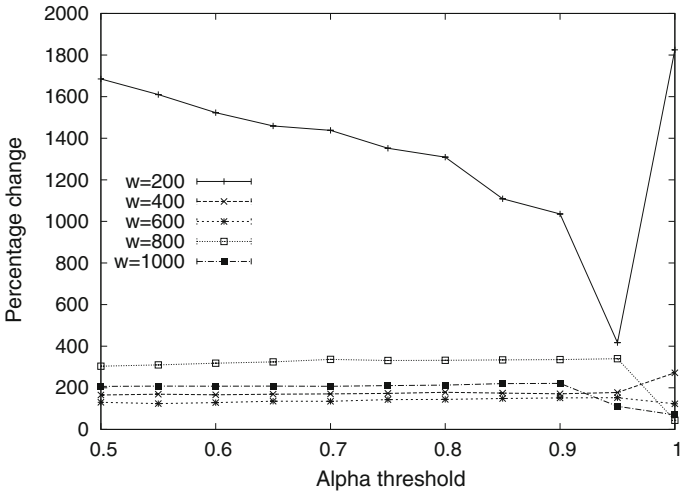

Fig. 9 Absolute percentage change of the output tuple values when substituting a regular sliding window $W(V, w)$ with an uncertain sliding window $W(V, w, \alpha)$ for different configurations of window size w when varying the α probabilistic threshold

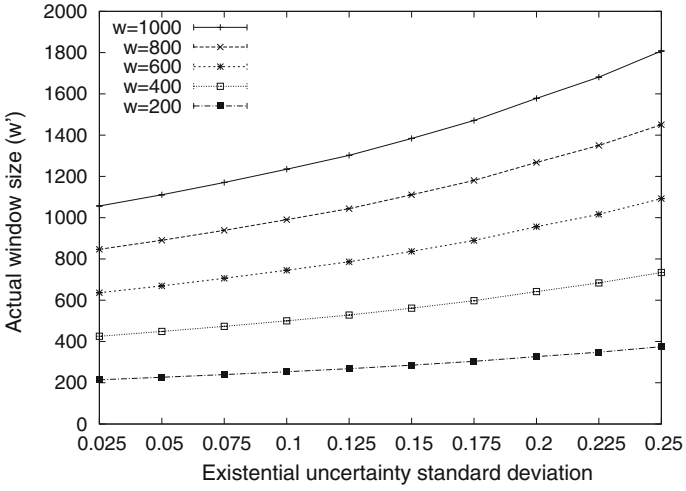

Fig. 10 Actual size of uncertain sliding windows when varying the existential uncertainty standard deviations (σ). Memory footprint increases as the existential uncertainty standard deviation increases

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 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 Fig. 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-

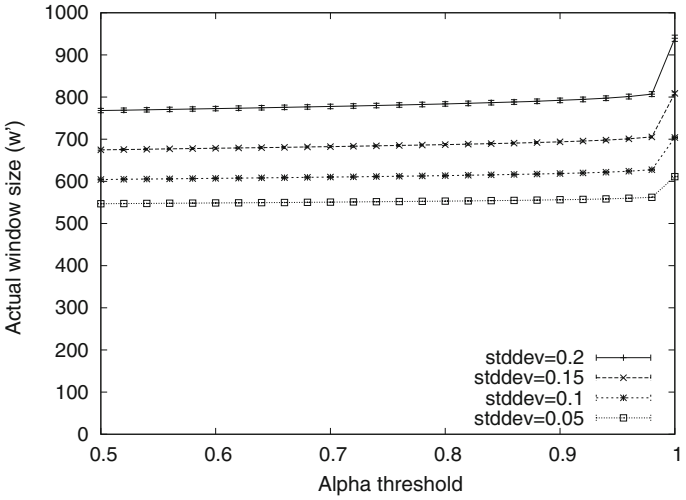

Fig. 11 Actual size of uncertain sliding windows when varying the probabilistic threshold α and the existential uncertainty σ. Window size is more sensitive to σ than α. It also presents a steep increase as α approaches to 1

