An Overview on Querying and Learning in Temporal Probabilistic Databases

Maximilian Dylla*

Google
Paris, France
maxdylla@google.com

Abstract: Probabilistic databases store, query and manage large amounts of uncertain information in an efficient way. This paper summarizes my thesis which advances the state-of-the-art in probabilistic databases in three different ways: First, we present a closed and complete data model for temporal probabilistic databases. Queries are posed via temporal deduction rules which induce lineage formulas capturing both time and uncertainty. Second, we devise a methodology for computing the top-k most probable query answers. It is based on first-order lineage formulas representing sets of answer candidates. Moreover, we derive probability bounds on these formulas which enable pruning low-probability answers. Third, we introduce the problem of learning tuple probabilities, which allows updating and cleaning of probabilistic databases, and study its complexity and characterize its solutions.

1 Introduction

In the last decade, we saw another big rise in the amount of digital information. Database systems play a key role in managing this data because of their abilities in storing, querying, and updating large data collections. One of the basic assumptions in database systems is that the data is certain: a data record is either a perfect implementation of some real-world truth or, if it does not hold, absent in the database. In reality, however, a large amount of data is not deterministic, but rather uncertain. For example, coarse-grained input can result in uncertainty. For instance, a sensor with limited precision is inherently uncertain with respect to the precise physical value. Moreover, ambiguity can cause uncertainty. For instance, most sentences in natural language allow more than one interpretation, sometimes even leaving the reader in doubt. One way to store, query and manage large amounts of uncertain data are probabilistic databases (PDBs) [SOCK11]. My thesis [Dyl14] advances the field of probabilistic databases in several ways, where this paper serves as a brief overview.

Contributions and Outline. In Section 2, we devise a *temporal-probabilistic data model* which supports both time and probabilities as first-class citizens. In Section 3, we present

*This research was conducted while attending Max Planck Institute for Informatics. The views and conclusions contained herein are those of the author and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of Google.

an approach for computing the *top-k query answers*, which feature the highest probabilities among all answers. Our main technical tool are *first-order lineage* formulas, which can represent both entire sets of query answers and partially evaluated grounding states. We formally derive *probability bounds* for these formulas, enframing the probabilities of all answers represented by the formulas. In Section 4, we establish a methodology to *learn tuple probabilities* from labeled lineage formulas. We characterize the problem theoretically by investigating its *complexity* as well as the *nature of its solutions*.

2 Temporal Probabilistic Databases

In recent years, temporal databases [Jen00] (where tuples are valid only at some points in time) and probabilistic databases [SOCK11] (whose tuples exist with a probability only) have emerged as two intensively studied areas of database research. So far, the two fields have however been investigated largely only in isolation. In this section, we sketch a temporal-probabilistic database (TPDB) model [DMT13, Dyl14] supporting both temporal as well as probabilistic data.

Example 1. *Our running example is based on information extraction which aims to automatically harvest factual knowledge from web sources. The temporal probabilistic database of Figure 1 captures a number of extracted facts around the actor "Robert DeNiro" and their origin, that is, the web domain they were extracted from. The probability of each tuple represents to which degree we believe this extraction is correct. Tuple I_1 expresses that* DeNiro *got married to* Abbott *on April 28th, 1976, which is encoded into the time interval* $[1976\text{-}04\text{-}28, 1976\text{-}04\text{-}29)$. *The time and probability annotations together express that this tuple is* true *for the given time interval with probability* 0.2*, and it is* false *for this interval with probability* 0.8. *Note that we allow relations without temporal annotations, e.g.,* FromDomain*, whose tuples are valid at all time points.*

WeddingExtraction

	Person$_1$	Person$_2$	Did	Valid Time	p
I_1	*DeNiro*	*Abbott*	1	$[1976\text{-}04\text{-}28, 1976\text{-}04\text{-}29)$	0.2
I_2	*DeNiro*	*Abbott*	2	$[1976\text{-}04\text{-}28, 1976\text{-}04\text{-}29)$	0.9
I_3	*DeNiro*	*Hightower*	2	$[1997\text{-}01\text{-}01, 1998\text{-}01\text{-}01)$	0.6

