Efficient Querying and Learning in Probabilistic and Temporal Databases

Maximilian Dylla

Thesis for obtaining the title of Doctor of Engineering
of the Faculties of Natural Sciences and Technology
of Saarland University

Saarbrücken, Germany, 2014

Abstract

Probabilistic databases store, query, and manage large amounts of uncertain information. This thesis advances the state-of-the-art in probabilistic databases in three different ways:

1. We present a closed and complete data model for temporal probabilistic databases and analyze its complexity. Queries are posed via temporal deduction rules which induce lineage formulas capturing both time and uncertainty.

2. 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. Theoretically derived probability bounds on these formulas enable pruning low-probability answers.

3. We introduce the problem of learning tuple probabilities which allows updating and cleaning of probabilistic databases. We study its complexity, characterize its solutions, cast it into an optimization problem, and devise an approximation algorithm based on stochastic gradient descent.

All of the above contributions support consistency constraints and are evaluated experimentally.

Kurzfassung

Probabilistische Datenbanken können große Mengen an ungewissen Informationen speichern, anfragen und verwalten. Diese Doktorarbeit treibt den Stand der Technik in diesem Gebiet auf drei Arten vorran:

1. Ein abgeschlossenes und vollständiges Datenmodell für temporale, probabilistische Datenbanken wird präsentiert. Anfragen werden mittels Deduktionsregeln gestellt, welche logische Formeln induzieren, die sowohl Zeit als auch Ungewissheit erfassen.

2. Ein Methode zur Berechnung der k Anworten höchster Wahrscheinlichkeit wird entwickelt. Sie basiert auf logischen Formeln erster Stufe, die Mengen an Antwortkandidaten repräsentieren. Beschränkungen der Wahrscheinlichkeit dieser Formeln ermöglichen das Kürzen von Antworten mit niedriger Wahrscheinlichkeit.

3. Das Problem des Lernens von Tupelwahrscheinlichkeiten für das Aktualisieren und Bereiningen von probabilistischen Datenbanken wird eingeführt, auf Komplexität und Lösungen untersucht, als Optimierungsproblem dargestellt und von einem stochastischem Gradientenverfahren approximiert.

All diese Beiträge unterstützen Konsistenzbedingungen und wurden experimentell analysiert.

Acknowledgement

I would like to thank my supervisor Martin Theobald for investing a lot of time in discussing technical subjects with me no matter whether he was based in Saarbrücken or Antwerp. Moreover, I am grateful to Gerhard Weikum for his consistent support throughout my PhD. Every discussion with Rainer Gemulla spurred numerous technical ideas and insights, which were very helpful for me. Besides, I want to express my gratitude to all my co-authors, Iris Miliaraki, Yafang Wang, Marc Spaniol, Mauro Sozio, and Timm Meiser, for doing great work. Furthermore, I thank all remaining members of the databases and information systems group for forming a very knowledgeable group and making it an awesome place to work.

With respect to this thesis my thanks go to Avishek Anand, Radu Curticapean, Marvin Künnemann, and Bilyana Taneva for proof reading preliminary versions of it. Also, I owe Dan Suciu a dept of gratitude for providing technical comments on the complicated parts of my thesis.

I owe all my friends in Saarbrücken a lot, since they provided me the right counterbalance to my work on the thesis. Here, honorable mentions go to Daniel Fischer for being ready to meet literally at any time. Finally, I want to thank my family for continuous support and motivation during all these years.

Contents

Chapter 1

Introduction

1.1 Motivation

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. Nowadays, it is hard to imagine a single major company or other organization not running a database system for coping with its administrative, financial or corporate data. 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.

Sources of Probabilistic Data A *sensor*, for instance, comes with a limited precision, and hence its measurements are inherently uncertain with respect to the precise physical value. Also, *coarse-grained information* carries uncertainty about the underlying more fine-grained information. For example, if we are interested in the birth place of Albert Einstein and we only know that he was born in Germany, then we are unsure about the city of birth. Moreover, *ambiguity* can cause uncertainty. For instance, most sentences in natural language allow more than one interpretation, sometimes leaving the reader in doubt. Likewise, if we *lack information*, maybe due to an anonymization process reporting an age interval rather than the actual age of a person, we are confronted with several alternatives which in turn generate uncertainty. Similarly, when we observe *inconsistent information*, such as two differing locations of a conference venue, then we can no longer be certain about the true information. Finally, there will always be *uncertainty about the future*, as can be noticed for instance in every day's weather forecast.

Probabilistic Databases One way to store, query and manage large amounts of uncertain data are probabilistic databases [139]. They lie in the intersection of database systems [2, 65], (for handling large amounts of data), first-order logic [133, 146], (for writing structured queries on the data), and probability theory [45, 130] (for quantifying the uncertainty). So far, research in probabilistic databases focused mainly on two aspects: the *data model* and the *query evaluation problem*.

For the former, a variety of approaches for compactly representing the data uncertainty were presented. These start at simple models, such as *tuple-independent* probabilistic databases [29, 139] which have a probability value attached to each tuple, where these values are assumed to be independent. More expressive are *pc-tables* [60] where each tuple is annotated by a logical formula defining its dependencies. Finally, there are sophisticated models which capture statistical *correlations* among database tuples [76, 119, 128].

Regarding the query evaluation problem, there exist queries for which computing the probabilities of their respective answers is $\#\mathcal{P}$-hard [31, 58]. This caused a series of works taking mainly two directions. Either these approaches [31, 29, 32] classify queries syntactically to decide whether they feature efficient probability computations. Or they target the representation of the derivation of query answers [74, 105, 129], which commonly are logical formulas called *lineage*, to facilitate the answers' probability computations.

In recent years a number of probabilistic database systems were released as open-source prototypes including *MayBMS* [10], *MystiQ* [18], *PrDB* [128], *SPROUT* [106], and *Trio* [14], which allow for storing and querying uncertain data, by some of the aforementioned methodologies. These systems found wide recognition in the database community.

Challenges In order to make probabilistic databases as broadly applicable as the conventional database systems, a number of challenges remain.

1. Besides potentially being uncertain, data can be annotated by other dimensions such as time, location or data source. These techniques are already supported by traditional database systems. To enable this kind of data in probabilistic databases, we need to extend the probabilistic data models to support additional data dimensions.

2. Allowing a wide range of expressive queries which can be executed efficiently, was one of the ingredients which made traditional database systems successful. Even though the query evaluation problem was studied in probabilistic databases, for many queries efficient ways of computing answers along with probabilities are not established yet.

3. Most importantly, the field of creating and updating probabilistic databases is in an early stage, where only very few results exist [83, 137]. Nevertheless, we have to support learning or updating of probabilities

in the data, since this is needed in many applications: when a user provides feedback by labeling a query answer, we have to integrate this additional information into the probabilistic database, e.g. by altering probabilities. Similarly, imagine an engineer who adds consistency constraints to the database, this rules out certain possibilities, hence affecting probabilities. Finally, we wish to create probabilistic databases from incomplete databases by learning probabilities for each completion.

In this thesis, we tackle each of the above challenges.

