Supporting Set Operations in Temporal-Probabilistic Databases

Katerina Papaioannou*, Martin Theobald†, Michael Böhlen*

* *Department of Computer Science, University of Zurich*
{papaioannou,boehlen}@ifi.uzh.ch

† *Computer Science & Communications Research Unit, University of Luxembourg*
martin.theobald@uni.lu

***Abstract*—In temporal-probabilistic (TP) databases, the combination of the temporal and the probabilistic dimension adds significant overhead to the computation of set operations. Although set queries are guaranteed to yield linearly sized output relations, all of the existing solutions exhibit a quadratic runtime complexity. They suffer from redundant interval comparisons and additional joins for the formation of lineage expressions. In this paper, we formally define TP set operations and study their properties. For their efficient computation, we introduce the *lineage-aware temporal window*, a mechanism that binds intervals with lineage expressions. We suggest the *lineage-aware window advancer* (LAWA) for producing *lineage-aware temporal windows*, which enable direct filtering of irrelevant intervals and finalization of output lineage expressions. This way, we compute TP set operations in linearithmic time. A series of experiments over both synthetic and real-world datasets show that (a) our approach has predictable performance, which depends only on the size of the input relations and not on the number of time intervals per fact or the overlap of the time intervals, and that (b) it outperforms state-of-the-art approaches.**

a (productsBought)

Product	λ	T	p
'milk'	a_1	[2,10)	0.3
'chips'	a_2	[4,7)	0.8
'dates'	a_3	[1,3)	0.6

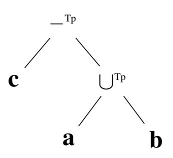
b (productsOrdered)

Product	λ	T	p
'milk'	b_1	[5,9)	0.6
'chips'	b_2	[3,6)	0.9

c (productsInStock)

Product	λ	T	p
'milk'	c_1	[1,4)	0.6
'milk'	c_2	[6,8)	0.7
'chips'	c_3	[4,5)	0.7
'chips'	c_4	[7,9)	0.8

(a) Input Relations



(b) Query Plan

Product	λ	T	p
'milk'	c_1	[1,2)	0.6
'milk'	$c_1 \wedge \neg a_1$	[2,4)	0.42
'milk'	$c_2 \wedge \neg(a_1 \vee b_1)$	[6,8)	0.196
'chips'	$c_3 \wedge \neg(a_2 \vee b_2)$	[4,5)	0.014
'chips'	c_4	[7,9)	0.8

(c) Query Result

Fig. 1: The Supermarket Application Scenario

I. Introduction

The need to manage large, temporal-probabilistic (TP) datasets appears in a wide range of applications, such as temporal predictions (e.g., weather) as well as in sensor (e.g. RFID) and experimental data, due to erroneous per-time-point measurements. The combination of the temporal and the probabilistic dimension in databases requires that the result of relational algebraic operators complies with the semantics of each dimension. Probabilistic databases rely on the *possible-worlds semantics* to define for which instances of the probabilistic database an answer tuple is valid. Conversely, temporal databases use the *sequenced semantics* to define at which time points (i.e., snapshots of the temporal database) an answer tuple is valid. The possible-worlds and the sequenced semantics very nicely complement each other, since they both employ the notion of *data lineage* to guarantee a closed and complete representation model for temporal, uncertain data.

In this paper, we introduce a *sequenced* TP data model and, under this model, we define and implement TP set operations. In the following example, we illustrate their usefulness in an application involving temporal predictions.

Example 1: Consider the supermarket application of Figure 1. The supermarket records data related to purchases of clients (**a**), online shopping carts (**b**), and inventory (**c**). At each time point (e.g., a day), the supermarket aims at predicting the products that clients want to buy or order versus those that it has in stock. For example, tuple *('milk', a_1, [2,10), 0.3)* captures that, at each day from the 2nd to the 10th of the month, "milk is bought" with probability *0.3*. There is no other tuple in **a** that predicts the probability of buying 'milk' over an interval overlapping with $[2,10)$. Assume the supermarket wants to determine, at each time point, the probability that a product is in stock but no client wants to order or buy this product. The corresponding query is $Q = \mathbf{c} -^{\mathrm{Tp}} (\mathbf{a} \cup^{\mathrm{Tp}} \mathbf{b})$, i.e., the union of relations **a** and **b**, followed by a difference with relation **c** (see Fig. 1b). Answer tuple *('milk', $c_1 \wedge \neg a_1$, [2,4), 0.42)* (see Fig. 1c) expresses that, with probability *0.42*, *'milk'* is in stock but is not ordered or bought during interval $[2,4)$.

TP set operations are of great interest due to the overhead added in their computation when combining the temporal and probabilistic dimension. They are however a class of operations that have received little attention so far: they have not been explicitly defined in existing TP approaches [1], with TP set difference not being supported at all. Existing temporal techniques, on the other hand, suffer from two main drawbacks. First, approaches used for the computation of temporal set operations [2], [3] replicate input tuples with adjusted intervals before the actual algebraic operations are applied. They rely on joins with inequality conditions that have quadratic complexity due to unproductive tuple comparisons. Second, stitching lineage expressions to the output tuples in a relational manner requires additional joins in comparison to the set operations that are available in current temporal database implementations. As far as existing temporal-probabilistic [1] or purely probabilistic [4] approaches are concerned, set operations are reduced to joins, since their computation not only requires the comparison of relational attributes among the input tuples, but also the combination of their lineage expressions. However, the computation of TP set operations under a sequenced TP data model requires more sophisticated solutions for the computation of output intervals than the use of temporal predicates in joins.

Outline & Contributions.

- We propose a *sequenced temporal-probabilistic data model* that complies with both the sequenced semantics from temporal databases [3], [5] and the possible-worlds semantics from probabilistic databases [6], [7].
- We formally define *TP set operations* and study the properties of TP set queries under this model. To our knowledge, TP set queries have not previously been investigated under a sequenced temporal-probabilistic model.
- We introduce a novel *lineage-aware temporal window*, a mechanism that binds an interval with the lineages of the tuples valid during the interval. Such windows are produced using the *lineage-aware window advancer* (LAWA) and they enable finalizing lineages and filtering out irrelevant intervals directly at the time of their creation. This way, LAWA guarantees $O(n \log n)$ worst-case complexity for the computation of TP set operations, improving over existing implementations with quadratic complexity.
- We experimentally demonstrate that LAWA is the only approach among both temporal and probabilistic databases that *does not deteriorate in performance* as the data history grows. In contrast to existing techniques, our solution does not depend on the characteristics of the dataset (such as the number of intervals per fact, or the overlap among intervals), but only on the size of the input relations.

The remainder of this paper is organized as follows. Section II provides an overview of related works on temporal and probabilistic databases with a focus on set operations. Section III introduces our TP data model, while Section IV defines the model's query semantics. Section V defines TP set operations over duplicate-free input relations. Section VI introduces the lineage-aware temporal window and our implementation of TP set queries. Section VII presents a comprehensive performance study that compares our implementation of TP set operations with existing timestamp-adjustment and lineage-computation approaches. Section VIII finally concludes the paper.

II. Related Work

We next review related approaches from both temporal and probabilistic databases and explain their limitations in terms of supporting TP set operations. Set difference, for example, has received little attention in temporal databases and can only be computed using the generic normalization operator [3]. Under a combined temporal and probabilistic data model, there is currently no solution that supports set difference.

Temporal Set Operations. In temporal databases, the result of a temporal set operation op^T is defined as the result of applying op over a sequence of atemporal instances (the so-called snapshots) of the input relations—a key concept in temporal databases termed *snapshot reducibility* [8], [9], [10]. Maximal intervals are produced by merging consecutive time points to which the same input tuples have contributed (*change preservation*). Dignös et al. [3], [2] use *data lineage* to guarantee change preservation for all relational operations under a sequenced semantics. They adapt the *Normalization* operator, introduced by Toman et al. [11], to compute temporal set queries. Intuitively, the normalization $N(\mathbf{r},\mathbf{s})$ of a relation $\mathbf{r}$ based on another relation $\mathbf{s}$ replicates the tuples of $\mathbf{r}$ and assigns new time intervals to them. The new intervals are obtained by splitting the original intervals based on tuples of $\mathbf{s}$ with which they overlap. Normalization is a generic operator that subsequently requires an outer join of $\mathbf{r}$ and $\mathbf{s}$ with quadratic complexity. Since it is not symmetric, it has to be computed once for each of the two input relations [3], [2].

Temporal joins can be used for the computation of TP set intersection. Efficient solutions for temporal joins have been widely discussed in the literature [12], [13], [14], [15]. Specific solutions either partition the data [15] in ways that are not beneficial for our case, since TP relations are duplicate-free (see Section III), or they require fixed-length input schemas [14]. *Timeline Index* (TI) is a data structure introduced by Kaufmann et al. [12], [16] to efficiently compute temporal aggregation, join and time-travel operations. TI of relation $\mathbf{r}$ maps each start or end point in $\mathbf{r}$ to a list of ids of tuples that start or end at this time point. *Timeline Join* (TJ) is applied on the indexes created for the input relations and implements a combination of a merge- and a hash-join. The performance of TJ suffers because the original tuples need to be fetched both for the application of a filtering condition and for the creation of the output tuples.

Overlap Interval Partitioning (OIP) by Dignös et al. [13] is designed to compute a join $\mathbf{r} \bowtie^T \mathbf{s}$ among tuples with overlapping time intervals. Initially, OIP splits the time domain into k granules of equal size. Adjacent granules are combined to form the partitions of an input relation $\mathbf{r}$ so that each tuple in $\mathbf{r}$ is assigned to the smallest partition into which it fits. In order to compute the overlap join, the overlapping partitions of $\mathbf{r}$ and $\mathbf{s}$ are identified (fast), and then a nested loop is performed to join the tuples of these partitions (slow). This approach finds all pairs of tuples (r, s), for $r \in \mathbf{r}$ and $s \in \mathbf{s}$, with overlapping time intervals. Although OIP can be extended to apply additional filtering conditions, e.g., equality conditions on the atemporal attributes of the tuples that are paired, its performance deteriorates when the condition has low selectivity (see Section VII).