DivorceExtraction

	Person$_1$	Person$_2$	Did	Valid Time	p
I_4	*DeNiro*	*Abbott*	1	$[1988\text{-}09\text{-}01, 1988\text{-}12\text{-}01)$	0.4

FromDomain

	Did	Domain	p
I_5	1	*Wikipedia.org*	1.0
I_6	2	*People.com*	1.0

Figure 1: An Example Temporal Probabilistic Database

Time. As a fist step, we introduce our notion of time. We consider the *time universe* $\mathcal{U}^T$ as a linearly ordered, finite sequence of *time points*, e.g., days, minutes or even milliseconds. Then, a *time-interval* consists of a contiguous and finite set of time points over $\mathcal{U}^T$, which we denote by a half-open interval $[t_b, t_e)$ where $t_b, t_e \in \mathcal{U}^T$ and t_b before t_e.

Example 2. *Regarding Figure 1 we can choose $\mathcal{U}^T$ to be the sequence of all days of the 20th century.*

Temporal Relation. Before we formally define TPDBs, we establish a useful property of temporal relations, e.g., *WeddingExtraction*. Assume we drop *Did* as an argument of *WeddingExtraction*, then a probabilistic database engine might conclude that *DeNiro* got married twice on April 28th, 1976 with different probabilities. To resolve this issue, we enforce the time-intervals of tuples with identical non-temporal arguments $\bar{a}$ (e.g., the first three columns of *WeddingExtraction*) to be disjoint, termed *duplicate-free* [DBG12].

Definition 1. *A temporal relation instance $\mathcal{R}$ is called* duplicate-free*, if for all pairs of tuples $R(\bar{a}, t_b, t_e), R(\bar{a}', t'_b, t'_e) \in \mathcal{R}$ it holds that:*

$$\bar{a} = \bar{a}' \;\Rightarrow\; [t_b, t_e) \cap [t'_b, t'_e) = \emptyset$$

The above definition is fulfilled for all relation instances of Figure 1, since I_1, I_2, and I_3 have differing non-temporal arguments.

Temporal Probabilistic Database. Now, we define a *temporal probabilistic database* as the triple $(\mathcal{T}, p, \mathcal{U}^T)$ [DMT13, Dyl14]. Here, $\mathcal{T}$ is the set of all tuples across all relations (which we assume to be duplicate-free). Furthermore, p is a function $p : \mathcal{T} \to (0, 1]$ which assigns a non-zero probability value to each tuple. The probability values of different tuples are assumed to be independent. Finally, $\mathcal{U}^T$ is the time universe as defined before.

Example 3. *For the database of Figure 1 we have $\mathcal{T} = \{I_1, \ldots, I_9\}$ and $p(I_1) = 0.2$.*

Temporal Deduction Rules. To derive new knowledge from a TPDB, we employ temporal deduction rules. These can be viewed as generally applicable "if-then-rules". Formally, deduction rules have the shape of a logical implication with a conjunction of both positive and negative literals in the body and exactly one positive literal in its head. This class of rules coincides with safe Datalog without recursion [AHV95] or nested select-project-join queries in SQL.

Example 4. *Given the tuples of Figure 1 we first aim to reconcile the facts, i.e., when* DeNiro *got married and divorced, by writing these two temporal deduction rules:*

$$\underbrace{Wedding(P_1, P_2, T_b, T_e)}_{head} \leftarrow \underbrace{\begin{pmatrix} WeddingExtraction(P_1, P_2, Did, T_b, T_e) \\ \wedge\ FromDomain(Did, D) \end{pmatrix}}_{body} \quad (1)$$

$$Divorce(P_1, P_2, T_b, T_e) \leftarrow \begin{pmatrix} DivorceExtraction(P_1, P_2, Did, T_b, T_e) \\ \wedge\ FromDomain(Did, D) \end{pmatrix} \quad (2)$$