1.2 Contributions

Based on *tuple-independent probabilistic databases* with *lineage*, this thesis presents theoretical, algorithmic, and experimental results on the following subjects:

1. In Chapter 3, we devise a *temporal-probabilistic data model* which supports both time and probabilities as first-class citizens. We prove it to be *closed and complete* and characterize its *complexity* properties. Moreover, we present an efficient *algorithm* which allows extending any probabilistic database system with lineage to handle temporal data, and analyze the resulting properties experimentally.

2. In Chapter 4, we present 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, capturing the probabilities of all represented answers. The resulting *algorithm* is evaluated experimentally.

3. In Chapter 5, we establish the first 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*, cast it into an *optimization problem*, and give an *algorithm* for solving it. Additionally, we discuss how to apply the learning of tuple probabilities to *creating*, *updating*, and *cleaning* of probabilistic databases. We provide an extensive *experimental evaluation.*

Cutting across the above chapters, we consider *consistency constraints* in each of our contributions. Some results of this thesis were published previously in the proceedings of international conferences [39, 40, 41, 98, 143, 152, 153].

Outline The remainder of this thesis is structured as follows. In Chapter 2 we introduce the background required for the understanding of this work by covering relational databases and probabilistic databases. The third, fourth, and fifth chapters are the main parts of the thesis, where we discuss the temporal-probabilistic data model, querying for the top-k answers, and the framework of learning tuple probabilities. Related work is discussed in the individual chapters. Finally, Chapter 6 presents system implementation issues before Chapter 7 concludes this thesis.

Visual Overview To ease the understanding of the interactions among the different topics and chapters, we provide a visual overview depicted in Figure 1.1. Returning to this figure after reading each chapter helps to grasp the big picture of this thesis.

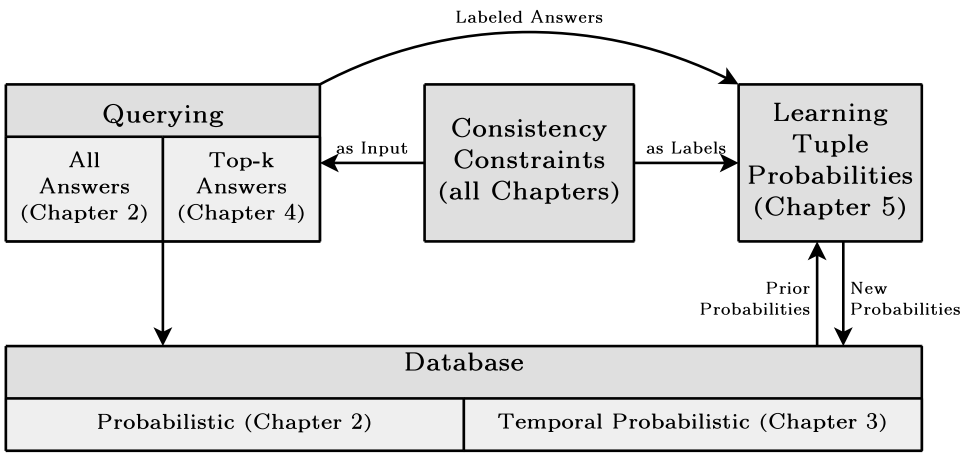

Labeled Answers
Querying
All Answers (Chapter 2)
Top-k Answers (Chapter 4)
as Input
Consistency Constraints (all Chapters)
as Labels
Learning Tuple Probabilities (Chapter 5)
Prior Probabilities
New Probabilities
Database
Probabilistic (Chapter 2)
Temporal Probabilistic (Chapter 3)

Figure 1.1: Visual Overview of the Thesis

Chapter 2

Background and Preliminaries

This chapter introduces the data model that is common to all following chapters, namely *tuple-independent probabilistic databases* with *lineage* and *constraints*. We start by introducing traditional (non-probabilistic) databases in Section 2.1. Building on their concepts and notations, we devise probabilistic databases in Section 2.2. Next, we present basic algorithms necessary for query answering in Section 2.3 before we discuss approaches which are related to our data model in Section 2.4. Finally, Section 2.5 presents an application of probabilistic databases, namely information extraction.

2.1 Relational Databases

The primary purpose of this section is to establish all terms and definitions from databases necessary for this work. For foundations and broader background, however, we refer the reader to a text book, e.g. [2, 65].

2.1.1 Relations and Tuples

We start with the basics of databases, namely relations and tuples. A relation can be viewed as a table. A row or entry in the table is called tuple. From a formal perspective, we follow standard database terminology [2, 65]. We consider a *relation* R to be a logical predicate of arity $r \geq 1$. For a fixed universe of constants $\mathcal{U}$, a *relation instance* $\mathcal{R}$ is a finite subset $\mathcal{R} \subseteq \mathcal{U}^r$. We call the elements of $\mathcal{R}$ *tuples* and write $R(\bar{a})$ to indicate a tuple in $\mathcal{R}$, where $\bar{a}$ is a vector of constants. As shorthand notation we employ I as an identifier for a tuple. Furthermore, we use $R(\bar{X})$ to refer to a first-order literal over relation R where $\bar{X}$ is a vector of variables and constants. Then, $Var(\bar{X})$ denotes the set of variables contained in $\bar{X}$.

Definition 2.1. *Given relations $R_1, \ldots, R_n$, usually called schemata, a* database *comprises the relation instances $\mathcal{R}_i$ whose tuples we collect in the set $\mathcal{T} := \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_n$.*

Example 2.1. *We consider two relation instances from the movie domain describing directors of movies and awards movies have won.*

Directed

	Director	Movie
I_1	*Coppola*	*ApocalypseNow*
I_2	*Coppola*	*Godfather*
I_3	*Tarantino*	*PulpFiction*

WonAward

	Movie	Award
I_4	*ApocalypseNow*	*BestScript*
I_5	*Godfather*	*BestDirector*
I_6	*Godfather*	*BestPicture*
I_7	*PulpFiction*	*BestPicture*

For example, the tuple Directed(Coppola, ApocalypseNow)*, which we also abbreviate by I_1, indicates that* Coppola *directed the movie* ApocalypseNow*. So, the above database contains two relation instances with $\mathcal{T} = \{I_1, \ldots, I_7\}$.*

2.1.2 Deduction Rules

To derive new knowledge from a database, we employ deduction rules. These can be viewed as generally applicable "if-then-rules". That is, given a condition, its conclusion follows. Formally, we follow Datalog terminology and notation [2, 21]. Deduction rules have the shape of a logical implication with a conjunction of both positive and negative literals in the body (the antecedent) and exactly one positive head literal (the consequent). Relations occurring in the head literal of a deduction rule are called *intensional relations* [2]. In contrast, relations holding database tuples, i.e. from $\mathcal{T}$, are named *extensional relations*. These two sets of relations must not overlap and are used differently in deduction rules.