Sweeping-based approaches, finally, have been widely used for the computation of overlap joins [14], [17] in temporal settings. A sweepline moves over all start and end points of tuples, and determines, for each time point, the tuples of both input relations that are valid. These approaches cannot directly be applied for the computation of TP set operations. First, they generally do not consider join conditions on the non-temporal attributes. Second, they support set intersection but cannot produce all output tuples needed for set difference and union. The creation of output intervals through the tuples that the sweepline intersects is not sufficient for these two set operations.

Probabilistic Set Operations. In probabilistic databases, the result of a probabilistic set operation op^p is defined as the result of applying op over the set of all possible instances of the input relations. The Trio system [18] was among the first to recognize *data lineage*, in the form of a Boolean formula, as a means to capture the possible instances at which an output tuple is valid. In an effort to provide a *closed and complete* representation model for uncertain relational data, they introduced *Uncertainty and Lineage Databases* (ULDBs) [19]. The algebraic operators are modified to compute the lineage of the result tuples in a ULDB, thus capturing all information needed for computing query answers and their probabilities. Recently, Fink et al. [4], [20] reduced the computation of probabilistic algebraic operations to conventional operations so that these can be performed using a DBMS, rather than by an application layer built on top of it.

Temporal-Probabilistic Set Operations. Dylla et al. [1] introduced a closed and complete TP database model, coined TPDB, based on existing temporal and probabilistic concepts. Query processing is performed in two steps. The first step, grounding, evaluates a chosen deduction rule (formulated in Datalog with additional time variables and temporal predicates) and computes the lineage expressions of the deduced tuples. The second step, deduplication, removes the duplicates that could occur in the grounding step by adjusting their intervals. Although the TPDB data model is generic, the query processing is hindered by the grounding step (as shown in Section VII) and cannot cover operations whose results include subintervals that are only present in one of the two input relations, as is the case for TP set difference. Moreover, TBDB does not fulfill all requirements of a sequenced semantics, as it is common in temporal databases.

III. Data Model & Notation

We denote a **temporal-probabilistic schema** by $R^{\text{Tp}}(F, \lambda, T, p)$, where $F = (A_1, A_2, \ldots, A_m)$ is an ordered set of attributes, and each attribute A_i is assigned to a fixed domain Ω_i. λ is a Boolean formula corresponding to a lineage expression. T is a *temporal attribute* with domain $\Omega^T \times \Omega^T$, where Ω^T is a finite and ordered set of *time points*. p is a *probabilistic attribute* with domain $\Omega^p = (0,1] \subset \mathbb{R}$. A **temporal-probabilistic relation r** over R^{Tp} is a finite set of tuples. Each tuple $r \in \mathbf{r}$ is an ordered set of values in the appropriate domains. The value of attribute A_i of r is denoted by $r.A_i$. The conventional attributes $F = (A_1, A_2, \ldots, A_m)$ of tuple r form a so-called *fact*, and we write $r.F$ to denote the fact f captured by tuple r. For example, the base tuple ('milk', a_1, $[2,10)$, 0.3) of relation **a** (see Fig. 1a) includes the fact $a_1.F$ = ('milk'), the lineage expression $a_1.\lambda = a_1$, the time interval $a_1.T = [2,10)$, and the probability value $a_1.p = 0.3$. The temporal-probabilistic annotations of the schema express that (i) $a_1 = \mathit{true}$ with probability $a_1.p$ for every time point in $a_1.T$, (ii) $a_1 = \mathit{false}$ with probability $1 - a_1.p$ for every time point in $a_1.T$, (iii) and a_1 is always *false* outside $a_1.T$.

By following conventions from [1], [2], [3], [21], we assume duplicate-free input and output relations. Formally, a temporal-probabilistic relation **r** is **duplicate-free** iff $\forall r, r' \in \mathbf{r}(r \neq r' \Rightarrow r.F \neq r'.F \vee r.T \cap r'.T = \emptyset))$. In other words, the intervals of any two tuples of **r** with the same fact f do not overlap.

A **lineage expression** λ is a Boolean formula, consisting of tuple identifiers and the three Boolean connectives $\neg$ ("not"), $\wedge$ ("and") and $\vee$ ("or"). Tuple identifiers represent Boolean random variables among which we assume independence [1], [21], [22]). For a base tuple r, $r.\lambda$ is an atomic expression consisting of just r itself. For a result tuple $\tilde{r}$ derived from one or more TP operations, $\tilde{r}.\lambda$ is a Boolean expression as defined above. For a result tuple, lineage is determined by the temporal-probabilistic operators (formally defined in Section IV) that were applied to derive that tuple from the base tuples. The probability of a result tuple is computed via a probabilistic valuation of the tuple's lineage expression, using either exact (see, e.g., [22], [23], [24]) or approximate (see, e.g., [25], [26], [27], [28], [29]) algorithms. For example, in the result relation of Fig. 1c, the lineage $c_1 \wedge \neg a_1$ yields a marginal probability of $0.6 \cdot (1 - 0.3) = 0.42$ by assuming independence among the base tuples c_1 and a_1 (see Fig. 1a). Further, we write $\lambda_t^{\mathbf{r},f}$ to refer to the lineage expression of a tuple in relation **r** with fact f that is valid at time point t. When there are no tuples in **r** with fact f at time point t, we write $\lambda_t^{\mathbf{r},f} = \texttt{null}$.

IV. Query Semantics

For our combined query semantics, we adopt both the *sequenced semantics* [5], widely used for the temporal dimension, and the *possible-worlds semantics* [7], commonly used for the probabilistic dimension. The sequenced semantics is consistent with viewing a temporal database as a sequence of atemporal databases (the "snapshots"), one for each time point t in Ω^T. Conceptually, query evaluation then resolves to evaluating a query against each of these snapshots and producing maximal output intervals according to time points with equivalent *data lineage*. Thus, an output interval consists of time points, in which the corresponding fact has been derived based on the same input tuples. The possible-worlds semantics defines a probabilistic database as a probability distribution over a finite set of possible states (aka. "worlds"), in which the probabilistic database could be. Conceptually, a query is evaluated against each of the possible worlds. The marginal probability of an answer tuple then is defined as the sum of the possible-worlds probabilities, for which the answer tuple exists. Data lineage [19], [18], in the form of a Boolean expression, serves as a concise condition that is satisfied over the possible worlds in which each answer tuple exists.

The query semantics of our sequenced TP data model is based on an intriguing analogy between the temporal and probabilistic semantics: rather than iterating over snapshots or possible worlds, they both use the notion of data lineage to define their operational semantics. Given a TP relation $\mathbf{r}$, a tuple $r \in \mathbf{r}$ is valid at every time point t included in its time interval $r.T$ with probability $r.p$. Thus, all the tuples of a TP relation $\mathbf{r}$ that are valid at time point t with a given probability are included in the *probabilistic snapshot* of $\mathbf{r}$ at t. Specifically, we obtain the probabilistic snapshot of a TP relation $\mathbf{r}$ with schema $R^{\text{Tp}} = (F, \lambda, T, p)$ at time point t by applying the *timeslice operator* τ_t^{p}, which is defined as:

$$\tau_t^{\text{p}}(\mathbf{r}^{\text{Tp}}) = \{(r.F, r.\lambda, [t, t+1), r.p) \mid r \in \mathbf{r} \wedge t \in r.T\}$$

Definition 1: **(TP Snapshot Reducibility)** *Let* $\mathbf{r}_1, \ldots, \mathbf{r}_m$ *be a set of TP relations, let* op^{Tp} *be an m-ary temporal-probabilistic operator, let* op^{p} *be the corresponding probabilistic operator, let* Ω^T *be the time domain, and let* $\tau_t^{p}(\mathbf{r})$ *be the timeslice operator. The operator* op^{Tp} *is* snapshot reducible *to* op^{p} *iff, for all* $t \in \Omega^T$*, it holds that:*

$$\tau_t^{\text{p}}(op^{\text{Tp}}(\mathbf{r}_1, \ldots, \mathbf{r}_m)) \equiv op^{\text{p}}(\tau_t^{\text{p}}(\mathbf{r}_1), \ldots, \tau_t^{\text{p}}(\mathbf{r}_m))$$

Snapshot reducibility states that a probabilistic snapshot of the result of an m-ary TP operation $op^{\text{Tp}}(\mathbf{r}_1, \ldots, \mathbf{r}_m)$ at any time point t is equivalent to the result derived from the corresponding probabilistic operation op^{p} on the probabilistic snapshots of the input relations at t. Applying an atemporal operation over all probabilistic snapshots thus is consistent with snapshot reducibility in temporal databases and implies that the result at any time point t, both in terms of probability values and facts, is determined only by the input tuples that are valid at t. The application of op^{p} guarantees that the computations at each time point will yield Boolean lineage expressions that are consistent with the possible-worlds semantics [18], [19].

As example, consider the query of Fig. 1b over the relations of Fig. 1a. According to the lineage expression of tuple ('milk', [2,4), $c_1 \wedge \neg a_1$, 0.42), at $t = 2$, the fact 'milk' has been derived from the input tuples a_1 and c_1, i.e., the only input tuples valid at this time point. Since the probability of 'milk' at $t = 2$ is only affected by the probabilities of a_1 and c_1, it can be computed based on the lineage expression $c_1 \wedge \neg a_1$.