Both rules propagate the persons P_1 and P_2 and the time interval as specified by $[T_b, T_e)$ to the head relation, e.g., Wedding *or* Divorce*. More interestingly, we are now able to deduce the time interval of their marriage starting at the beginning of the wedding and ending at the end of the divorce:*

$$Marriage(P_1, P_2, T_{b,1}, T_{e,2}) \leftarrow \begin{pmatrix} Wedding(P_1, P_2, T_{b,1}, T_{e,1}) \wedge \\ Divorce(P_1, P_2, T_{b,2}, T_{e,2}) \ \wedge \end{pmatrix} T_{e,1} <^T T_{b,2} \quad (3)$$

Thereby, we consider only weddings that took place before divorces as stated by the condition $T_{e,1} <^T T_{b,2}$. If a couple is still married, we let the time interval of their marriage

start at the wedding until the last possible time point (denoted by the constant t_{max}).

$$Marriage(P_1, P_2, T_b, t_{max}) \leftarrow (Wedding(P_1, P_2, T_b, T_e) \ \wedge \ \neg Divorce'(P_1, P_2)) \quad (4)$$

Here, the existence of any divorce independent of time is modeled by the projection:

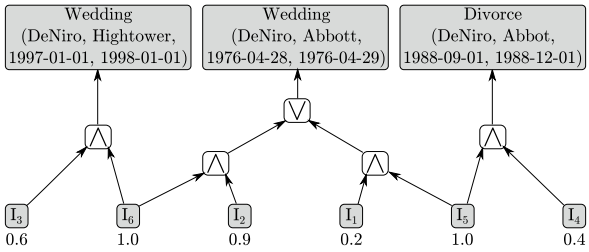
$$Divorce'(P_1, P_2) \leftarrow Divorce(P_1, P_2, T_b, T_e)$$

Queries. Based on deduction rules we define a query to be a conjunction of head literals.

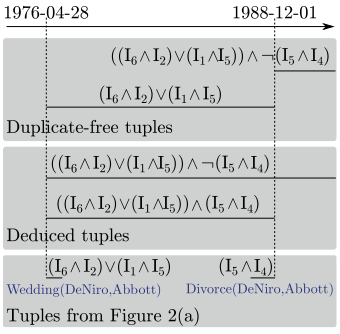
Example 5. *Extending Example 4, we formulate the query $Wedding(P_1, P_2, T_1, T_2) \wedge Divorce(P_1, P_2, T_2, T_3)$, which asks for persons P_1, P_2 who got divorced right after their wedding.*

Lineage. Next, we instantiate the deduction rules over the TPDB, i.e., we employ the rule to deduce new tuples, which is called grounding. Thereby we trace the deduction history of the new tuples. In database terminology this is called *data lineage* [BDSH+08], which we represent by a propositional formula. More detailed, lineage relates each new tuple with the tuples $\mathcal{T}$ via the three Boolean connectives $\wedge$, $\vee$ and $\neg$. These reflect the semantics of the relational operations that were applied to deduce that tuple.

Example 6. *If we apply the rules of Equations* (10) *and* (11) *to the tuples of Figure 1, we obtain the lineage formulas depicted in Figure 2(a). There, Equation* (10) *can be instantiated twice to deduce* Wedding(DeNiro,Abbott,1976-04-28,1976-04-29)*, which results in the two conjunctions in the middle. Then, both conjunctions are connected by a disjunction representing these two options.*



(a) Lineage Formulas of Example 6



(b) Lineage and Deduplication

Figure 2: Lineage

In general, if we execute temporal deduction rules, the newly deduced tuples may not necessarily define a duplicate-free relation. We illustrate this issue by the following example.

Example 7. *In Example 6, we deduced tuples about DeNiro's wedding to and divorce from Abbott which are displayed on the bottom of Figure 2(b). Applying the deduction rules of Equations* (3) *and* (4) *to these tuples yields the tuples in the middle of the figure, which have equivalent non-temporal arguments, i.e.,* DeNiro *and* Abbott*, but their time-intervals are overlapping, which contradicts Definition 1 of duplicate-free relation instances.*

Hence, in order to convert a temporal relation instance with duplicates (as shown in the middle of Figure 2(b)) into a duplicate-free temporal relation (as shown on the top of Figure 2(b)), we provide the following definition [DMT13, Dyl14]. Let $\lambda(I)$ stand for the lineage formula attached to a deduced tuple I.