Definition 2.2. *A* deduction rule *is a logical rule of the form*

$$R_0(\bar{X}_0) \leftarrow \bigwedge_{i=1,\ldots,n} R_i(\bar{X}_i) \wedge \bigwedge_{j=1,\ldots,m} \neg R_j(\bar{X}_j) \quad \wedge \Phi(\bar{X})$$

where

1. *R_0 denotes the intensional relation of the head literal, whereas R_i and R_j refer to intensional or extensional relations;*
2. *$n \geq 1$, $m \geq 0$, thus requiring at least one positive relational literal;*
3. *$\bar{X}_0$, $\bar{X}_i$, $\bar{X}_j$, and $\bar{X}$ denote tuples of variables and constants, where $Var(\bar{X}_0) \cup Var(\bar{X}_j) \cup Var(\bar{X}) \subseteq \bigcup_i Var(\bar{X}_i)$;*
4. *$\Phi(\bar{X})$ is a conjunction of arithmetic comparisons such as $=$ and $\neq$.*

Whenever we employ a set of deduction rules*, we call it* $\mathcal{D}$*.*

By definition we require each deduction rule to have at least one positive literal in its body. Moreover, the third condition ensures *safe* [2] deduction rules, by insisting that all variables in the head $Var(\bar{X}_0)$, in negated literals $Var(\bar{X}_j)$, and in arithmetic predicates $Var(\bar{X})$ are bound by variables $Var(\bar{X}_i)$ in positive relational predicates. Additionally, all variables are implicitly universally quantified. As denoted by the forth condition, we allow a conjunction of arithmetic comparisons such as $=$ and $\neq$.

Example 2.2. *Imagine we are interested in famous movie directors. To derive these from the tuples in Example 2.1, we can reason as follows: "if a director's movie won an award, the director should be famous." In a logical formula we express this as:*

$$FamousDirector(D) \leftarrow Directed(D, M) \wedge WonAward(M, A) \tag{2.1}$$

The above rule fulfills all requirements of Definition 2.2, since (1) all relations the body are extensional, (2) there are two positive literals, $n = 2$*, and no negative literal,* $m = 0$*, and (3) the variable* D *of the head is bound in the body.*

With the exception of Section 4.6.2, this thesis considers only *non-recursive* deduction rules. Thus, our class of deduction rules corresponds to safe, non-recursive Datalog programs, which also coincides with the core operations expressible in Relational Algebra [2].

2.1.3 Grounding

The process of applying a deduction rule to a database instance, i.e. employing the rule to derive new tuples, is called grounding. For that, we show how to instantiate deduction rules, which we achieve by *substitutions* [2, 146].

Definition 2.3. *A* substitution $\sigma : \mathcal{V} \rightarrow \mathcal{V} \cup \mathcal{U}$ *is a mapping from variables* $\mathcal{V}$ *to variables and constants* $\mathcal{V} \cup \mathcal{U}$*. A substitution* σ *is applied to a first-order formula* Φ *as follows:*

Definition	*Condition*
$\sigma(\bigwedge_i \Phi_i) := \bigwedge_i \sigma(\Phi_i)$	
$\sigma(\bigvee_i \Phi_i) := \bigvee_i \sigma(\Phi_i)$	
$\sigma(\neg\Phi) := \neg\sigma(\Phi_i)$	
$\sigma(R(\bar{X})) := R(\bar{Y})$	$\sigma(\bar{X}) = \bar{Y}$

In general, substitutions can rename variables or replace variables by constants. If all variables are substituted by constants, then the resulting rule or literal is called *ground*.

Example 2.3. *A valid substitution is given by* $\sigma(D) = \text{Coppola}, \sigma(M) = \text{Godfather}$*, where we replace the variables* D *and* M *by the constants* Coppola *and* Godfather*, respectively. If we apply the substitution to the deduction rule of Equation* (2.1)*, we receive*

$$\mathit{FamousDirector}(\text{Coppola}) \leftarrow \begin{pmatrix} \mathit{Directed}(\text{Coppola}, \text{Godfather}) \\ \wedge \mathit{WonAward}(\text{Godfather}, A) \end{pmatrix}$$

where A *remains a variable.*

We now collect all substitutions for a deduction rule which are possible over a given database or a set of tuples. These substitutions are called *groundings* [2, 21].

Definition 2.4. *Given a set of tuples* $\mathcal{T}$ *and a deduction rule* D

$$R_0(\bar{X}_0) \leftarrow \bigwedge_{i=1,\dots,n} R_i(\bar{X}_i) \wedge \bigwedge_{j=1,\dots,m} \neg R_j(\bar{X}_j) \quad \wedge \Phi(\bar{X})$$

the groundings $G(D, \mathcal{T})$ *are all substitutions* σ *where*

1. σ*'s preimage coincides with* $\bigcup_i \mathit{Var}(\bar{X}_i)$*;*
2. σ*'s image consists of constants only;*
3. $\forall i : \sigma(R_i(\bar{X}_i)) \in \mathcal{T}$*;*
4. $\sigma(\Phi(\bar{X})) \equiv \mathit{true}$*.*

The first and second condition requires the substitution to bind all variables in the deduction rule to constants. In addition, all positive ground literals have to match a tuple in $\mathcal{T}$. In the case of deterministic databases, negative literals must not match a tuple. Later, in probabilistic databases they may match a tuple, which is why we omit a requirement on them. The last condition ensures that the arithmetic literals are satisfied.

Example 2.4. *Let the deduction rule of Equation* (2.1) *be* D*. For the tuples of Example 2.1, there are four groundings* $G(D, \{I_1, \dots, I_7\}) = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$*, where:*

$$\begin{array}{ll} \sigma_1(D) = \mathit{Coppola} & \sigma_2(D) = \mathit{Coppola} \\ \sigma_1(M) = \mathit{ApocalypseNow} & \sigma_2(M) = \mathit{Godfather} \\ \sigma_1(A) = \mathit{BestScript} & \sigma_2(A) = \mathit{BestDirector} \\ \\ \sigma_3(D) = \mathit{Coppola} & \sigma_4(D) = \mathit{Tarantino} \\ \sigma_3(M) = \mathit{Godfather} & \sigma_4(M) = \mathit{PulpFiction} \\ \sigma_3(A) = \mathit{BestPicture} & \sigma_4(A) = \mathit{BestPicture} \end{array}$$

All substitutions are valid groundings according to Definition 2.4, because (1) their preimages coincide with all variables of D*, (2) their images are* *constants only, (4) there are no arithmetic literals, and (3) all positive body literals match tuples:*

Literal	*Tuple*	*Literal*	*Tuple*
$\sigma_1(\mathit{Directed}(D, M))$	I_1	$\sigma_1(\mathit{WonAward}(M, A))$	I_4
$\sigma_2(\mathit{Directed}(D, M))$	I_2	$\sigma_2(\mathit{WonAward}(M, A))$	I_5
$\sigma_3(\mathit{Directed}(D, M))$	I_2	$\sigma_3(\mathit{WonAward}(M, A))$	I_6
$\sigma_4(\mathit{Directed}(D, M))$	I_3	$\sigma_4(\mathit{WonAward}(M, A))$	I_7

Finally, we employ the groundings of a deduction rule to derive new tuples by instantiating the head literal of the rule.

Definition 2.5. *Given a deduction rule $D := (R_0(\bar{X}_0) \leftarrow \Psi)$ and a set of tuples $\mathcal{T}$, the new tuples are:*

$$\mathit{NewTuples}(D, \mathcal{T}) := \{\sigma(R_0(\bar{X}_0)) \mid \sigma \in G(D, \mathcal{T})\}$$

We note that the same new tuple might result from more than one substitution, as illustrated by the following example.