Definition 2: **(TP Change Preservation)** *Let* $\mathbf{r}_1, \ldots, \mathbf{r}_m$ *be a set of TP relations, let* op^{Tp} *be an m-ary temporal-probabilistic operator, and let* $u.T_s$*,* $u.T_e$ *denote the start and end points of an interval associated with a tuple u. For each tuple* $u \in \mathbf{u}$*, where* $\mathbf{u} = op^{Tp}(\mathbf{r}_1, \ldots, \mathbf{r}_m)$*, it holds that:*

$$\forall t, t' \in u.T(\lambda_t^{\mathbf{u}, u.F} \equiv \lambda_{t'}^{\mathbf{u}, u.F}) \wedge$$
$$\nexists u' \in \mathbf{u}((u'.T_e = u.T_s \vee u'.T_s = u.T_e) \wedge (u'.\lambda \equiv u.\lambda))$$

Intuitively, change preservation ensures that only consecutive time points of tuples with equivalent lineage expressions are grouped into intervals. For example, the output tuples ('milk', [1,2), c_1, 0.6) and ('milk', [2,4), $c_1 \wedge \neg a_1$, 0.42) were not merged into the interval [1,4), since they do not have equivalent lineages. Change preservation guarantees that a fact is valid over the same possible worlds with maximal intervals. The first line of Def. 2 ensures that the lineage expression at all time points in the interval of a result tuple is the same. The second line ensures that the time intervals produced by coalescing time points with the equivalent lineage expressions are maximal. [1]

V. TP Set Operations & Queries

A. TP Set Operations

In TP databases, the result of a *TP set union* includes, at each time point $t \in \Omega^T$, the facts for which there is a non-zero probability to be in $\mathbf{r}$ or in $\mathbf{s}$; the result of a *TP set intersection* includes, at each time point, the facts for which there is a non-zero probability to be in $\mathbf{r}$ and in $\mathbf{s}$; and the result of a *TP set difference* between two TP relations $\mathbf{r}$ and $\mathbf{s}$ includes, at each time point, the facts for which there is a non-zero probability to be in $\mathbf{r}$ and not in $\mathbf{s}$.

Definition 3: **(TP Set Operations)** *Let* $\mathbf{r}$ *and* $\mathbf{s}$ *be temporal-probabilistic relations with schema (F, λ, T, p), and let* $\lambda_t^{\mathbf{r},f}$ *denote the lineage expression of the tuple in relation* $\mathbf{r}$ *that includes fact f and is valid at time point t. Given a result tuple* $\tilde{r}$ *and the lineage-concatenation functions depicted in Table I, we define the three TP set operations* $\mathbf{r} \cup^{Tp} \mathbf{s}$*,* $\mathbf{r} \cap^{Tp} \mathbf{s}$ *and* $\mathbf{r} -^{Tp} \mathbf{s}$ *as follows:*

$$\tilde{r} \in \mathbf{r} \cup^{\text{Tp}} \mathbf{s} \iff \forall t \in \tilde{r}.T((\lambda_t^{\mathbf{r},\tilde{r}.F} \neq \texttt{null} \ \vee \ \lambda_t^{\mathbf{s},\tilde{r}.F} \neq \texttt{null}) \wedge$$
$$\tilde{r}.\lambda \equiv \mathbf{or}(\lambda_t^{\mathbf{r},\tilde{r}.F}, \lambda_t^{\mathbf{s},\tilde{r}.F})) \wedge$$
$$\forall t' \notin \tilde{r}.T(\tilde{r}.\lambda \not\equiv \mathbf{or}(\lambda_{t'}^{\mathbf{r},\tilde{r}.F}, \lambda_{t'}^{\mathbf{s},\tilde{r}.F}))$$

$$\tilde{r} \in \mathbf{r} \cap^{\text{Tp}} \mathbf{s} \iff \forall t \in \tilde{r}.T(\lambda_t^{\mathbf{r},\tilde{r}.F} \neq \texttt{null} \wedge \lambda_t^{\mathbf{s},\tilde{r}.F} \neq \texttt{null} \wedge$$
$$\tilde{r}.\lambda \equiv \mathbf{and}(\lambda_t^{\mathbf{r},\tilde{r}.F}, \lambda_t^{\mathbf{s},\tilde{r}.F})) \wedge$$
$$\forall t' \notin \tilde{r}.T(\tilde{r}.\lambda \not\equiv \mathbf{and}(\lambda_{t'}^{\mathbf{r},\tilde{r}.F}, \lambda_{t'}^{\mathbf{s},\tilde{r}.F}))$$

$$\tilde{r} \in \mathbf{r} -^{\text{Tp}} \mathbf{s} \iff \forall t \in \tilde{r}.T(\lambda_t^{\mathbf{r},\tilde{r}.F} \neq \texttt{null} \wedge$$
$$\tilde{r}.\lambda \equiv \mathbf{andNot}(\lambda_t^{\mathbf{r},\tilde{r}.F}, \lambda_t^{\mathbf{s},\tilde{r}.F})) \wedge$$
$$\forall t' \notin \tilde{r}.T(\tilde{r}.\lambda \not\equiv \mathbf{andNot}(\lambda_{t'}^{\mathbf{r},\tilde{r}.F}, \lambda_{t'}^{\mathbf{s},\tilde{r}.F}))$$

The above definition of TP set operations specifies the intervals and lineage expressions of a result tuple $\tilde{r}$. The first line of the definition of each operation relates to Def. 1. It states that, at any time point $t \in \tilde{r}.T$, fact $\tilde{r}.F$ must be included in the corresponding input tuples from $\mathbf{r}$ and $\mathbf{s}$. Consequently, the lineage expression of the output tuple $\tilde{r}$ at each time point $t \in \tilde{r}.T$ (cf. second line) is computed based on the same input tuples, according to the lineage-concatenating functions of Table I. In the case of set union, there must exist at least one tuple in either one of the two input relations that also

[1]Rather than performing logical equivalence checks among Boolean formulas, which are co-NP-complete, we resort to a syntactic comparison of the lineage sets in our implementation.

TABLE I: Definition of lineage-concatenation functions.

$\textbf{\textit{and}}(\lambda_1,\lambda_2)$	$= (\lambda_1)\wedge(\lambda_2)$
$\textbf{\textit{andNot}}(\lambda_1,\lambda_2)$	$= \begin{cases} (\lambda_1) & \text{if } \lambda_2 = \texttt{null} \\ (\lambda_1)\wedge\neg(\lambda_2) & \text{otherwise} \end{cases}$
$\textbf{\textit{or}}(\lambda_1,\lambda_2)$	$= \begin{cases} (\lambda_1) & \text{if } \lambda_2 = \texttt{null} \\ (\lambda_2) & \text{if } \lambda_1 = \texttt{null} \\ (\lambda_1)\vee(\lambda_2) & \text{otherwise} \end{cases}$

includes $\tilde{r}.F$ over $\tilde{r}.T$. For set intersection, there must exist corresponding tuples in both input relations. For set difference, an output tuple is produced at all time points t, at which there exists a tuple of the left relation r that is valid at t in $r.T$. This happens in two cases: (a) if a fact f is included in a tuple of $\mathbf{r}$ but in no tuple in $\mathbf{s}$, and (b) if a fact f is included in a tuple of $\mathbf{r}$ but, with a probability of less than 1, also in a tuple of $\mathbf{s}$. The first case resembles the definition of temporal set difference, where, at each time point in the output, there exist facts that are included in tuples of $\mathbf{r}$ and not in tuples of $\mathbf{s}$. The second case occurs due to the probabilistic dimension. The result of a probabilistic set difference between $\mathbf{r}$ and $\mathbf{s}$ includes all facts, which have a non-zero probability to be in $\mathbf{r}$ and not in $\mathbf{s}$.

Example 2: Figure 2 shows the relations $\mathbf{a}$ and $\mathbf{c}$ of Fig. 1a as well as selected output tuples of $\mathbf{a} -^{\text{Tp}} \mathbf{c}$. Different colors are used for different facts: green is used for 'milk', blue for 'dates' and red for 'chips'. Output tuples are drawn below the time axis. For example, the output tuple ('milk', $a_1 \wedge \neg c_2$, $[6,8)$, 0.09) satisfies Def. 3: for all time points in $[6,8)$, it holds that $\lambda_t^{\mathbf{a},\text{'milk'}} = a_1 \neq \texttt{null}$ and $\lambda_t^{\mathbf{c},\text{'milk'}} = c_2$. Thus, $\forall t \in [6,8)$, $\textbf{andNot}(\lambda_t^{\mathbf{a},\text{'milk'}}, \lambda_t^{\mathbf{c},\text{'milk'}}) \equiv a_1 \wedge \neg c_2$.

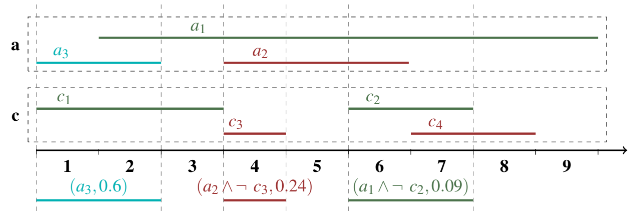

a
c
a_1
a_3
a_2
c_1
c_2
c_3
c_4
1
2
3
4
5
6
7
8
9
$(a_3, 0.6)$
$(a_2 \wedge \neg\, c_3, 0.24)$
$(a_1 \wedge \neg\, c_2, 0.09)$

Fig. 2: Selected output tuples.