Definition 2. *Let a temporal relation R, non-temporal constants $\bar{a}$, a time point $t \in \mathcal{U}^T$, and a set of tuples $\mathcal{T}$ be given. Then, L is defined as the set of lineages of tuples $R(\bar{a}, t_b, t_e)$ that are valid at time point t:*

$$L(R, \bar{a}, t, \mathcal{T}) := \{\lambda(I) \mid I = R(\bar{a}, t_b, t_e) \in \mathcal{T}, t_b \leq t < t_e\}$$

We create duplicate free tuples $I' = R(\bar{a}, t_b, t_e)$ *such that for any pair of time points $t_0, t_1 \in [t_b, t_e)$ it holds that:*

$$L(R, \bar{a}, t_0, \mathcal{T}) = L(R, \bar{a}, t_1, \mathcal{T}) \tag{5}$$

Furthermore, we define the new tuples' lineages to be:

$$\lambda(I') := \bigvee_{\phi_i \in L(R, \bar{a}, t_b, \mathcal{T})} \phi_i \tag{6}$$

In short, for each time-point t we create the disjunction of all tuples being valid at t (see Equation (6)). More detailed, for a given relation instance and the non-temporal arguments of a tuple, L is the set of all tuples' lineages that share the same non-temporal arguments and which are valid at time point t. Hence, consecutive time-points for which L contains the same lineage formulas form the new intervals (see Equation (5)).

Example 8. *Applying Definition 2 to the tuples shown in the middle of Figure 2(b) yields the tuples shown at the top of the figure. For instance, if we inspect L at the time points* 1988-11-30 *and* 1988-12-01*, we notice that $\{((I_6 \wedge I_2) \vee (I_1 \wedge I_5)) \wedge (I_5 \wedge I_4), ((I_6 \wedge I_2) \vee (I_1 \wedge I_5)) \wedge \neg(I_5 \wedge I_4)\} \neq \{((I_6 \wedge I_2) \vee (I_1 \wedge I_5)) \wedge \neg(I_5 \wedge I_4)\}$, so two different tuples have to be kept in the relation.*

Probability Computations. Since in a probabilistic database, each database tuple exists only with a given probability, we can quantify the probability that each deduced tuple exists. Formally, we compute the *probability* $P(\phi)$ of any lineage formula ϕ over tuples in $\mathcal{T}$ as the sum of the probabilities of all subsets $\mathcal{W} \subseteq \mathcal{T}$. For this, we consider only subsets, which satisfy ϕ, i.e., $\mathcal{W} \models \phi$, by setting all tuples in $\mathcal{W}$ to be *true* and all others to be *false* [SOCK11]:

$$P(\phi) := \sum_{\mathcal{W} \models \phi} P(\mathcal{W}) \qquad \text{where} \qquad P(\mathcal{W}) = \prod_{t \in \mathcal{W}} p(t) \cdot \prod_{t \in \mathcal{T} \setminus \mathcal{W}} 1 - p(t) \tag{7}$$

In practice, we can compute the probability $P(\phi)$ – in many cases – directly on the structure of the lineage formula ϕ. Let $Tup(\phi) \subseteq \mathcal{T}$ denote the set of tuples occurring in ϕ. Then, the following efficient computations can be employed [SOCK11]:

Definition			Condition
$P(I)$	$:=$	$p(I)$	$I \in \mathcal{T}$
$P(\bigwedge_i \phi_i)$	$:=$	$\prod_i P(\phi_i)$	$i \neq j \Rightarrow Tup(\phi_i) \cap Tup(\phi_j) = \emptyset$
$P(\bigvee_i \phi_i)$	$:=$	$1 - \prod_i (1 - P(\phi_i))$	$i \neq j \Rightarrow Tup(\phi_i) \cap Tup(\phi_j) = \emptyset$
$P(\neg\phi)$	$:=$	$1 - P(\phi)$	

(8)