Example 2.5. *Let D be the deduction rule of Equation* (2.1)*. Continuing Example 2.4, there are two new tuples:*

$$\mathit{NewTuples}(D, \{I_1, \ldots, I_7\}) = \left\{ \begin{array}{c} \mathit{FamousDirector}(\mathit{Coppola}), \\ \mathit{FamousDirector}(\mathit{Tarantino}) \end{array} \right\}$$

The first new tuple originates from σ_1, σ_2, σ_3 of Example 2.4 whereas the second new tuple results only from σ_4.

2.1.4 Queries and Answers

We now move on to define queries and their answers over a database. Our queries are conjunctions of positive and negative literals which coincide with the bodies of deduction rules.

Definition 2.6. *Given deduction rules $\mathcal{D}$ with their intensional relations, a* query *Q is a conjunction:*

$$Q(\bar{X}_0) := \bigwedge_{i=1,\ldots,n} R_i(\bar{X}_i) \;\wedge \bigwedge_{j=1,\ldots,m} \neg R_j(\bar{X}_j) \;\wedge \Phi(\bar{X})$$

where

1. *all R_i, R_j are intensional;*
2. *$\bar{X}_0$ are called* query variables *and it holds that $\mathit{Var}(\bar{X}_0) = \bigcup_{i=1,\ldots,n} \mathit{Var}(\bar{X}_i)$;*

3. *all variables in negated or arithmetic literals are bound by positive literals:* $Var(\bar{X}) \subseteq \bigcup_{i=1,\dots,n} Var(\bar{X}_i)$ *and* $\forall j \in \{1,\dots,m\}$ $Var(\bar{X}_j) \subseteq \bigcup_{i=1,\dots,n} Var(\bar{X}_i)$*;*

4. $\Phi(\bar{X})$ *is a conjunction of arithmetic comparisons.*

The first condition allows to ask for head literals of any deduction rule. The set of variables in positive literals are precisely the query variables. The final two conditions ensure safeness as in deduction rules. We want to remark that, for a theoretical analysis, it suffices to have only one intensional literal as query.

Example 2.6. *Extending Examples 2.1 and 2.2, we can formulate the query*

$$Q(D) := \mathit{FamousDirector}(D) \wedge (D \neq \mathit{Tarantino})$$

which asks for famous directors excluding Tarantino, as we may have watched all of his movies. The only query variable is D*.*

As queries take the shape of bodies of deduction rules, we employ the groundings of Definition 2.4 on queries as well. Assuming that $\mathcal{T}$ comprises all database tuples and all new tuples resulting from deduction rules, we rely on $G(Q(\bar{X}), \mathcal{T})$ to define answers.

Definition 2.7. *For a set of tuples* $\mathcal{T}$ *and a query* $Q(\bar{X})$*, the set of* answers *is given by:*

$$\mathit{Answers}(Q(\bar{X}), \mathcal{T}) := \{\sigma(Q(\bar{X})) \mid \sigma \in G(Q(\bar{X}), \mathcal{T})\}$$

Each answer binds all query variables by constants.

Example 2.7. *For the query* $Q(D)$ *of Example 2.6 and the deduction rule of Example 2.2 there is only one answer, namely* FamousDirector(Coppola)*.*

2.2 Probabilistic Databases

We now move on to present probabilistic databases which extend the relational databases of the previous section by probabilities.

2.2.1 Possible Worlds Semantics

In this section, we relax the common assumption in databases that all tuples do certainly exist. Depending on the existing tuples, a database can be in different states. Each such state is called a *possible world* [3, 139].

Definition 2.8. *For a database with tuples* $\mathcal{T}$*, a* possible world *is a subset* $\mathcal{W} \subseteq \mathcal{T}$*.*

The interpretation of a possible world is as follows: all tuples in $\mathcal{W}$ exist, whereas all tuples in $\mathcal{T}\backslash\mathcal{W}$ do not exist. In the absence of any constraints that would restrict this set of possible worlds (see Section 2.2.6), any subset $\mathcal{W}$ of tuples in $\mathcal{T}$ forms a valid possible world (i.e., a possible instance) of the probabilistic database. Hence, there are $2^{|\mathcal{T}|}$ possible worlds.

Example 2.8. *Considering the database of Example 2.1, a possible world is* $\mathcal{W} := \{I_2, I_4, I_6\}$*, which hence has only one tuple in the* Directed *relation and two tuples in the* WonAward *relation.*

2.2.2 Probabilistic Database

Based on the possible worlds semantics, we introduce probabilistic databases [139], which associate a probability with each possible world.

Definition 2.9. *Given a set of tuples* $\mathcal{T}$ *with possible worlds* $\mathcal{W}_1, \dots, \mathcal{W}_n$*, a* probabilistic database *assigns a probability* $P : 2^{\mathcal{T}} \rightarrow [0,1]$ *to each possible world* $\mathcal{W} \subseteq \mathcal{T}$*, such that:*

$$\sum_{\mathcal{W} \subseteq \mathcal{T}} P(\mathcal{W}) = 1$$

In other words, in a probabilistic database the probabilities of the possible worlds $P(\mathcal{W})$ form a probability distribution. Thus, each possible world can be seen as the outcome of a probabilistic experiment.

Example 2.9. *If we allow only two possible worlds* $\mathcal{W}_1 := \{I_1, I_3, I_5, I_7\}$ *and* $\mathcal{W}_2 := \{I_2, I_4, I_6\}$ *over the tuples of Example 2.1, we can set their probabilities to* $P(\mathcal{W}_1) = 0.4$ *and* $P(\mathcal{W}_2) = 0.6$ *to obtain a valid probabilistic database.*

2.2.3 Tuple-Independence

Because there can be exponentially many possible worlds, it is prohibitive to store every possible world along with its probability in a database system. Instead, we opt for a simpler method, that is, annotating each tuple with a probability value. We assume the probabilities of all tuples to be independent [45, 130], which yields *tuple-independent probabilistic databases* [29, 139]

Definition 2.10. *For tuples* $\mathcal{T}$ *, a* tuple-independent probabilistic database $(\mathcal{T}, p)$ *is a pair, where*

1. *p is a function* $p : \mathcal{T} \rightarrow (0,1]$*, which assigns a non-zero* probability value $p(I)$ *to each tuple* $I \in \mathcal{T}$*;*

2. *the probability values of all tuples in* $\mathcal{T}$ *are assumed to be* independent*;*

3. every subset $\mathcal{W} \subseteq \mathcal{T}$ is a possible world and has probability:

$$P(\mathcal{W}, \mathcal{T}) := \prod_{I \in \mathcal{W}} p(I) \cdot \prod_{I \in \mathcal{T} \setminus \mathcal{W}} (1 - p(I))$$

The probability $p(I)$ of a tuple I denotes the confidence in the existence of the tuple in the database, i.e., a higher value $p(I)$ denotes a higher confidence in I being valid. However, the probabilities of different tuples do not depend on each other, that is, they are probabilistically independent. This allows us to multiply the probabilities of the tuples to obtain the probability of the possible world. In addition, from a probabilistic perspective, each tuple corresponds to a binary random variable.