The third line of the definition of each TP set operator is a direct consequence of Def. 2. It guarantees that, when merging consecutive time points into an interval, we consider only the ones for which the condition in the first line is satisfied. In other words, a new interval is created whenever there is a change in the validity of a tuple from either $\mathbf{r}$ or $\mathbf{s}$ at the currently considered time point. In Example 2, at time points $t = 5$ and $t = 8$, $\lambda_t^{\mathbf{a},\text{'milk'}} = a_1$ and $\lambda_t^{\mathbf{c},\text{'milk'}} = \texttt{null}$. Thus, outside the interval $[6,8)$ of tuple ('milk', [6,8), $a_1 \wedge \neg c_2$, 0.09), there are no time points for which $\textbf{andNot}(\lambda_t^{\mathbf{a},\text{'milk'}}, \lambda_t^{\mathbf{c},\text{'milk'}}) \equiv a_1 \wedge \neg c_2$. Fig. 3 shows the result of all TP set operations between relations $\mathbf{a}$ and $\mathbf{c}$ in Fig. 1a.

$\mathbf{a} \cup^{\text{Tp}} \mathbf{c}$

Product	λ	T	p
'milk'	c_1	[1,2)	0.6
'milk'	$a_1 \vee c_1$	[2,4)	0.72
'milk'	a_1	[4,6)	0.3
'milk'	$a_1 \vee c_2$	[6,8)	0.79
'milk'	a_1	[8,10)	0.3
'chips'	$a_2 \vee c_3$	[4,5)	0.94
'chips'	a_2	[5,7)	0.8
'chips'	c_4	[7,9)	0.8
'dates'	a_3	[1,3)	0.6

$\mathbf{a} -^{\text{Tp}} \mathbf{c}$

Product	λ	T	p
'milk'	$a_1 \wedge \neg c_1$	[2,4)	0.12
'milk'	a_1	[4,6)	0.3
'milk'	$a_1 \wedge \neg c_2$	[6,8)	0.09
'milk'	a_1	[8,10)	0.3
'chips'	$a_2 \wedge \neg c_3$	[4,5)	0.24
'chips'	a_2	[5,7)	0.8
'dates'	a_3	[1,3)	0.6

$\mathbf{a} \cap^{\text{Tp}} \mathbf{c}$

Product	λ	T	p
'milk'	$a_1 \wedge c_1$	[2,4)	0.18
'milk'	$a_1 \wedge c_2$	[6,8)	0.21
'chips'	$a_2 \wedge c_3$	[4,5)	0.56

Fig. 3: TP set operations computed for the relations of Fig. 1a.

B. *TP Set Queries & Complexity*

Having defined TP set operations, we can now move on to TP set queries, which are expressions of TP set operations over a set of TP relations.

Definition 4: **(TP Set Query)** *Let* $\mathbf{r}_1,\ldots,\mathbf{r}_m$ *be a set of duplicate-free TP relations. A* TP set query Q *is any expression of TP set operators that adheres to the following grammar:*

$$Q ::= \mathbf{r}_i \mid Q \cup^{\text{Tp}} Q \mid Q \cap^{\text{Tp}} Q \mid Q -^{\text{Tp}} Q \mid (Q)$$

The following theorem and corollary establish an interesting relationship between *safe queries* [22], [23] in probabilistic databases and tractable queries in our TP setting. The theorem is based on the observation that repeated applications of TP set operations create regular lineage expressions, which are in *one-occurrence form* (1OF) [7] if none of the input relations occurs more than once in a TP set query. Formally, a formula is in 1OF iff no tuple identifier occurs more than once in the formula. Correspondingly, we call a TP set query Q *non-repeating* iff every input relation $\mathbf{r}_i$ occurs at most once in Q.

Theorem 1: Any non-repeating TP set query Q *over duplicate-free TP relations yields lineage formulas in 1OF.*

Proof: Consider a TP set operation over two TP relations $\mathbf{r}$ and $\mathbf{s}$, both having schema (F, λ, T, p). Since $\mathbf{r}$ and $\mathbf{s}$ are duplicate-free, we cannot have two tuples in either $\mathbf{r}$ or $\mathbf{s}$ that share the same fact at overlapping time intervals. Assume we have n_1 tuples in $\mathbf{r}$ and n_2 tuples in $\mathbf{s}$ with the same fact f, but each with non-overlapping time intervals. Then, for $n = n_1 + n_2$ input intervals, we can at most obtain $2n - 1$ output intervals. According to change preservation (Def. 2), we create the same amount of output tuples, one for each output interval and each with a different combination of tuple identifiers in their lineage (Def. 3). Next, inductively, during any further application of a TP set operation (over non-repeating subgoals), change preservation will only merge two consecutive time intervals iff their lineages are equivalent. This cannot occur, since all of the lineages that are created by an individual TP set operator are different. That is, for a non-repeating TP set query, each tuple identifier can occur at most once in the lineage of a result tuple, which means that the lineages are in 1OF. ■

Corollary 1: Any non-repeating TP set query Q over duplicate-free TP relations has PTIME data complexity.

The proof of the corollary follows directly from Theorem 1, since computing the marginal probability of a Boolean formula in 1OF can be done in linear time in the size of the formula for independent random variables [7]. Also, all temporal alignment operations are of polynomial complexity (see [2], [3] as well as the algorithms in Section VI).

The above class of non-repeating TP set queries over duplicate-free TP relations nicely complements the dichotomy theorem [22], [23] established for unions of conjunctive queries (UCQs) in probabilistic databases. Each individual TP set operation over two compatible relation schemas resolves to (a union of) at most two conjunctive queries, in which no intermediate duplicates due to a projection onto a subset of attributes in F may arise. Although repeated applications of TP set operations in a query do not necessarily form UCQs, the overall query remains hierarchical [7], since all attributes in F are propagated through the operations. Change preservation, on the other hand, which is required for a sequenced temporal semantics, preserves these complexity considerations by merging only intervals with equivalent lineage expressions into a single output interval. TP set queries with repeating subgoals however remain #P-hard as shown in [30] (consider, e.g., the query $(\mathbf{r}_1 \cup^{\text{Tp}} \mathbf{r}_2) -^{\text{Tp}} (\mathbf{r}_1 \cap^{\text{Tp}} \mathbf{r}_3)$).

VI. Implementation

In this section, we introduce the *lineage-aware temporal window*, a novel mechanism that enables finalizing output lineages and filtering out intervals when they are produced, thus avoiding redundant computations that occur when these two steps are decoupled [1], [2]. We present the *lineage-aware window-advancer* (LAWA), an algorithm to produce lineage-aware temporal windows, and we show that it can be used to improve over the quadratic complexity of existing approaches that compute TP set operations.

A. Lineage-Aware Temporal Window

A *lineage-aware temporal window* is a mechanism that associates candidate output intervals with the lineage expressions of the valid input tuples. It has schema (F, `winTs`, `winTe`, $\lambda_{\mathtt{r}}$, $\lambda_{\mathtt{s}}$). F is a fact included in tuples over the interval $[\mathtt{winTs}, \mathtt{winTe})$. $\lambda_{\mathtt{r}}$ and $\lambda_{\mathtt{s}}$ are the lineage expressions of the input tuples of the left input relation $\mathbf{r}$ and the right input relation $\mathbf{s}$, respectively, that are valid over $[\mathtt{winTs}, \mathtt{winTe})$ and include F. The flexibility of the *lineage-aware temporal window* is based on the fact that the lineages of valid tuples of each input relation are separately recorded. Given a TP set operation, $\lambda_{\mathtt{r}}$ and $\lambda_{\mathtt{s}}$ can be used to determine if fact F and interval $[\mathtt{winTs}, \mathtt{winTe})$ yield an output tuple. If this is the case, $\lambda_{\mathtt{r}}$ and $\lambda_{\mathtt{s}}$ are combined accordingly to form the lineage expression of this output tuple.

All lineage-aware temporal windows are produced by LAWA, a sweeping algorithm we describe in Algorithm 1. Traditionally, sweeping algorithms use a vertical sweepline, and they determine the output tuples based on the input tuples that intersect with this sweepline [17], [14]. This works well for TP set intersection. However, for TP set difference and set union, there are cases when the interval of an output tuple is not determined only by the tuples that intersect with the sweepline. In order to handle such cases, we use a *sweeping window*. The left and right boundaries of the window correspond to the start and end points of a maximal interval that is associated with a potential output interval.

LAWA processes the tuples of two duplicate-free TP relations $\mathbf{r}$ and $\mathbf{s}$ with schema (F, λ, T, p) that are sorted by their facts and starting points of their intervals. It produces lineage-aware temporal windows whose left (`winTs`) and right (`winTe`) boundaries are computed during a sweep of the start (`Ts`) and end (`Te`) points of the tuples. The left boundary $\mathtt{winTs}_i$ of a window i is greater or equal to $\mathtt{winTe}_{i-1}$ of the previous window. Its right boundary $\mathtt{winTe}_i$ is the smallest among the end points of the tuples expected to overlap with this window, i.e., tuples with $\mathtt{Ts} \leq \mathtt{winTs}$ and $\mathtt{Te} > \mathtt{winTs}$, and the start points of the tuples of the two relations to be processed next.