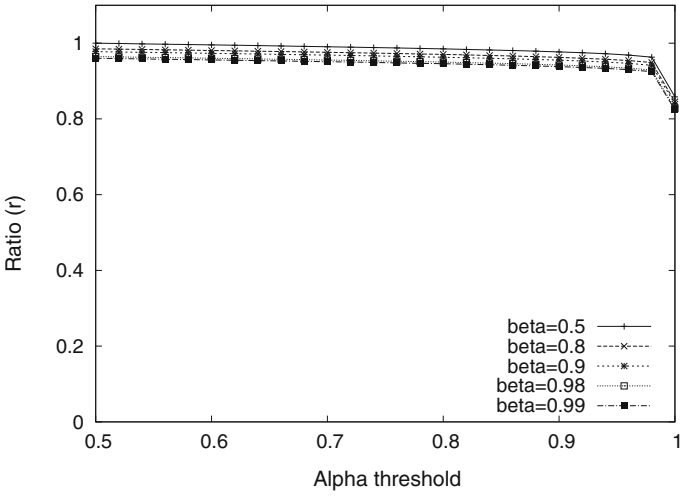

Fig. 12 Ratio of uncertain sliding window lengths maintained by eviction policies *uncert-evict-beta* and *uncert-evict* when varying the α probabilistic threshold. *uncert-evict-beta* policy maintains windows that are up to 18 % smaller

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 12 shows the results when comparing the eviction policies *uncert-evict* and *uncert-evict-beta* reported in Algorithms 1 and 2. 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 $\beta \in [0.5, 0.99]$.

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.

7.4.3 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 13 shows the results for our first experiment, in which we compare the processing time per tuple of the three algorithms 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

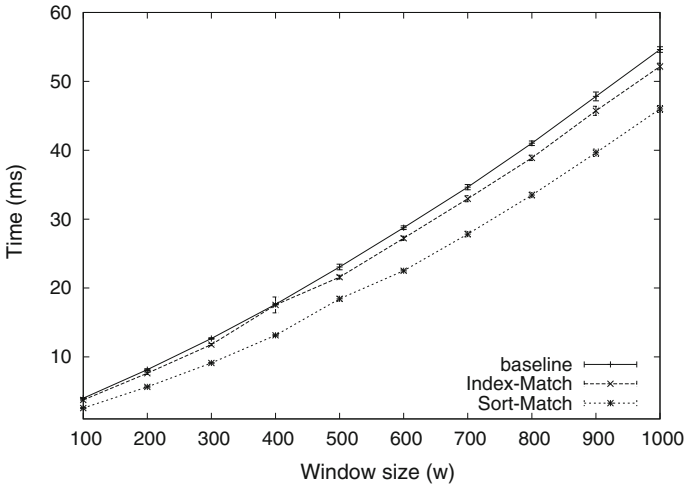

Fig. 13 Performance of pruning strategies when varying the sliding window size. *Sort-Match* outperforms *Index-Match* for different window sizes