Example 2.10. *Assume we are unsure about the existence of each tuple in Example 2.1, hence we annotate them by probabilities.*

Directed

	Director	Movie	p
I_1	*Coppola*	*ApocalypseNow*	0.7
I_2	*Coppola*	*Godfather*	0.5
I_3	*Tarantino*	*PulpFiction*	0.2

WonAward

	Movie	Award	p
I_4	*ApocalypseNow*	*BestScript*	0.1
I_5	*Godfather*	*BestDirector*	0.8
I_6	*Godfather*	*BestPicture*	0.9
I_7	*PulpFiction*	*BestPicture*	0.5

Here, Coppola *directed the movie* Godfather *only with probability* 0.5*. In addition, the possible world $\mathcal{W} := \{I_1, I_3, I_5, I_7\}$ has the probability:*

$$P(\mathcal{W}, \{I_1, \ldots, I_9\}) = 0.7 \cdot (1 - 0.5) \cdot 0.2 \cdot (1 - 0.1) \cdot 0.8 \cdot (1 - 0.9) \cdot 0.5 = 0.00252$$

In Section 2.2.2 we required probabilistic databases to form a probability distribution over possible worlds. For tuple-independent probabilistic databases, we can prove this condition.

Proposition 2.1. *Given a tuple-independent probabilistic database $(\mathcal{T}, p)$, then $P(\mathcal{W}, \mathcal{T})$ of Definition 2.10 is a probability distribution over the possible worlds $\mathcal{W} \subseteq \mathcal{T}$:*

$$\sum_{\mathcal{W} \subseteq \mathcal{T}} P(\mathcal{W}, \mathcal{T}) = 1$$

Proof. We prove the proposition by induction over the cardinality of $\mathcal{T}$.
<u>Basis i = 1:</u> $\sum_{\mathcal{W} \subseteq \{I_1\}} P(\mathcal{W}, \{I_1\}) = p(I_1) + (1 - p(I_1)) = 1$
<u>Step $(i-1) \to i$:</u> Let $\mathcal{T} := \{I_1, \ldots, I_i\}$ where I_i is the new tuple.

$$\begin{aligned} &\textstyle\sum_{\mathcal{W} \subseteq \mathcal{T}} P(\mathcal{W}, \mathcal{T}) \\ &\quad = \textstyle\sum_{\mathcal{W} \subseteq \mathcal{T}} \prod_{I \in \mathcal{W}} p(I) \cdot \prod_{I \in \mathcal{T} \setminus \mathcal{W}} (1 - p(I)) \\ &\quad = \underbrace{(p(I_i) + (1 - p(I_i)))}_{=1} \cdot \underbrace{\textstyle\sum_{\mathcal{W} \subseteq \mathcal{T} \setminus \{I_i\}} \prod_{I \in \mathcal{W}} p(I) \cdot \prod_{I \in \mathcal{T} \setminus \mathcal{W}} (1 - p(I))}_{=1 \text{ by hypothesis}} \end{aligned}$$

□

In the remaining parts of the thesis, unless stated otherwise, we will always refer to tuple-independent probabilistic databases when we mention probabilistic databases.

2.2.4 Propositional Lineage

In this section, we introduce how to trace the derivation history of new tuples. In database terminology this is called *data lineage* [14, 26, 125], 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 $\neg$, which reflect the semantics of the relational operations that were applied to derive that tuple.

Definition 2.11. *We establish lineage inductively via the function*

$$\lambda : \textit{GroundLiterals} \rightarrow \textit{Lineage}$$

which is defined as follows:

1. *For tuples $\mathcal{T}$ and $R(\bar{a})$ with R being* extensional *and $R(\bar{a}) \in \mathcal{T}$ we have*

   $$\lambda(R(\bar{a})) := I$$

   where I is a Boolean (random) variable representing the validity of the tuple.

2. *For tuples $\mathcal{T}$, deduction rules $\mathcal{D}$, and $R(\bar{a})$ with R being* intensional*, lineage is defined as*

   $$\lambda(R(\bar{a})) := \bigvee_{\substack{D \in \mathcal{D},\\ \sigma \in G(D,\mathcal{T}),\\ \sigma(\bar{X}_0)=\bar{a}}} \left(\bigwedge_{i=1,\ldots,n} \lambda(\sigma(R_i(\bar{X}_i))) \quad \wedge \bigwedge_{\sigma(R_j(\bar{X}_j)) \in \mathcal{T}} \neg\lambda(\sigma(R_j(\bar{X}_j))) \right)$$

   where D is a deduction rule with R in its head literal:

   $$R(\bar{X}_0) \leftarrow \bigwedge_{i=1,\ldots,n} R_i(\bar{X}_i) \wedge \bigwedge_{j=1,\ldots,m} \neg R_j(\bar{X}_j) \quad \wedge \Phi(\bar{X})$$

3. *If there is* no match *to $R(\bar{a})$ in both $\mathcal{T}$ and $\mathcal{D}$:*

   $$\lambda(R(\bar{a})) := \textit{false}$$

In the first case, we replace a ground literal $R(\bar{a})$ by a Boolean random variable representing the database tuple. The second case is a little more involved: the ground literal $R(\bar{a})$ is replaced by the disjunction over all deduction rules and all groundings of these, where the grounded head literal matched $R(\bar{a})$. Likewise, negative literals are only traced, if they occur in the tuples. which resembles In the third case all literals not being matched at all are replaced by *false*, which resembles the *closed world assumption* [2]. Finally, arithmetic literals do not occur in the lineage formulas, since the groundings evaluate them to *true* (see Definition 2.4). Similarly, because queries have the shape of deduction rule bodies, we write $\lambda(Q(\bar{a}))$ to refer to the lineage formula of a query answer.

Example 2.11. *Building on Examples 2.4 and 2.5, we determine the lineage of the tuple* FamousDirector(Coppola)*, which was produced by the three substitutions σ_1, σ_2, and σ_3. The second case of Definition 2.11 delivers a disjunction ranging over both substitutions:*

$$\begin{array}{ll}
\multicolumn{2}{l}{\lambda(\mathit{FamousDirector}(\mathit{Coppola}) =} \\
\left(\begin{array}{c} \lambda(\mathit{Directed}(\mathit{Coppola}, \mathit{ApocalypseNow})) \\ \wedge\ \lambda(\mathit{WonAward}(\mathit{ApocalypseNow}, \mathit{BestScript})) \end{array}\right) & \textit{from } \sigma_1 \\
\vee & \\
\left(\begin{array}{c} \lambda(\mathit{Directed}(\mathit{Coppola}, \mathit{Godfather})) \\ \wedge\ \lambda(\mathit{WonAward}(\mathit{Godfather}, \mathit{BestDirector})) \end{array}\right) & \textit{from } \sigma_2 \\
\vee & \\
\left(\begin{array}{c} \lambda(\mathit{Directed}(\mathit{Coppola}, \mathit{Godfather})) \\ \wedge\ \lambda(\mathit{WonAward}(\mathit{Godfather}, \mathit{BestPicture})) \end{array}\right) & \textit{from } \sigma_3
\end{array}$$

Then, the first case of Definition 2.11 replaces all ground literals by their tuple identifiers:

$$\underbrace{(I_1 \wedge I_4)}_{\textit{from } \sigma_1} \vee \underbrace{(I_2 \wedge I_5)}_{\textit{from } \sigma_2} \vee \underbrace{(I_2 \wedge I_6)}_{\textit{from } \sigma_3}$$

Next, we study the computational complexity of lineage tracing. It is known that grounding non-recursive Datalog rules, which coincides with our class of deduction rules, has polynomial data complexity [78]. Now, we extend this result to lineage tracing.