Algorithm 1: LAWA(`status`)

```
(prevWinTe, currFact, rValid, sValid, r, s) = status;
if rValid = null ∧ sValid = null then
    if r = null ∧ s ≠ null then return (null, null) ;
    else if r = null ∧ s ≠ null then
        winTs = s.Ts; currFact = s.F;
    else if r ≠ null ∧ s = null then
        winTs = r.Ts; currFact = r.F;
    else
        if r.F = currFact ∧ s.F ≠ currFact then
            winTs = r.Ts
        if r.F ≠ currFact ∧ s.F = currFact then
            winTs = s.Ts
        else if r.Ts < s.Ts then
            winTs = r.Ts; currFact = r.F;
        else winTs = s.Ts; currFact = s.F; ;
else winTs = prevWinTe ;
if r ≠ null ∧ r.F = currFact ∧ r.Ts = winTs then
    rValid = r; r = getNext(r);
if s ≠ null ∧ s.F = currFact ∧ s.Ts = winTs then
    sValid = s; s = getNext(s);
winTe = min(minTs(r, s), minTe(rValid, sValid));
$\lambda_r$ = null; $\lambda_s$ = null; window = null;
if rValid ≠ null then $\lambda_r$ = rValid.$\lambda$;
if sValid ≠ null then $\lambda_s$ = sValid.$\lambda$;
window = (currFact, winTs, winTe, $\lambda_r$ , $\lambda_s$) ;
if rValid ≠ null ∧ rValid.Te=winTe then rValid = null;
if sValid ≠ null ∧ sValid.Te=winTe then sValid = null;
prevWinTe=winTe;
status = (rValid, sValid, r, s, currFact, prevWinTe);
return (window, status);
```

The input of LAWA is a structure (`status`) with the necessary status information: the right boundary of the last candidate window (`prevWinTe`), the fact that is currently being processed (`currFact`), the current tuples of **r** (`rValid`) and **s** (`sValid`) that are valid over the sweeping window [`winTs`, `winTe`), and the next tuples of relations **r** (`r`) and **s** (`s`). All variables are initialized to `null` except for `r` and `s` that are initialized to the first tuples of the corresponding relations. The value of `prevWinTe` is initialized to -1.

Initially, the left boundary `winTs` of the new window is determined. If at least one tuple is valid, the new window is adjacent to the previous one, with `winTs = prevWinTe` (Line 16). Otherwise, `winTs`, and potentially `currFact`, are determined by the new tuples. Three possible scenarios exist: (a) both relations have been scanned (Line 3), (b) one of the two relations has already been scanned (Lines 4–7), (c) there are available tuples from both **r** and **s**, but they include different facts (Lines 9–12) and (d) there are available tuples from both **r** and **s** that include the same fact (Lines 13–15).

Since the input relations are duplicate-free, i.e., no two tuples of the same relation can include the same fact and be valid at the same time point, `rValid` and `sValid` correspond to exactly one input tuple each. If `rValid` and `sValid` are not `null`, they correspond to tuples that were also overlapping with the previous window. Otherwise, they need to be updated to `r` or `s` if the latter include a fact equal to `currFact` and have a start point equal to `winTs` (Lines 17–20). The right boundary `winTe` is updated to the minimum time point among the end points of `rValid` and `sValid` and the current start points of `r` and `s`, i.e., the next tuples to be processed (Line 21). Here, the tuples `r` and `s` must be considered because the start point of an unprocessed tuple marks a change in the tuples that are valid over that interval.

After λ_r and λ_s are extracted from `rValid` and `sValid` (Lines 23–24), all the information for the creation of a lineage-aware temporal window is recorded (Line 25). `rValid` and `sValid` are updated for the next call of LAWA based on whether the tuples they correspond to are still valid outside the window, i.e., when the end points of these tuples are larger than `winTe`. Finally, LAWA also returns its `status`, which is used in the implementation of the actual TP set operations.

Example 3: In Fig. 4, we illustrate three calls of LAWA with the left and right relations being **c** and **a** of Fig.1a, respectively. Before the first call, the input relations have been sorted by their facts and start points. The time points used to determine the right boundary of a window are annotated with a blue cross. In the first call of LAWA, illustrated at the bottom, the left and right boundary of the window are set to `winTs` $=1$ and `winTe` $=2$, respectively. After `winTs` is determined, the only tuple valid is `rValid` $=c_1$. Thus, given that there is no valid tuple in **a** yet, `winTe` is set to the start point of a_1, i.e., the next tuple of **a** to be processed. This time point is smaller than the end point `Te` $=4$ of `rValid` or the start point `Ts` $=6$ of the upcoming tuple of **c** (c_2). In the second call of LAWA, illustrated in the middle, the left boundary of the next window to be examined is equal to the right boundary of the previous window, i.e., `winTs` $=2$, given that the fact ('milk') is still being processed. The tuples valid after time point $t=2$ are `rValid` $=c_1$ and `sValid` $=a_1$. The right boundary of the

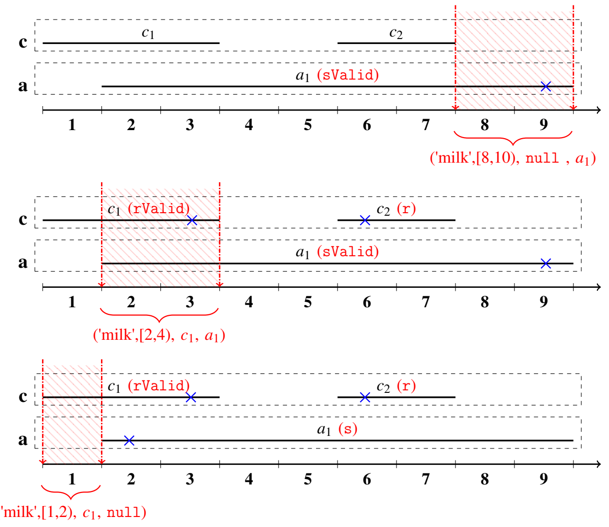


Fig. 4: Three calls of LAWA for the input relations **c** and **a**.

window is the minimum of `rValid.Te` $=4$, `sValid.Te` $=10$ and c_2.`Ts` $=6$, and thus `winTe` $=4$. A similar pattern goes on until the last call of LAWA, illustrated on the top of Fig. 4, where `winTs` $=8$ and `winTe` $=10$. Then, `rValid` and `sValid` are set to `null` and no further windows are produced.

B. Basic TP Set Algorithms

Exploiting the flexibility of a *lineage-aware temporal window*, we reduce the implementation of TP set operations into a four-step process (Fig. 5). The sorting step is a prerequisite for the creation of windows using LAWA. When a window is created, a lineage-based filter (λ_{filter}) is directly applied. The λ_{filter} is different for each TP set operation. In contrast to previous works of either temporal or probabilistic set operations, this step involves no application of additional algebraic operations, no tuple replication and no redundant interval comparisons. After the filtering step, the final lineage expression of an output tuple is created by applying the lineage-concatenating function ($\lambda_{function}$) of the respective TP set operation (Def. 3) on $\lambda_{\mathtt{r}}$ and $\lambda_{\mathtt{s}}$.

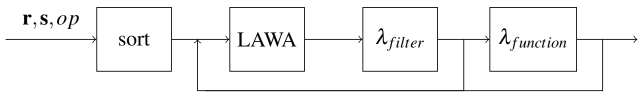


Fig. 5: Process overview.

The algorithms *Intersect*(**r**, **s**), *Union*(**r**, **s**) and *Except*(**r**, **s**) correspond to $\mathbf{r} \cap^{\mathrm{Tp}} \mathbf{s}$, $\mathbf{r} \cup^{\mathrm{Tp}} \mathbf{s}$ and $\mathbf{r} -^{\mathrm{Tp}} \mathbf{s}$, respectively. In all algorithms, input relations are initially sorted based on their facts F and start points `Ts` (Line 1) when the status of LAWA is initialized. As long as the terminating condition (Line 3) is satisfied, LAWA passes through all start and end points in a smaller-to-larger fashion and produces candidate windows (Line 4). The windows produced by LAWA are filtered based on the lineages of the tuples that are valid during the interval it covers (Line 5). The filter used for each operation, as well as the terminating condition and the lineage-concatenating function, directly stem from the definitions of the operation. For example, in the case of set difference $\mathbf{r} -^{\mathrm{Tp}} \mathbf{s}$, windows are produced as long as there are tuples in the outer relation (i.e., while $\mathtt{r} \neq \mathtt{null}$). The interval of a lineage-aware temporal window corresponds to an output tuple only if there is a tuple of the outer relation that is valid over $[\mathtt{winTs}, \mathtt{winTe})$ (i.e., when $\lambda_{\mathtt{r}} \neq \mathtt{null}$).

For *Union*(**r**, **s**) and *Except*(**r**, **s**), when the while-loop terminates, there might still be one more window, corresponding to the subinterval of the last valid tuple of **r** (`rValid`) or the last valid tuple of **s** (`sValid`). Thus, LAWA is called one more time (Line 8).

Algorithm 2: *Intersect*(**r**, **s**)

```
1  sort(r{F, Ts}); sort(s{F, Ts});
2  status = (−1, null, null, null, fetchRow(r), fetchRow(s));
3  while status.r ≠ null ∧ status.s ≠ null do
4      (w, status) = LAWA(status);
5      if w.λr ≠ null ∧ w.λs ≠ null then
6          o = o ∪ {(F, and(w.λr, w.λs), [w.winTs, w.winTe))};
7  return o;
```

Algorithm 3: *Union*(**r**, **s**)

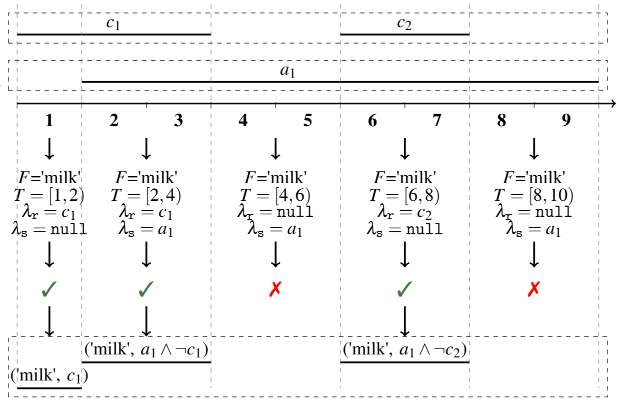
```
1  sort(r{F, Ts}); sort(s{F, Ts});
2  status = (−1, null, null, null, fetchRow(r), fetchRow(s));
3  while status.r ≠ null ∨ status.s ≠ null do
4      (w, status) = LAWA(status);
5      if w.λr ≠ null ∨ w.λs ≠ null then
6          o = o ∪ {(w.F, or(w.λr, w.λs), [w.winTs, w.winTe))};
7  if status.rValid ≠ null ∨ status.sValid ≠ null then
8      (w, status) = LAWA(status);
9      o = o ∪ {(w.F, or(w.λr, w.λs), [w.winTs, w.winTe))};
10 return o;
```