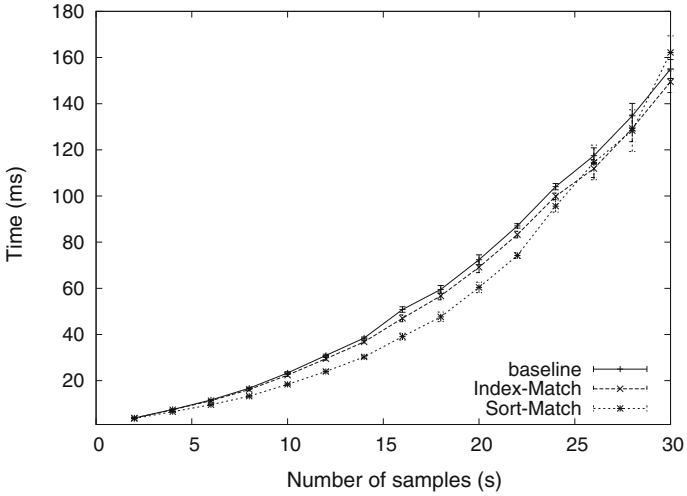

Fig. 14 Performance of pruning strategies when varying the number of samples

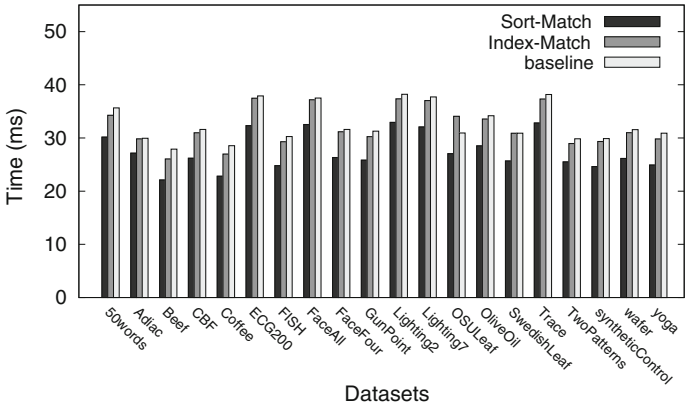

Fig. 15 Average time performance of different pruning strategies when processing different datasets

(w) between 100 and 1,000. 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 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 and 12 (e.g., in WiFi-based localization services [50], multiple reader RFID systems [53], and wireless sensor deployments [39]). 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 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*.

8 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.

8.1 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.

8.2 Integration into system S

We are working on integrating uncertain data streams, as defined in Sect. 3, into System S [21]—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 operator toolkit is being enhanced with operators that can work in the presence of value and existential uncertainty.

9 Conclusions and future work

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.

We believe that several stream mining applications can benefit from the proposed approach. Algorithms such as finding quantiles, heavy hitters, and frequent itemsets over sliding windows can be extended to support uncertain data streams by using our proposal as a foundation for more advanced analytics. We plan to carefully study the details of these research directions in our future work.

References

1. Abadi D, Ahmad Y, Balazinska M, Çetintemel U, Cherniack M, Hwang JH, Lindner W, Maskey A, Rasin A, Ryvkina E, Tatbul N, Xing Y, Zdonik S (2005) The design of the Borealis stream processing engine. In: CIDR
2. Aggarwal CC (2009) Managing and mining uncertain data. Springer, Berlin
3. Aggarwal CC, Yu PS (2008) A framework for clustering uncertain data streams. In: IEEE ICDE
4. Aßfalg J, Kriegel H-P, Kröger P, Renz M (2009) Probabilistic similarity search for uncertain time series. In: SSDBM
5. Aßfalg J, Kriegel HP, Kröger P, Renz M (2009) Probabilistic similarity search for uncertain time series. In: SSDBM, pp 435–443
6. Benjelloun O, Sarma A, Halevy A, Widom J (2006) Uldbs: databases with uncertainty and lineage. In: VLDB
7. Bernecker T, Kriegel HP, Renz M, Verhein F, Züfle A (2009) Probabilistic frequent itemset mining in uncertain databases. In: KDD, pp 119–128
8. Biem A, Bouillet E, Feng H, Ranganathan A, Riabov A, Verscheure O, Koutsopoulos H, Moran C (2010) IBM infosphere streams for scalable, real-time, intelligent transportation services. In: ACM SIGMOD
9. Calders T, Garboni C, Goethals B (2010) Approximation of frequentness probability of itemsets in uncertain data. In: Data mining (ICDM), 2010 IEEE 10th international conference on IEEE, pp 749–754
10. Cheng R, Kalashnikov D, Prabhakar S (2004) Querying imprecise data in moving object environments. IEEE Trans Knowl Data Eng 16(9):1112–1127
11. Dai X, Yiu M, Mamoulis N, Tao Y, Vaitis M (2005) Probabilistic spatial queries on existentially uncertain data. In: SSTD
12. Dallachiesa M, Aggarwal C, Palpanas T (2014) Node classification in uncertain graphs. In: SSDBM 32
13. Dallachiesa M, Nushi B, Mirylenka K, Palpanas T (2012) Uncertain time-series similarity: return to the basics. PVLDB 5(11):1662–1673
14. Dallachiesa M, Palpanas T (2013) Identifying streaming frequent items in ad hoc time windows. Data Knowl Eng 87:66–90
15. Dallachiesa M, Palpanas T, Ilyas FI (2014) Top-k nearest neighbor search in uncertain data series. Proc VLDB Endowment
16. Daskalakis C, Diakonikolas I, Servedio RA (2012) Learning poisson binomial distributions. In: Proceedings of the 44th symposium on theory of computing. ACM, pp 709–728
17. Diao Y, Li B, Liu A, Peng L, Sutton C, Tran TTL, Zink M (2009) Capturing data uncertainty in high-volume stream processing. In: CIDR