Lemma 2.1. *For a fixed set of deduction rules $\mathcal{D}$, grounding with lineage as of Definition 2.11 has polynomial data complexity in $|\mathcal{T}|$.*

Proof. We have to show, that Definition 2.11 creates an overhead which is polynomial in $|\mathcal{T}|$. In the first and third case of the definition, we solely rely on a look-up in $\mathcal{D}$ or $\mathcal{T}$, which is computable in polynomial time. The second case iterates over all deduction rules $D \in \mathcal{D}$. For each deduction rule D, it performs a number look-ups bound by $|G(D, \mathcal{T})| \cdot |D|$. Since grounding has polynomial data complexity, $G(D, \mathcal{T})$ is of polynomial size in $\mathcal{T}$. Thus, also the third case has polynomial data complexity. $\square$

We next introduce a normal form of propositional lineage formulas, which is very common in logic [133]. Assuming lineage formulas to be in a normal form will simplify proofs that follow later on.

Definition 2.12. *A propositional lineage formula ϕ is in* disjunctive normal form (DNF) *if $\phi = \psi_1 \vee \cdots \vee \psi_n$ and each clause ψ_i is of the form $\bigwedge_j I_j \wedge \bigwedge_k I_k$.*

For illustration, the lineage formula of Example 2.11 is in disjunctive normal form. In general, any propositional formula can be transformed into disjunctive normal form [133], which we rely on in order to show the following statement.

Proposition 2.2. *The deduction rules of Definition 2.2 allow us to express any propositional lineage formula.*

Proof. Consider a probabilistic database $(\mathcal{T}, p)$ and an arbitrary propositional formula ϕ connecting tuple identifiers. Without loss of generality, let the formula ϕ be in disjunctive normal form and range over only one relation R. First, we introduce one additional tuple $R(0)$ and set $p(R(0)) = 1$. Then, for each clause $\psi_i = \bigwedge_j I_j \ \wedge \bigwedge_k \neg I_k$ of ϕ, we create one deduction rule:

$$R'(0) \leftarrow R(0) \wedge \bigwedge_j R(j) \ \wedge \bigwedge_k \neg R(k)$$

The lineage formula of the intensional tuple $R'(0)$ is exactly ϕ. The reason is that each rule creates one clause. Then, these clauses are connected by a disjunction originating from the second case of Definition 2.11. □

From the above statement, it follows that the lineage formulas considered in this work take more general forms than lineage formulas resulting from (unions) of conjunctive queries [28, 31], which produce only disjunctive normal forms which are restricted to positive literals.

2.2.5 Probability Computation

Since in a probabilistic database, each tuple exists only with a given probability, we can quantify the probability that each answer exists, which is the topic of this section. Based on [52, 125, 139] we compute probabilities of query answers via their lineage formulas. To achieve this, we interpret the propositional lineage formulas over a possible world of a probabilistic database $(\mathcal{T}, p)$ as follows. We say that a possible world $\mathcal{W}$ is a *model* [146] for a propositional lineage formula ϕ, denoted as $\mathcal{W} \models \phi$, if, by setting all tuples in $\mathcal{W}$ to *true* and all tuples in $\mathcal{T} \backslash \mathcal{W}$ to *false*, $\mathcal{W}$ represents a truth assignment that satisfies ϕ. Moreover, let the set $\mathcal{M}(\phi, \mathcal{T})$ contain all possible worlds $\mathcal{W} \subseteq \mathcal{T}$ being a model for a propositional lineage formula ϕ.

$$\mathcal{M}(\phi, \mathcal{T}) := \{\mathcal{W} \mid \mathcal{W} \subseteq \mathcal{T}, \mathcal{W} \models \phi\} \tag{2.2}$$

If it is clear from the context, we drop $\mathcal{T}$ as an argument of $\mathcal{M}$. We compute the *probability* of any Boolean formula ϕ over tuples in $\mathcal{T}$ as the sum of the probabilities of all the possible worlds that are a model for ϕ:

$$P(\phi) := \sum_{\mathcal{W} \in \mathcal{M}(\phi, \mathcal{T})} P(\mathcal{W}, \mathcal{T}) \tag{2.3}$$

Here, $P(\mathcal{W}, \mathcal{T})$ is as in Definition 2.10. We can interpret the above probability as the marginal probability of the lineage formula ϕ. The above sum can range over exponentially many terms. However, in practice, we can—in many cases—compute the probability $P(\phi)$ 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 computations can be employed:

$$
\begin{array}{r|c}
\text{Definition} & \text{Condition} \\
\hline
P(I) := p(I) & I \in \mathcal{T} \\
P(\bigwedge_i \phi_i) := \prod_i P(\phi_i) & i \neq j \Rightarrow Tup(\phi_i) \cap Tup(\phi_j) = \emptyset \\
P(\bigvee_i \phi_i) := 1 - \prod_i (1 - P(\phi_i)) & i \neq j \Rightarrow Tup(\phi_i) \cap Tup(\phi_j) = \emptyset \\
P(\phi \vee \psi) := P(\phi) + P(\psi) & \phi \wedge \psi \equiv \mathit{false} \\
P(\neg \phi) := 1 - P(\phi) & \\
P(\mathit{true}) := 1 & \\
P(\mathit{false}) := 0 &
\end{array}
\tag{2.4}
$$

The first line captures the case of a tuple I, for which we return its attached probability value $p(I)$. The next two lines handle *independent-and* and *independent-or* operations for conjunctions and disjunctions over tuple-disjoint subformulas ϕ_i, respectively. In the forth line, we address disjunctions for subformulas ϕ and ψ that denote disjoint probabilistic events (*disjoint-or*). The fifth line handles negation. Finally, the probability of *true* and *false* is 1 and 0, respectively.

Example 2.12. *Let us compute the probability $P(I_1 \wedge I_2 \wedge \neg I_3)$ over the tuples of Example 2.10. First, the second line of Equation* (2.4) *is applicable, which yields $P(I_1) \cdot P(I_2) \cdot P(\neg I_3)$. Then, we replace the negation and obtain $P(I_1) \cdot P(I_2) \cdot (1 - P(I_3))$. Now, looking up the tuples' probability values in Example 2.10 yields* $0.7 \cdot 0.5 \cdot (1 - 0.2) = 0.28$.

The definition of $P(\phi)$ presented in Equation (2.4) can be evaluated in linear time in the size of ϕ. However, for general lineage formulas, computing $P(\phi)$ is $\#\mathcal{P}$-hard [31, 29, 105]. Here, $\#\mathcal{P}$ [147] is the class of counting problems. Its prototypical problem #SAT asks for the number of satisfying assignments of a propositional formula. We next present a number of deduction rules which can yield lineage formulas exhibiting hard probability computations.