Algorithm 4: *Except*(**r**, **s**)

```
1  sort(r{F, Ts}); sort(s{F, Ts});
2  status = (−1, null, null, null, fetchRow(r), fetchRow(s));
3  while status.r ≠ null do
4      (w, status) = LAWA(status);
5      if w.λr ≠ null then
6          o = o ∪ {(w.F, andNot(w.λr, w.λs), [w.winTs, w.winTe))};
7  if status.rValid ≠ null then
8      (w, status) = LAWA(status);
9      o = o ∪ {(w.F, andNot(w.λr, w.λs), [w.winTs, w.winTe))};
10 return o;
```

Example 4: In Fig. 6, we illustrate the computation of set difference $\sigma_{\mathrm{F='milk'}}(\mathbf{c}) -^{\mathrm{TP}} \sigma_{\mathrm{F='milk'}}(\mathbf{a})$ for relations **c** and **a** in Fig. 1a. The first candidate window $[1,2)$ has $\lambda_{\mathtt{s}} = \mathtt{null}$ and $\lambda_{\mathtt{r}} = c_1$. For set difference the current window yields a result tuple, since, over interval $[1,2)$, the fact ('milk') is included in a tuple of the left input relation **c** with lineage $\lambda_{\mathtt{s}} = c_1$. In contrast, the candidate ('milk', $[4,6)$, `null`, a_1) is rejected since ('milk') is not included in a tuple of the left input relation **c** over $[4,6)$.

Time and Space Complexity: The time complexity of all TP set operations is determined by the complexity of the blocks presented in Fig. 5. Sorting has complexity $O(|\mathbf{r}|\log|\mathbf{r}| + |\mathbf{s}|\log|\mathbf{s}|)$ if it is comparison-based. A variant of counting-based sorting could also be used [12] (which is the case if Ω^T fits into main-memory), and in this case the corresponding complexity is even linear. After sorting, LAWA will sweep over all tuples in the sorted input relations **r** and **s**, accessing two input tuples at a time to determine the next window.



c
c_1
c_2
a
a_1
1
2
3
4
5
6
7
8
9
F='milk' $T=[1,2)$ $\lambda_r = c_1$ $\lambda_s = \mathtt{null}$
F='milk' $T=[2,4)$ $\lambda_r = c_1$ $\lambda_s = a_1$
F='milk' $T=[4,6)$ $\lambda_r = \mathtt{null}$ $\lambda_s = a_1$
F='milk' $T=[6,8)$ $\lambda_r = c_2$ $\lambda_s = \mathtt{null}$
F='milk' $T=[8,10)$ $\lambda_r = \mathtt{null}$ $\lambda_s = a_1$
✓
✓
✗
✓
✗
('milk', $a_1 \wedge \neg c_1$)
('milk', $a_1 \wedge \neg c_2$)
('milk', c_1)

Fig. 6: $\sigma_{\mathrm{F='milk'}}(\mathbf{c}) -^{\mathrm{TP}} \sigma_{\mathrm{F='milk'}}(\mathbf{a})$

Proposition 1: Let **r**, **s** *be two duplicate-free temporal-probabilistic relations. The upper bound of the number of windows produced by the window advancer is $n_r + n_s - f_d$ where n_r, n_s are the number of start and end points in* **r** *and* **s**, *and f_d is number of distinct facts in these relations.*

By Proposition 1, the number of candidate windows considered by the algorithm is linear in the number of time intervals, and thus to the size of the input relations. Thus, LAWA has a time complexity of $O(|\mathbf{r}| + |\mathbf{s}|)$, given that $|\mathbf{r}|$ and $|\mathbf{s}|$ are the numbers of tuples in the input relations **r** and **s**, respectively. Moreover, the filtering and lineage-concatenation step for each candidate output tuple is performed in $O(1)$. Thus, the overall time complexity for computing TP set operations is $O(|\mathbf{r}|\log|\mathbf{r}| + |\mathbf{s}|\log|\mathbf{s}|)$, but may even be reduced to $O(|\mathbf{r}| + |\mathbf{s}|)$ if counting-based sorting is applicable. The use of *lineage-aware temporal windows* not only avoids the use for time-consuming additional operations for the filtering and lineage-concatenation steps, but also allows them to be performed directly at the time a window is created. That is, no intermediate buffers need to be maintained (apart from very few pointers), and thus the space complexity of all TP set operators is constant.

VII. Experimental Evaluation

In this section, we evaluate LAWA in comparison to both temporal and temporal-probabilistic approaches that can be used for the computation of TP set operations. We perform experiments with real datasets as well as with synthetic datasets in which we vary (i) the number of facts in the input relations and (ii) the percentage of tuples whose intervals overlap. In all experiments, our approach empirically scales according to the bounds we provide in Section VI-B. LAWA is the only scalable approach that can be used for the computation of all three TP set operations, outperforming all state-of-the-art approaches for input relations of more than 10M tuples. In contrast to existing techniques, LAWA is robust, i.e., its performance behaves in a predictable manner with respect to the aforementioned characteristics of the datasets.

A. Experimental Setup

All of the following experiments were deployed on a 2xIntel(R) Xeon(R) CPU E5-24400 @2.40GHz machine with 64GB main memory, running CentOS 6.7. LAWA has been implemented in C++ [2], and all experiments were performed in main-memory. No indexes were used. In cases where PostgreSQL implementations were used, the maximum memory for sorting as well as for shared buffers was set to 1GB.

TABLE II: Approach Overview

Approach	$\mathbf{r} \cup^{Tp} \mathbf{s}$	$\mathbf{r} -^{Tp} \mathbf{s}$	$\mathbf{r} \cap^{Tp} \mathbf{s}$
LAWA	✓	✓	✓
NORM	✓	✓	✓
TPDB	✓	✗	✓
OIP	✗	✗	✓
TI	✗	✗	✓

The TP set operations that different approaches can compute are presented in Table II. Set difference is the least-supported operation, followed by set union and set intersection. Set intersection is the most-supported operation among the available systems, since it can be reduced to an interval join with an equality condition on the non-temporal attributes. Specifically, we compare our implementation of TP set operations using LAWA against:

Temporal-Probabilistic Database (TPDB) [1]: The implementation of TPDB is an application connected with a DBMS and consists of three stages. The first stage parses Datalog rules with temporal predicates and translates them to SQL queries. The second stage executes the SQL queries in the DBMS. Base relations are stored in the DBMS, while lineage is kept as an internal data structure in main-memory. The third stage focuses on lineage processing by processing the base tuples with their Boolean connectives. We use the authors' original implementation, connected to PostgreSQL 9.4.3.

Normalize (NORM) [2]: The *Normalize* operator is implemented in the kernel of PostgreSQL by modifying its parser, executor and optimizer. We migrated the authors' implementation to PostgreSQL 9.4.3 for a fair comparison. To support TP set operations, we introduced reduction rules that are proper combinations of the temporal and probabilistic reduction rules (cf. [2], [26]).

Timeline Index (TI) [12]: This approach was used, in its original implementation, for the computation of TP set intersection, by applying a temporal join with an additional condition on the non-temporal attributes as well as the lineage-concatenating function ***and*** (see Table. I).

Overlap Interval Partition Join (OIP) [13]: This approach is designed for overlap joins but does not support an additional filtering condition. For our experimental evaluation, we extended the authors' implementation, so that an equality condition on the non-temporal attributes of the tuples can be applied. In order to use OIP to compute set intersection, we first split each input relation into groups based on the facts included in each tuple. We then applied the OIP partitioning and join over each of these groups and merged the results.

[2]http://www.ifi.uzh.ch/en/dbtg/research/tpset.html

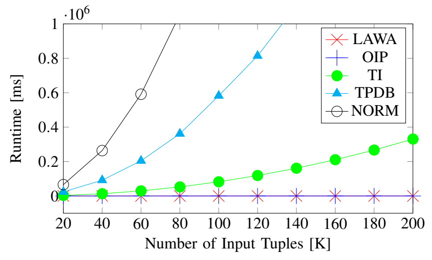
B. Synthetic Dataset

The parameters that we consider to populate a relation of our dataset are: (a) the length of the tuples' intervals, (b) the maximum time distance between two tuples that are consecutive and include the same fact, and (c) the number of different facts included in tuples of the relation. Assume all tuples of relations $\mathbf{r}$ and $\mathbf{s}$ have the same fact f. We define the *overlapping factor* of f as the number of maximal subintervals during which a tuple from $\mathbf{r}$ and $\mathbf{s}$ overlap, divided by the total number of maximal subintervals. Its value thus ranges in $[0, 1]$. The higher the value of this metric, the more pairs of input tuples form output tuples, and therefore the more we stress-test the performance of the various approaches for TP set operations. According to Definition 3, overlapping time points are relevant for all set operations, whereas time points for which a fact is only included in the left input relation are only relevant for TP set difference.

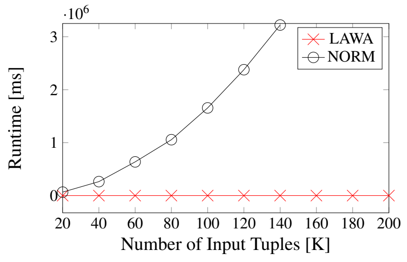
1. Runtime. In the first setting, we fix the input tuples of all datasets to a single fact. We fix the overlapping factor to 0.6, and we randomly select the length of the intervals and the distance between two consecutive intervals in $[0, 3]$. We then systematically increase the number of input tuples. In Fig. 7 and Fig. 8, we illustrate the performance of all the approaches for the computation of TP set operations for smaller datasets with up to 100K tuples and for larger datasets with up to 50M tuples, respectively.