18. Fernandez M, Williams S (2010) Closed-form expression for the poisson-binomial probability density function. IEEE Trans Aerosp Electron Syst 46(2):803–817
19. Fung BCM, Wang K, Chen R, Yu PS (2010) Privacy-preserving data publishing: A survey of recent developments. ACM Comput Surv 42(4):14
20. Gedik B (2013) Generic windowing support for extensible stream processing systems. Softw Pract Exp 44(9):1105–1128
21. Gedik B, Andrade H (2012) A model-based framework for building extensible, high performance stream processing middleware and programming language for IBM infosphere streams. Softw Pract Exp 42(11):1363–1391
22. Getoor L, Friedman N, Koller D, Taskar B (2003) Learning probabilistic models of link structure. J Mach Learn Res 3:679–707
23. Halpern J (2003) Reasoning about uncertainty. MIT Press, Cambridge
24. Hirzel M, Andrade H, Gedik B, Kumar V, Losa G, Mendell M, Nasgaard H, Soulé R, Wu KL (2009) SPL language specification. Technical report RC24897. IBM Research
25. Hong Y (2011) On computing the distribution function for the sum of independent and non-identical random indicators. Technical report, Department of Statistics, Virginia Tech
26. Jayram TS, McGregor A, Muthukrishnan S, Vee E (2007) Estimating statistical aggregates on probabilistic data streams. In: ACM PODS
27. Jin C, Yi K, Chen L, Yu JX, Lin X (2008) Sliding-window top-k queries on uncertain streams. Proc VLDB Endowment 1(1):301–312
28. Kanagal B, Deshpande A (2008) Online filtering, smoothing and probabilistic modeling of streaming data. In IEEE ICDE
29. Keogh E, Xi X, Wei L, Ratanamahatana CA (2006) The UCR time series classification/clustering homepage. http://www.cs.ucr.edu/-eamonn/time_series_data
30. Kriegel H, Kunath P, Pfeifle M, Renz M (2006) Probabilistic similarity join on uncertain data. In: DASFAA
31. Kuo W, Zuo M (2003) Optimal reliability modeling: principles and applications. Wiley, New York
32. Leung CKS, Hao B (2009) Mining of frequent itemsets from streams of uncertain data. In: IEEE ICDE
33. Lian X, Chen L (2011) Similarity join processing on uncertain data streams. IEEE TKDE 23(11)
34. Liao L, Fox D, Kautz H (2007) Extracting places and activities from gps traces using hierarchical conditional random fields. Int J Rob Res 26(1):119–134
35. Liao L, Patterson DJ, Fox D, Kautz H (2007) Learning and inferring transportation routines. Artif Intell 171(5):311–331
36. Moon B, Jagadish HV, Faloutsos C, Saltz JH (2001) Analysis of the clustering properties of the hilbert space-filling curve. IEEE TKDE 13(1)
37. Neumeyer L, Robbins B, Nair A, Kesari A (2010) S4: distributed stream computing platform. In: KDCloud
38. Nybø R (2008) Time series opportunities in the petroleum industry. In: ESTSP 08, European symposium on time series prediction, Porvoo, Finland
39. Raza U, Camerra A, Murphy AL, Palpanas T, Picco GP (2012) What does model-driven data acquisition really achieve in wireless sensor networks? In: PERCOM
40. Ré C, Letchner J, Balazinska M, Suciu D (2008) Event queries on correlated probabilistic streams. In: ACM SIGMOD
41. Sarangi S, Murthy K (2010) DUST: a generalized notion of similarity between uncertain time series. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 383–392
42. Singh S, Mayfield C, Shah R, Prabhakar S, Hambrusch SE, Neville J, Cheng R (2008) Database support for probabilistic attributes and tuples. In: IEEE ICDE
43. Sow D, Biem A, Blount M, Ebling M, Verscheure O (2010) Body sensor data processing using stream computing. In: MIR
44. Sun L, Cheng R, Cheung DW, Cheng J (2010) Mining uncertain data with probabilistic guarantees. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 273–282
45. Tran TT, Peng L, Diao Y, McGregor A, Liu A (2012) Claro: modeling and processing uncertain data streams. VLDB J Int J Very Large Data Bases 21(5):651–676
46. Tran TT, Peng L, Li B, Diao Y, Liu A (2010) Pods: a new model and processing algorithms for uncertain data streams. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM, pp 159–170
47. Wang L, Cheung D, Cheng R, Lee S, Yang X (2012) Efficient mining of frequent itemsets on large uncertain databases. IEEE Trans Knowl Data Eng 24(12):2170–2183