Lemma 2.2. *Let a probabilistic database $(\mathcal{T}, p)$ and the following deduction rules be given:*

$$H(0) \leftarrow R(X) \wedge S(X,Y) \wedge T(Y)$$

$$\begin{aligned} H(1) &\leftarrow R(X) \wedge S(X,Y) \\ H(1) &\leftarrow S(X,Y) \wedge T(Y) \end{aligned}$$

$$\begin{aligned} H(2) &\leftarrow R(X) \wedge S_1(X,Y) \\ H(2) &\leftarrow S_1(X,Y) \wedge S_2(X,Y) \\ H(2) &\leftarrow S_2(X,Y) \wedge T(Y) \end{aligned}$$

$$
\begin{aligned}
&H(3) \leftarrow R(X) \wedge S_1(X,Y)\\
&H(3) \leftarrow S_1(X,Y) \wedge S_2(X,Y)\\
&H(3) \leftarrow S_2(X,Y) \wedge S_3(X,Y)\\
&H(3) \leftarrow S_3(X,Y) \wedge T(Y)\\
&\ldots
\end{aligned}
$$

Then, for each $H(k)$ the corresponding probability computations $P(\lambda(H_k))$ are $\#\mathcal{P}$-hard in $|\mathcal{T}|$.

In the lemma above, k is a constant, hence, $H(0)$ is a ground literal resembling a Boolean query. A proof for the above statement can be found in [31]. To address the computationally hard cases, we employ the following equation, called *Shannon expansion*, which is applicable to any propositional lineage formula:

$$P(\phi) := p(I) \cdot P(\phi_{[I/true]}) + (1 - p(I)) \cdot P(\phi_{[I/false]}) \tag{2.5}$$

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*. Shannon expansion is based on the following logical equivalence:

$$\phi \equiv (I \wedge \phi_{[I/true]}) \vee (\neg I \wedge \phi_{[I/false]}) \tag{2.6}$$

The resulting disjunction fulfills the disjoint-or condition (see Equation (2.4)) with respect to I. Repeated applications of Shannon expansions may however increase the size of ϕ exponentially, hence do not circumvent the computational hardness.

Example 2.13. *We calculate the probability of the lineage formula of Example 2.11 as follows:*

$$P((I_1 \wedge I_4) \vee (I_2 \wedge I_5) \vee (I_2 \wedge I_6))$$

The top-level operator is a disjunction where the third line of Equation (2.4) *is not applicable, since I_2 occurs in two subformulas. Hence, we first apply a Shannon expansion for I_2:*

$$p(I_2) \cdot P((I_1 \wedge I_4) \vee I_5 \vee I_6) + (1 - p(I_2)) \cdot P(I_1 \wedge I_4)$$

Now, we can resolve the disjunction and the conjunction by independent-or and independent-and, respectively:

$$p(I_2) \cdot (1 - (1 - p(I_1) \cdot p(I_4)) \cdot (1 - p(I_5)) \cdot (1 - p(I_6))) + (1 - p(I_2)) \cdot p(I_1) \cdot p(I_4)$$

Derivative of Probability Computations

As introduced in [77, 116], we can quantify the impact of the probability of a tuple $p(I)$ on the probability $P(\phi)$ of a propositional lineage formula ϕ by its partial derivative.

Definition 2.13. *Given a propositional lineage formula ϕ and a tuple $I \in Tup(\phi)$, the partial derivative of $P(\phi)$ with respect to $p(I)$ is*

$$\frac{\partial P(\phi)}{\partial p(I)} := \frac{P(\phi_{[I/true]}) - P(\phi_{[I/false]})}{P(true) - P(false)} = P(\phi_{[I/true]}) - P(\phi_{[I/false]})$$

Again, $\phi_{[I/true]}$ means that all occurrences of I in ϕ are replaced by *true* and analogously for *false.*

Example 2.14. *We determine the derivative of the probability of the propositional lineage formula $\phi := I_1 \wedge I_4$ with respect to the tuple I_4:*

$$\begin{aligned}\tfrac{\partial P(\phi)}{\partial p(I_4)} &= P((I_1 \wedge I_4)_{[I_4/true]}) - P((I_1 \wedge I_4)_{[I_4/false]})\\ &= p(I_1) - P(false)\\ &= p(I_1)\end{aligned}$$

2.2.6 Constraints

To rule out cases which are inconsistent with the real world, we support consistency constraints. For instance, if for the same person two places of birth are stored in the database, then we intend to remove one of them by a consistency constraint. In general, we consider the constraints to be presented in the form of a single propositional lineage formula ϕ_c, which connects tuple identifiers. Intuitively, the constraint formula ϕ_c describes all possible worlds that are valid. In contrast, all possible worlds that do not satisfy the constraint will be dropped from the probability computations. Because it is tedious to manually formulate a propositional formula over many database tuples, we allow ϕ_c to be induced by deduction rules $\mathcal{D}_c$ and two sets of queries $\mathcal{C}_p$ and $\mathcal{C}_n$ as follows. For simplicity, we assume $\mathcal{C}_p \cap \mathcal{C}_n = \emptyset$ and $\mathcal{D}_c \cap \mathcal{D}_q = \emptyset$, where $\mathcal{D}_q$ are the deduction rules of the query.

Definition 2.14. *Let a set of deduction rules $\mathcal{D}_c$ and two sets $\mathcal{C}_p$ and $\mathcal{C}_n$ of intensional literals from $\mathcal{D}_c$ be given. If $\mathcal{T}$ contains all tuples deducible by $\mathcal{D}_c$ the constraint formula ϕ_c is obtained by:*

$$\phi_c := \bigwedge_{\substack{C_p(\bar{X}) \in \mathcal{C}_p,\\ C_p(\bar{a}) \in Answers(C_p(\bar{X}), \mathcal{T})}} \lambda(C_p(\bar{a})) \quad \wedge \bigwedge_{\substack{C_p(\bar{X}) \in \mathcal{C}_n,\\ C_n(\bar{a}) \in Answers(C_n(\bar{X}), \mathcal{T})}} \neg\lambda(C_n(\bar{a}))$$

Hence, based on the above definition, we create constraints on probabilistic databases by deduction rules. All answers from literals in $\mathcal{C}_p$ yield propositional lineage formulas, which must always hold, whereas the lineage formulas being derived from literals in $\mathcal{C}_n$ may never hold. We connect all these grounded constraints, i.e. their lineage formulas, by a conjunction to enforce all of them together. It is important to note that the deduction rules of the constraints do not create any new tuples, but merely serve the purpose of creating ϕ_c.

Example 2.15. *Let us formalize that every movie is directed by only one person. We create the deduction rule*

$$\mathit{Constraint}(P_1, P_2, M) \leftarrow (\mathit{Directed}(P_1, M) \wedge \mathit{Directed}(P_2, M) \wedge P_1 \neq P_2)$$

and insert Constraint(P_1, P_2, M) *into $\mathcal{C}_n$, which hence disallows the existence of two persons P_1 and P_2 both directing the same movie.*

For brevity, we also abbreviate constraints consisting of a single deduction rule by the rule body only, that is neglecting the head literal.

Example 2.16. *We can write the constraint of Example 2.15 without the head literal as follows:*

$$\neg(\mathit{Directed}(P_1, M) \wedge \mathit{Directed}(P_2, M) \wedge P_1 \neq P_2)$$

Here the negation signalizes that the former head literal was in $\mathcal{C}_n$.

With respect to the probability computations, constraints remove all possible worlds from the computations, which violate the constraint. This process is called *conditioning* [83], which formally reads as follows.