Example 9. *Considering the lineage formula of DeNiro and Abbot's wedding in Figure 2(a), we obtain* $P((I_6 \wedge I_2) \vee (I_1 \wedge I_5)) = 1 - (1 - P(I_6 \wedge I_2)) \cdot (1 - P(I_1 \wedge I_5))$ *by applying the third line of Equation* (8)*. Then, the second and first line yield* $1 - (1 - p(I_6) \cdot p(I_2)) \cdot (1 - p(I_1) \cdot p(I_5)) = 0.92$*.*

If none of the cases of Equation (8) applies, we employ the following equation, called *Shannon expansion* [SOCK11], which is applicable to any propositional lineage formula:

$$P(\phi) := p(I) \cdot P(\phi_{[I/true]}) + (1 - p(I)) \cdot P(\phi_{[I/false]}) \tag{9}$$

Here, the notation $\phi_{[I/true]}$ for a tuple $I \in Tup(\phi)$ denotes that we replace all occurrences of I in ϕ by *true*. Repeated applications of Shannon Expansions, however, result in an exponential runtime.

Further Properties. As layed out in [DMT13, Dyl14] the sketched data model is closed and complete, i.e., it can express all instances of temporal probabilistic data. Furthermore, the data model can be extended by consistency constraints.

3 Top-k Querying

Instead of computing all answers to a query in a bottom-up manner as in Section 2, the focus of this section is computing only the top-k answers with the highest probabilities among all answers. For that, we start at the query and expand deduction rules until we reach the database tuples, which is called top-down grounding [AHV95].

First-Order Lineage. During top-down grounding, not all variables variables will be bound to constants. Hence, we now extend propositional lineage of Section 2 to first-order lineage [DTM13, Dyl14], which can contain variables and quantifiers. In contrast to propositional lineage, first-order lineage does not represent single query answers, but rather entire sets of answers. Each answer in such a set will be characterized by constants binding the query variables. To facilitate the construction of first-order lineage, we write all quantifiers of variables only occurring in the body of a deduction rule explicitly.

Example 10. *We equivalently rewrite the deduction rules of Equations* (10) *and* (11) *as follows:*

$$Wedding(P_1, P_2, T_b, T_e) \leftarrow \exists Did, D \begin{pmatrix} WeddingExtraction(P_1, P_2, Did, T_b, T_e) \\ \wedge\ FromDomain(Did, D) \end{pmatrix} \tag{10}$$

$$Divorce(P_1, P_2, T_b, T_e) \leftarrow \exists Did, D \begin{pmatrix} DivorceExtraction(P_1, P_2, Did, T_b, T_e) \\ \wedge\ FromDomain(Did, D) \end{pmatrix} \tag{11}$$

Note that in the above deduction rules all variables not occurring in the head literal, i.e., Did *and* D*, are bound by existential quantifiers.*

Grounding. Instead of starting at the database and instantiating the deduction rules to deduce new tuples, as before, we now begin at the query and in each step resolve a literal.

If the literal matches the head literal of a deduction rule, we replace it by the body of the rule. Otherwise, if the literal matches a database relation, we replace it by database tuples. Finally, if the no tuple and head literal matches, we substitute it by false. Whenever we bind a variable to constants $a_1, \ldots, a_n$, we employ the equivalence

$$\exists X \Phi \equiv \Phi_{[X/a_1]} \vee \cdots \vee \Phi_{[X/a_n]} \tag{12}$$

where $[X/a_i]$ substitutes the variable X by the constant a_i.

Example 11. *We consider the query* $Wedding(\textit{DeNiro}, P, T_1, T_2)$*, which asks for weddings of* DeNiro*. First, we replace the literal by the deduction rule body of Equation* (10)*:*

$$\exists Did, D\, WeddingExtraction(\textit{DeNiro}, P, Did, T_1, T_2) \wedge FromDomain(Did, D)$$

The above first-order lineage formula represents a set of query answers, one for each tuple of constants binding P*,* T_1*, and* T_2*. If we next exchange* $FromDomain(Did, D)$ *for the tuples of* I_5 *and* I_6 *of Figure 1, we obtain*

$$\begin{array}{c}(\, WeddingExtraction(\textit{DeNiro}, P, 1, T_1, T_2) \wedge I_5) \\ \vee (\, WeddingExtraction(\textit{DeNiro}, P, 2, T_1, T_2) \wedge I_6)\end{array} \tag{13}$$

where the disjunction results from Equation (12)*. The lineage formula still represents the same set of answers.*

Probability Bounds. Next, given a first-order lineage formula Φ, we construct two propositional formulas ϕ_{low} and ϕ_{up} whose probabilities then serve as lower and upper bound on Φ, respectively.