Smaller Datasets [20K–200K]: In Fig. 7, the datasets range from 20K to 200K tuples. Fig. 7a focuses on TP set intersection. The runtimes of LAWA and OIP hardly increase for the small datasets. Both outperform NORM, TI and TPDB by a large margin. OIP is specifically designed for the computation of an overlap join, to which TP set intersection is reduced. NORM exhibits poor performance even if the number of input tuples is only 50K. In this approach, regardless of the operation, the two input relations need to first be normalized, such that, in their adjusted versions, the intervals would be either equal or disjoint. The most expensive part of the normalization of a relation $\mathbf{r}$ using relation $\mathbf{s}$ is an outer join that uses inequality conditions on the start and end points to guarantee an overlap of the intervals. Although an additional inner join is applied in the case of TP set intersection, the performance of NORM suffers because of the outer join. Since all tuples include the same fact, but not all of them overlap, such a join has quadratic complexity [31].

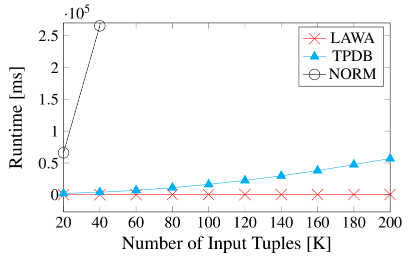
In TPDB, queries are expressed using Datalog. Each rule may contain a conjunction of literals over the arithmetic predicates $=^T$, $\neq^T$ and $\leq^T$. In order to express TP set intersection, we use 6 reduction rules, one for each overlap relationship defined by Allen [32]. TPDB then translates each



(a) Set Intersection



(b) Set Difference



(c) Set Union

Fig. 7: Synthetic Dataset [20K–200K]

rule to an inner join that is submitted to PostgreSQL. Although there is an equality condition on the non-temporal attributes, it is not used in the cases examined in Fig. 7 where all the tuples include the same fact. Thus, the joins are only based on the inequality conditions and perform a larger number of comparisons. TPDB is slower than the other approaches, but it is still faster than NORM, because the latter has to adjust each relation.

Although TI is faster than NORM and TPDB, it is one of the slowest approaches for set intersection. The index allows for the avoidance of redundant comparisons related to the interval overlap condition, and its creation cost is a small percentage of its runtime. Given the indexes of the input relations, TI performs a merge-join on them and produces (r_{id}, s_{id}) pairs. In order to form the output tuples, the input tuples corresponding to each pair need to be retrieved. Given the value of the overlapping factor and the existence of only one fact, a higher number of joined pairs is produced and thus a higher number of lookups is required. OIP splits the tuples of each input relation into partitions, based on the start/end points of their interval and its duration. Consequently, it offers a mechanism that performs interval comparisons between tuples only if their partitions overlap. If the partitions overlap, OIP performs a nested loop between the tuples of the two relations. As the overlapping factor is 0.6, which indicates that most of the pairs produced in the nested loop will indeed be output pairs, OIP has a very small percentage of false hits. Although OIP is tailored for an overlap join, for datasets of up to 200K tuples LAWA's performance is competitive, being on average 30 ms slower.

In the case of TP set difference, as illustrated in Fig. 7b, LAWA clearly outperforms NORM, for the same reasons as for TP set intersection. Fig. 7c compares LAWA with NORM and TPDB during the computation of TP set union. LAWA has the lowest runtime, whereas NORM has the highest one, being 5 orders of magnitude slower than LAWA. The window that sweeps over all the input tuples in LAWA makes no false hits in this case, since all of the subintervals that the window defines correspond to output intervals. NORM no longer requires a join but a union after the relations have been normalized. However, as in all the previous operations, NORM's performance is hindered by the computation of the timestamp adjustment. TPDB can also compute TP set union by using a deduction rule that corresponds to a conventional union instead of joins, and thus its performance is significantly better in comparison to TP set intersection.

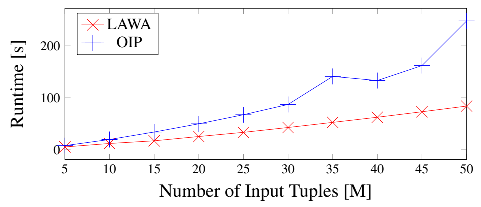


Fig. 8: Synthetic Dataset [5M–50M]

Larger Datasets [5M–50M]: LAWA is the only scalable approach that can be used for the computation of all three TP set operations. In Fig. 8, we depict the performance of LAWA for the computation of TP set intersection for larger datasets. The overlapping factor of the datasets remains fixed to 0.6, and the dataset sizes vary from 5M to 50M tuples. While OIP is also considered, the other approaches that were included in Fig. 7a are not taken into consideration, since their runtimes were already two to five orders of magnitude higher when applied on the smaller datasets. After 30M tuples, LAWA is at least 2 times faster than OIP and continues to scale better. OIP produced a small number of partitions that contain many tuples each. Such partitions are likely to overlap and the nested loop that matches their tuples is computationally expensive. As far as TP set difference and TP set union are concerned, LAWA has similar runtime as in the case of TP set intersection and it is the only scalable approach suitable for their computation within at most 100 seconds.

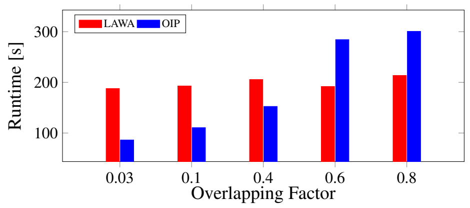
2. Robustness. In this experiment, we show that LAWA is a scalable operator whose runtime only depends on the size of the dataset and not on its other characteristics (i.e., neither on the value of the overlapping factor nor on the number of distinct facts captured by the input tuples).

TABLE III: Dataset Characteristics

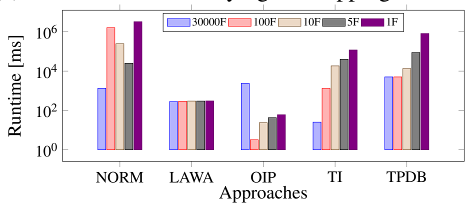
Overlapping Factor	0.03	0.1	0.4	0.6	0.8
Max. Interval Length (R)	100	100	50	3	10
Max. Interval Length (S)	3	10	10	3	10
Max. Time Distance	3				

In Fig. 9a, the performance of LAWA for set intersection is compared with the one of OIP, which has been the most competitive approach for datasets where all the tuples include the same fact. This time, the size of the dataset is fixed to

30M, and the overlapping factor is assigned to four different values in $[0,1]$. Table III depicts the *overlapping factor* of the datasets as well as their *maximum interval lengths* (in terms of the number of time points). The runtime of OIP increases as the overlapping metric increases. The reason is that the higher the overlapping factor, the more tuples occur in a partition and the nested loop performed in each partition is very time consuming. On the other hand, only minor variations are observed in the runtime of LAWA for the different values of the overlapping factor, thus demonstrating that the performance of LAWA is not negatively affected by interval-related characteristics of the dataset.



(a) Performance for varying overlapping factors.



(b) Performance for varying numbers of distinct facts.

Fig. 9: Robustness Tests

In Fig. 9b, we show how the number of distinct facts in the input relations affects the performance of LAWA and all other approaches during a TP set intersection. The size of the dataset is set to 60K, so that the runtimes of the approaches are comparable, and the overlapping metric is set to 0.6. The number of facts is set to values much less than the size of the dataset, but also to a value that is equal to half the size of the dataset. The runtime of LAWA remains stable as the number of the facts included in the input tuples decreases, whereas the performance of the other approaches deteriorates. OIP is an exception since, if the number of facts becomes comparable to the number of tuples, it suffers from the overhead of partitioning the tuples of each fact, performing the corresponding join and merging the results. Concerning the other approaches, TI has a better performance than LAWA but only in the case of 30K facts. This behaviour is expected, since there is a low number of joined pairs, thus reducing the number of required lookups. NORM's performance improves as well when the number of facts increases, but this approach does not scale to datasets with more than 30K tuples. TPDB, on the other hand, appears to have diminishing improvements.

C. Real-World Datasets

In this subsection, we compare the runtimes of TP set operations using two real-world temporal datasets. The main properties of these datasets are summarized in Table IV. The Meteo Swiss dataset[3] includes temperature predictions that have been extracted from the website of the Swiss Federal Office of Meterology and Climatology. The measurements were taken at 80 different meteorological stations in Switzerland from 2005 to 2015. Measurements are 10 minutes apart and – in order to produce intervals – we merged time points whose measurements differ by less than 0.1. The Webkit dataset[4] [13], [14], [15] records the history of 484K files of the SVN repository of the Webkit project over a period of 11 years at a granularity of milliseconds. The valid times indicate the periods when a file remained unchanged. For both datasets we produced a second relation by shifting the intervals of the original dataset, without modifying the lengths of the intervals. The start/end points of the new relation were randomly chosen, following the distribution of the original ones.