48. Wu KL, Yu PS, Gedik B, Hildrum K, Aggarwal CC, Bouillet E, Fan W, George D, Gu X, Luo G, Wang H (2007) Challenges and experience in prototyping a multi-modal stream analytic and monitoring application on system. In: VLDB
49. Yeh M, Wu K, Yu P, Chen M (2009) PROUD: a probabilistic approach to processing similarity queries over uncertain data streams. In: Proceedings of the 12th international conference on extending database technology: advances in database technology. ACM, pp 684–695
50. Youssef M, Mah M, Agrawala A (2007) Challenges: device-free passive localization for wireless environments. In: ACM MOBICOM
51. Zhang Q, Li F, Yi K (2008) Finding frequent items in probabilistic data. In: ACM SIGMOD
52. Zhang W, Lin X, Zhang Y, Wang W, Yu JX (2009) Probabilistic skyline operator over sliding windows. In: IEEE ICDE
53. Zhou Z, Gupta H, Das SR, Zhu X (2007) Slotted scheduled tag access in multi-reader rfid systems. In: IEEE ICNP

Michele Dallachiesa is currently CTO at Skysense, Inc. He holds a Ph.D. in computer science earned at the University of Trento, where he also received his M.Sc. and B.Sc. degrees and worked as Research Fellow at the machine Learning and Intelligent OptimizatioN (LION) Group. He visited as intern the IBM T.J. Watson Research Center and the QCRI Research Lab. His research interests are in databases, data stream processing, uncertainty, and network graph mining.

Gabriela Jacques-Silva is a Research Staff Member at the IBM T.J. Watson Research Center. Prior to that, she received a Ph.D. degree in Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, an M.Sc. degree from the Federal University of Rio Grande do Sul, and a B.Sc. degree from the Federal University of Santa Maria, Brazil. Gabriela's main research interests include stream processing, fault tolerance, and distributed systems.

Buğra Gedik is currently with the Computer Engineering Department, Bilkent University, Turkey. Prior to that, he worked as a Research Staff Member at the IBM T.J. Watson Research Center. He obtained a Ph.D. degree in Computer Science from Georgia Institute of Technology, USA; and prior to that, a B.Sc. degree in Computer Science from Bilkent University, Turkey. His research interests include stream computing, distributed systems, and databases.

Kun-Lung Wu is the manager of the Data-intensive Systems and Analytics Group at the IBM T.J. Watson Research Center. He is a fellow of the IEEE and a member of the ACM. He has received Ph.D. and M.Sc. degrees in Computer Science from the University of Illinois at Urbana-Champaign, and a B.Sc. degree in Electrical Engineering from National Taiwan University, Taiwan. His research interests include stream computing, big data analytics, and databases.

Themis Palpanas is a full professor in the Paris Descartes University, France. He received his Ph.D. degree from the University of Toronto, Canada. He has worked for the University of Trento, the IBM T.J. Watson Research Center, and the University of California, Riverside. He has also visited Microsoft Research and the IBM Almaden Research Center. His research interests include data analytics, streaming data processing, and data series management.