Definition 3. *Let* Φ *be a first-order lineage formula in negation normal form.*

1. *We construct the propositional lineage formula* ϕ_{up} *by substituting every literal* $R(\bar{X})$ *in* Φ *with* true *if* $R(\bar{X})$ *occurs positive in* Φ*, and by* false *otherwise.*
2. *We construct the propositional lineage formula* ϕ_{low} *by substituting every literal* $R(\bar{X})$ *in* Φ *with* false *if* $R(\bar{X})$ *occurs positive in* Φ*, and with* true *otherwise.*

Example 12. *We obtain* ϕ_{up} *from Equation* (13) *by setting the* WeddingExtraction *literals to* true*, which yields* $(true \wedge I_5) \vee (true \wedge I_6)$*. Hence,* $P((true \wedge I_5) \vee (true \wedge I_6)) = 1.0$ *is an upper bound.*

As shown in [DTM13, Dyl14], if $\phi_1, \ldots, \phi_n$ represent all query answers we would obtain by fully grounding Φ, then it holds that:

$$\forall i \in \{1, \ldots, n\} : \; P(\phi_{low}) \leq P(\phi_i) \leq P(\phi_{up})$$

Furthermore, with each step of the top-down grounding these bounds bounds converge monotonically to the probabilities of the answers. The resulting lower and upper bounds for all answer candidates can be plugged into any top-k algorithm [IBS08] that will then iteratively refine these lower and upper bounds until a termination condition is reached.

Pruning Answer Candidates. Given a set of answer candidates A_{top}, we can prune the answer candidate Φ, if $\forall \Psi \in A_{top} : P(\phi_{up}) \leq P(\psi_{low})$. Due to the monotonicity the lower bounds of all answer candidates in A_{top} can only increase, whereas the upper bound $P(\phi_{up})$ can only decrease. Hence, Φ never reaches a higher probability than any answer in A_{top}.

4 Learning Tuple Probabilities

Most works in the context of PDBs assume the database tuples along with their probabilities to be given as input. Also the preceding sections of this paper followed this route. Nevertheless, when creating, updating or cleaning a TPBD, the tuple probabilities have to be altered or even be newly created—in other words: they have to be *learned*, which is the topic of this section.

Example 13. *Assume we are given the TPDB of Figure 1 along with the deduction rules of Equations* (10) *and* (11)*. Figure 3 shows the resulting lineage formulas, where, however,*

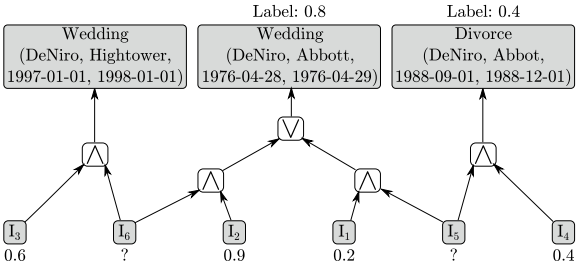


Figure 3: Lineage Formulas

the probabilities of the tuples of the FromDomain *relation are missing and marked with question marks. Instead, we are given labels for the deduced tuples from which we intend to learn the missing probabilities. For instance, a human assessor may have labeled the wedding of* DeNiro *and* Abbott *on April 28th 1976 to be 80% correct.*

Learning Problem. Formally, for a temporal probabilistic database $(\mathcal{T}, p, \mathcal{U}^T)$, we consider $\mathcal{T}_l \subseteq \mathcal{T}$ to be the set of database tuples for which we learn their probability values. That is, initially $p(I)$ is unknown for all $I \in \mathcal{T}_l$. Conversely, $p(I)$ is known and fixed for all $I \in \mathcal{T} \backslash \mathcal{T}_l$. To be able to complete $p(I)$, we are given labels in the form of pairs (ϕ_i, l_i), each containing a propositional lineage formula ϕ_i (i.e., a query answer) and its desired probability l_i. Now, we define the resulting learning problem.