TABLE IV: Real-World Dataset Properties

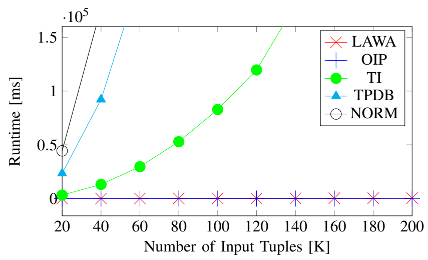
	Meteo	**Webkit**
Cardinality	10.2M	1.5M
Time Range	347M	7M
Min. Duration	600	0.02
Max. Duration	19.3M	6M
Avg. Duration	152M	1.7M
Num. of Facts	80	484K
Distinct Points	545K	144K
Max Num. of Tuples (per time point)	140	369K
Avg Num. of Tuples (per time point)	37	21

In Fig. 10 and Fig. 11, we perform TP set intersection, difference and union over two equally sized relations created from random subsets of the initial dataset and its shifted counterpart, respectively. The runtime of each approach is based on the number of tuples in the input relations. In all cases, LAWA has the best performance. All approaches perform similarly to the synthetic dataset, with the exception of TI and NORM for the Webkit dataset. In this dataset, the maximum number of tuples starting or ending at a certain time point is very high, thus negatively affecting the performance of TI that has to make pairs among all of the tuples at a time point before it rejects the ones that do not match the nontemporal condition. Also, the number of facts is much higher than in the Meteo Swiss Dataset, making NORM significantly faster.

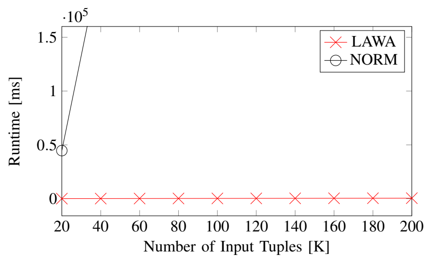
VIII. Conclusions

We proposed a novel data model that—for the first time in the literature—unifies the two areas of temporal and probabilistic databases under a sequenced semantics. We defined and implemented TP set operations, which can be supported very efficiently for a wide range of queries but received only very little attention so far. We introduced the lineage-aware temporal window as a mechanism to accelerate the computation of TP set operations. Our LAWA algorithm produces lineage-aware temporal windows that can be filtered directly by the time of their creation based on input lineage expressions. Using a generic window-sweeping technique, LAWA manages to produce all output intervals, not only for TP set intersection

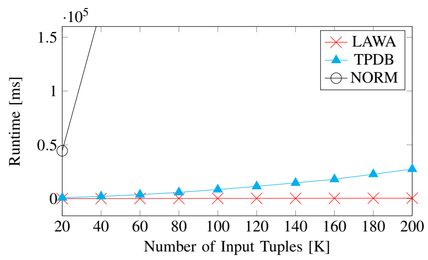
[3]Federal Office of Meteorology and Climatology: http://www.meteoswiss.ch (2016)
[4]The WebKit Open Source Project: http://www.webkit.org (2012)



(a) Set Intersection

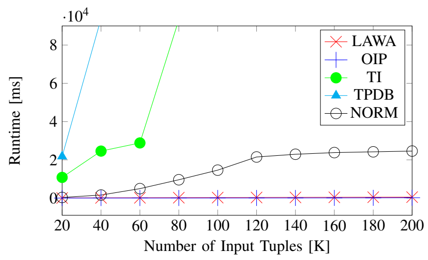


(b) Set Difference

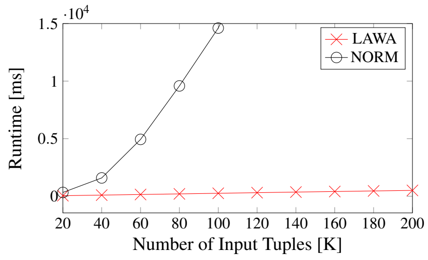


(c) Set Union

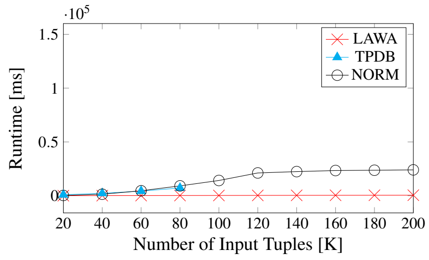
Fig. 10: Meteo Swiss Dataset



(a) Set Intersection



(b) Set Difference



(c) Set Union

Fig. 11: Webkit Dataset

but also for TP set difference and TP set union, in a scalable and predictable manner. A thorough experimental evaluation reveals that our implementation is robust and outperforms comparable approaches from both temporal and probabilistic databases. As future work, we intend to investigate both tuple correlations and support for full relational algebra.

References

[1] M. Dylla, I. Miliaraki, and M. Theobald, "A temporal-probabilistic database model for information extraction," *PVLDB*, vol. 6, no. 14, pp. 1810–1821, 2013.

[2] A. Dignös, M. H. Böhlen, J. Gamper, and C. S. Jensen, "Extending the Kernel of a Relational DBMS with Comprehensive Support for Sequenced Temporal Queries," *TODS*, vol. 41, no. 4, pp. 26:1–26:46, 2016.

[3] A. Dignös, M. H. Böhlen, and J. Gamper, "Temporal alignment," in *SIGMOD*, 2012, pp. 433–444.

[4] R. Fink, D. Olteanu, and S. Rath, "Providing support for full relational algebra in probabilistic databases," in *ICDE*, 2011, pp. 315–326.

[5] M. H. Böhlen and C. Jensen, "Sequenced Semantics," in *Encyclopedia of Database Systems*. Springer Berlin, Heidelberg, Germany, 2009, pp. 2619–2621.

[6] D. Suciu, "Probabilistic Databases," in *Encyclopedia of Database Systems*. Springer Berlin, Heidelberg, Germany, 2009, pp. 2150–2155.

[7] D. Suciu, D. Olteanu, R. Christopher, and C. Koch, *Probabilistic Databases*, 1st ed. Morgan & Claypool Publishers, 2011.

[8] M. Al-Kateb, A. Ghazal, A. Crolotte, R. Bhashyam, J. Chimanchode, and S. P. Pakala, "Temporal query processing in teradata," in *EDBT/ICDT*, 2013, pp. 573–578.

[9] N. A. Lorentzos and Y. G. Mitsopoulos, "SQL extension for interval data," *TKDE*, vol. 9, no. 3, pp. 480–499, 1997.

[10] J. R. R. Viqueira and N. A. Lorentzos, "SQL Extension for Spatio-temporal Data," *VLDB-J*, vol. 16, no. 2, pp. 179–200, 2007.

[11] D. Toman, "Point-based temporal extensions of SQL and their efficient implementation," in *Temporal databases: research and practice*. Springer, 1998, pp. 211–237.

[12] M. Kaufmann, A. A. Manjili, P. Vagenas, P. M. Fischer, D. Kossmann, F. Färber, and N. May, "Timeline index: a unified data structure for processing queries on temporal data in SAP HANA," in *SIGMOD*, 2013, pp. 1173–1184.

[13] A. Dignös, M. H. Böhlen, and J. Gamper, "Overlap interval partition join," in *SIGMOD*, 2014, pp. 1459–1470.

[14] D. Piatov, S. Helmer, and A. Dignös, "An interval join optimized for modern hardware," in *ICDE*, 2016, pp. 1098–1109.

[15] F. Cafagna and M. H. Böhlen, "Disjoint interval partitioning," *VLDB J.*, vol. 26, no. 3, pp. 447–466, 2017.

[16] M. Kaufmann, P. Vagenas, P. M. Fischer, D. Kossmann, and F. Färber, "Comprehensive and interactive temporal query processing with SAP HANA," *PVLDB*, vol. 6, no. 12, pp. 1210–1213, 2013.

[17] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter, "Scalable sweeping-based spatial join," in *VLDB*, 1998, pp. 570–581.

[18] A. D. Sarma, M. Theobald, and J. Widom, "Exploiting lineage for confidence computation in uncertain and probabilistic databases," in *ICDE*, 2008, pp. 1023–1032.

[19] O. Benjelloun, A. D. Sarma, A. Y. Halevy, M. Theobald, and J. Widom, "Databases with uncertainty and lineage," *VLDB J.*, vol. 17, pp. 243–264, 2008.

[20] R. Fink and D. Olteanu, "Dichotomies for queries with negation in probabilistic databases," *ACM Trans. Database Syst.*, vol. 41, pp. 4:1–4:47, 2016.

[21] D. Olteanu, J. Huang, and C. Koch, "Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases," in *ICDE*, 2009, pp. 640–651.

[22] N. N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *VLDB J.*, vol. 16, no. 4, pp. 523–544, 2007.

[23] ——, "The dichotomy of probabilistic inference for unions of conjunctive queries," *J. ACM*, vol. 59, no. 6, pp. 30:1–30:87, 2012.

[24] D. Olteanu and J. Huang, "Using OBDDs for efficient query evaluation on probabilistic databases," in *SUM*, 2008, pp. 326–340.

[25] R. Fink, J. Huang, and D. Olteanu, "Anytime approximation in probabilistic databases," *VLDB J.*, vol. 22, no. 6, pp. 823–848, 2013.

[26] R. Fink and D. Olteanu, "On the optimal approximation of queries using tractable propositional languages," in *ICDT*, 2011, pp. 174–185.

[27] W. Gatterbauer and D. Suciu, "Oblivious bounds on the probability of boolean functions," *TODS*, vol. 39, no. 1, p. 5, 2014.

[28] ——, "Approximate lifted inference with probabilistic databases," *PVLDB*, vol. 8, no. 5, pp. 629–640, 2015.

[29] D. Olteanu, J. Huang, and C. Koch, "Approximate confidence computation in probabilistic databases," in *ICDE*, 2010, pp. 145–156.

[30] S. Khanna, S. Roy, and V. Tannen, "Queries with difference on probabilistic databases," *PVLDB*, vol. 4, no. 11, pp. 1051–1062, 2011.

[31] Z. Khayyat, W. Lucia, M. Singh, M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and P. Kalnis, "Lightning fast and space efficient inequality joins," *PVLDB*, vol. 8, no. 13, pp. 2074–2085, 2015.

[32] J. F. Allen, "Maintaining Knowledge About Temporal Intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.