Definition 4. *We are given a temporal probabilistic database $(\mathcal{T}, p, \mathcal{U}^T)$, a set of tuples $\mathcal{T}_l \subseteq \mathcal{T}$ with unknown probability values $p(I_l), I_l \in \mathcal{T}_l$, and a multi-set of given labels $\mathcal{L} = \langle(\phi_1, l_1), \ldots, (\phi_n, l_n)\rangle$, where each ϕ_i is a propositional lineage formula over $\mathcal{T}$ and each $l_i \in [0, 1] \subset \mathbb{R}$ is a probability for ϕ_i. Then, the* learning problem *is defined as:*

$$\begin{aligned} &\textit{Determine: } p(I_l) \in [0,1] \subset \mathbb{R} \textit{ for all } I_l \in \mathcal{T}_l \\ &\textit{such that: } P(\phi_i) = l_i \textit{ for all } (\phi_i, l_i) \in \mathcal{L} \end{aligned}$$

Intuitively, we aim to set the probability values of the database tuples $I_l \in \mathcal{T}_l$ such that the labeled lineage formulas ϕ_i again yield the probability l_i. We want to remark that all probability values of tuples in $\mathcal{T} \backslash \mathcal{T}_l$ remain unaltered. Also, we note that the Boolean labels *true* and *false* can be represented as $l_i = 0.0$ and $l_i = 1.0$, respectively.

Example 14. *Formalizing the problem setting of Figure 3, we obtain* $\mathcal{T} = \{I_1, \ldots, I_6\}$, $\mathcal{T}_l = \{I_5, I_6\}$, $\mathcal{L} = \langle((I_6 \wedge I_2) \vee (I_1 \wedge I_5), 0.8), (I_5 \wedge I_4, 0.4)\rangle$.

Complexity. We next discuss the complexity of solving the learning problem. Unfortunately, it exhibits hard instances. First, computing $P(\phi_i)$ might require many Shannon expansions (see Equation (9)) causing exponential runtimes. But even for cases when all $P(\phi_i)$ can be computed in polynomial time (i.e., when Equation (8) is applicable), there are combinatorially hard cases of the above learning problem.

Lemma 1. *For a given instance of the learning problem of Definition 4, where all* $P(\phi_i)$ *with* $(\phi_i, l_i) \in \mathcal{L}$ *can be computed in polynomial time, deciding whether there exists a solution to the learning problem is* $\mathcal{NP}$*-hard.*

The above statement can be proven by encoding 3SAT into the learning problem [Dyl14].

Inconsistent Instances. After discussing the complexity of the learning problem, we characterize its solutions. First, there might also be *inconsistent* instances of the learning problem. That is, it may be impossible to define $p : \mathcal{T}_l \rightarrow (0, 1]$ such that all labels are satisfied.

Example 15. *If we consider* $\mathcal{T}_l := \{I_1, I_2\}$ *with the labels* $\mathcal{L} := \langle(I_1, 0.2), (I_2, 0.3), (I_1 \wedge I_2, 0.9)\rangle$*, then it is impossible to fulfill all three labels at the same time.*

Number of Solutions. From a practical point of view, there remain a number of questions regarding Definition 4. First, how many labels do we need in comparison to the number of tuples for which we are learning the probability values (i.e., $|\mathcal{L}|$ vs. $|\mathcal{T}_l|$)? And second, is there a difference in labeling lineage formulas that involve many tuples or very few tuples (i.e., $|\mathit{Tup}(\phi_i)|$)? Before we proceed to the theorem, recall that we can express probability computations $P(\phi)$ of a lineage formula ϕ as a polynomial following Equation (7). These polynomials have degree at most $|\mathit{Tup}(\phi)|$ as shown in [Dyl14].

Theorem 1. *If the labeling is consistent, the problem instances of Definition 4 can be classified as follows:*

1. *If* $|\mathcal{L}| < |\mathcal{T}_l|$*, the problem has infinitely many solutions.*
2. *If* $|\mathcal{L}| = |\mathcal{T}_l|$ *and the polynomials* $P(\phi_i) - l_i$ *have common zeros, then the problem has infinitely many solutions.*
3. *If* $|\mathcal{L}| = |\mathcal{T}_l|$ *and the polynomials* $P(\phi_i) - l_i$ *have no common zeros, then the problem has at most* $\prod_i |\mathit{Tup}(\phi_i) \cap \mathcal{T}_l|$ *solutions.*
4. *If* $|\mathcal{L}| > |\mathcal{T}_l|$*, then the polynomials* $P(\phi_i) - l_i$ *have common zeros, thus reducing this to one of the previous cases.*

The proof [Dyl14] of the above statement is accomplished via Bezout's theorem [DE10]. In general, a learning problem instance has many solutions, where Definition 4 does not specify a precedence, but all of them are equivalent. The number of solutions shrinks by adding labels to $\mathcal{L}$, or by labeling lineage formulas ϕ_i that involve fewer tuples in $\mathcal{T}_l$ (thus resulting in a smaller intersection $|\mathit{Tup}(\phi_i) \cap \mathcal{T}_l|$).

Example 16. *Employing Theorem 1 on the learning problem of Example 14, the third case applies, since* $|\mathcal{L}| = |\mathcal{T}_l|$ *and there are no common zeros. Hence, the theorem yields an upper bound of* $\prod_i |Tup(\phi_i) \cap \mathcal{T}_l| = |Tup((I_6 \wedge I_2) \vee (I_1 \wedge I_5)) \cap \mathcal{T}_l| \cdot |Tup(I_5 \wedge I_4) \cap \mathcal{T}_l| = |\{I_6, I_5\}| \cdot |\{I_5\}| = 2$. *The only solution, however, is* $p(I_5) = 0.4$ *and* $p(I_6) = 0.83$.

Solving the Learning Problem. Algorithmic solutions to the learning problem are beyond the scope of this paper, but we refer the interested reader to [Dyl14] for several gradient based approaches and a detailed experimental evaluation.

5 Conclusions

We presented a temporal probabilistic data model, devised how to compute the top-k query answers with the highest probabilities, and showed how to learn tuple probabilities in order to create or update probabilistic databases. For a more in-depth presentation of these aspects we refer the interested reader to [Dyl14].

References

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases*. Addison-Wesley, 1st edition, 1995.

[BDSH+08] Omar Benjelloun, Anish Das Sarma, Alon Halevy, Martin Theobald, and Jennifer Widom. Databases with uncertainty and lineage. *VLDB Journal*, 17(2):243–264, March 2008.

[DBG12] Anton Dignös, Michael H. Böhlen, and Johann Gamper. Temporal alignment. SIGMOD, pages 433–444. ACM, 2012.

[DE10] Alicia Dickenstein and Ioannis Z. Emiris. *Solving Polynomial Equations: Foundations, Algorithms, and Applications*. Springer, 1st edition, 2010.

[DMT13] Maximilian Dylla, Iris Miliaraki, and Martin Theobald. A Temporal-Probabilistic Database Model for Information Extraction. *VLDB*, 6(14):1810–1821, 2013.

[DTM13] Maximilian Dylla, Martin Theobald, and Iris Miliaraki. Top-k query processing in probabilistic databases with non-materialized views. ICDE, pages 122–133, 2013.

[Dyl14] Maximilian Dylla. *Efficient Querying and Learning in Probabilistic and Temporal Databases*. PhD thesis, Saarland University, Saarbrücken, Germany, 2014.

[IBS08] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Computing Surveys*, 40(4):11:1–11:58, October 2008.

[Jen00] Christian S. Jensen. *Temporal Database Management*. PhD thesis, Aalborg University, Aalborg, Denmark, April 2000.

[SOCK11] Dan Suciu, Dan Olteanu, R Christopher, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool, 1st edition, 2011.