Definition 2.15. *Let constraints be given as a propositional lineage formula ϕ_c over a probabilistic database $(\mathcal{T}, p)$. If ϕ_c is satisfiable, then the probability $P(\psi)$ of a propositional lineage formula ψ over $\mathcal{T}$ can be* conditioned *onto ϕ_c as follows:*

$$P(\psi \mid \phi_c) := \frac{P(\psi \wedge \phi_c)}{P(\phi_c)} \tag{2.7}$$

In the above definition, ψ can represent any lineage formula, in particular a query answer. After removing the possible worlds violating a constraint from the probabilistic database, conditioning reweights the remaining worlds such that they again form a probability distribution.

Example 2.17. *We consider the lineage formula $\psi = I_2 \wedge (I_5 \vee I_6)$ over the tuples of Example 2.10. Without any constraints, its probability is computed by Equation* (2.4) *as $P(\psi) = 0.5 \cdot (1 - (1 - 0.8) \cdot (1 - 0.9)) = 0.49$. If we set $\phi_c = I_2$ as a constraint, we remove all possible worlds that exclude I_2. Consequently, the probability is updated to:*

$$P(\psi \mid I_2) = \frac{P(I_2 \wedge (I_5 \vee I_6))}{P(I_2)} = \frac{p(I_2) \cdot P(I_5 \vee I_6)}{p(I_2)} = P(I_5 \vee I_6) = 0.98$$

In the following, we characterize a useful property of constraints. If a number of constraints do not share any tuple with a lineage formula ψ, then the probability $P(\psi)$ is not affected by the constraints.

Proposition 2.3. *If the constraints ϕ_c and the lineage formula ψ are independent with respect to tuples, i.e. $Tup(\psi) \cap Tup(\phi_c) = \emptyset$, then it holds that:*

$$P(\psi \mid \phi_c) = P(\psi)$$

Proof. Due to the second line of Equation (2.4) and $Tup(\psi) \cap Tup(\phi_c) = \emptyset$ we can write $P(\psi \wedge \phi_c) = P(\psi) \cdot P(\phi_c)$. Therefore, the following equation holds:

$$P(\psi \mid \phi_c) = \frac{P(\psi \wedge \phi_c)}{P(\phi_c)} = P(\psi) \cdot \frac{P(\phi_c)}{P(\phi_c)} = P(\psi)$$

□

Hence, if we have the constraint $\phi_c \equiv true$, the standard unconditioned probability computations of Section 2.2.5 arise as special case. Finally, since Equation (2.7) invokes probability computations on the constraint ϕ_c, constraints can yield $\#\mathcal{P}$-hard computations, which we capture next.

Observation 2.1. *Constraints can cause $\#\mathcal{P}$-hard probability computations.*

The reason is that one of the lineage formulas described in Lemma 2.2 could occur in ϕ_c.

Expressiveness The deduction rules of Definition 2.14 inducing the constraints can yield any propositional lineage formula, as formally shown in Proposition 2.2. We note that restrictions on the shape of the constraints, i.e. to avoid the $\#\mathcal{P}$-hardness of Observation 2.1, should follow work on tractable probability computations in probabilistic databases. The reason is that the complexity arises from the probability computations. In contrast, when solving constraints over deterministic databases, the complexity mainly results from finding a consistent subset of the database, rather than counting these subsets.

2.3 Algorithms

In this section, we establish algorithms for query answering with constraints (in Section 2.3.2), and without constraints (in Section 2.3.1). These correspond to procedural approaches implementing the previously introduced definitions.

2.3.1 Query Answering

We now describe how to compute query answers including their lineage from given deduction rules and tuples. The idea is to start at the database tuples. Then, we process deduction rules where all relations in the body were already considered and produce the new tuples from the head relation. This approach is usually referred to as *bottom-up* [2] grounding and is established in probabilistic databases [14, 125].

In detail, Algorithm 1 first collects all intensional relations occurring in the deduction rules in the set *In* as described in Line 2. Then, we iterate

Algorithm 1 QueryAnswering($\mathcal{T}, \mathcal{D}, Q(\bar{X})$)

```
Input: Tuples $\mathcal{T}$, deduction rules $\mathcal{D}$, query $Q(\bar{X})$
Output: Query answers with probabilities and lineage
 1: $\mathcal{T}_{closure} := \mathcal{T}$
 2: $In := \{R_0 \mid (R_0(\bar{X}_0) \leftarrow \Psi) \in \mathcal{D}\}$      ▷ All intensional relations
 3: while $In \neq \emptyset$ do
 4:     choose $R \in In$ such that                                    ▷ Bottom-up manner
 5:         $\{R_i, R_j \mid (R \leftarrow \bigwedge_i R_i \wedge \bigwedge_j \neg R_j) \in \mathcal{D}\} \cap In = \emptyset$
 6:     $In := In \backslash \{R\}$
 7:     $\mathcal{T}_{new} := \emptyset$
 8:     for $D = (R(\bar{X}) \leftarrow \Psi) \in \mathcal{D}$ do                ▷ Create new tuples
 9:         $\mathcal{T}_{new} := \mathcal{T}_{new} \cup NewTuples(D, \mathcal{T}_{closure})$   ▷ See Definition 2.5
10:     for $R(\bar{a}) \in \mathcal{T}_{new}$ do                              ▷ Trace lineage
11:         set $\lambda(R(\bar{a}))$                                      ▷ See Definition 2.11
12:     $\mathcal{T}_{closure} := \mathcal{T}_{closure} \cup \mathcal{T}_{new}$
13: determine $Answers(Q(\bar{X}), \mathcal{T}_{closure})$                    ▷ See Definition 2.7
14: for $Q(\bar{a}) \in Answers$ do                                      ▷ Trace lineage
15:     set $\lambda(Q(\bar{a}))$                                          ▷ See Definition 2.11
16: for $Q(\bar{a}) \in Answers$ do                               ▷ Compute probabilities
17:     compute $P(\lambda(Q(\bar{a})))$                                    ▷ See Section 2.2.5
18: return $Answers$
```

over all intensional relations (Line 3), every time picking a relation R where all relations of all rules defining R are either extensional or are already processed (Lines 4 and 5). Now, we instantiate the new tuples of R (Line 8) and trace their lineage (Line 11). Finally, when the while loop terminates, we compute the query answers based on all new tuples (Line 13), and afterwards we determine their lineage (Line 15) in order to compute the probability of each answer (Line 17). Of course, in addition we can apply any optimization known from databases [65] to Algorithm 1, such as propagating query constants through the rules or join-order optimization.

Example 2.18. *We execute Algorithm 1 for the deduction rule of Equation* (2.2) *and the query* FamousDirector(D) *over the tuples of Example 2.10. First, Line 2 collects all intensional relations, which yields* In = {Famous-

Director}. *Hence, we can choose* R := FamousDirector *in Line 4. The reason is that its defining deduction rule has only extensional relations in its body. Therefore, the intersection of Line 5 is empty. The loop of Line 8 iterates only over the deduction rule of Equation* (2.1) *which delivers the new tuples along with their lineage in Line 11:*

Tuple	Lineage
$FamousDirector(Coppola)$	$(I_1 \wedge I_4) \vee (I_2 \wedge I_5) \vee (I_2 \wedge I_6)$
$FamousDirector(Tarantino)$	$I_3 \wedge I_7$

Finally, Lines 13 to 15 return the above tuples as results, followed by probability computations of the lineage formulas.

2.3.2 Query Answering with Constraints

We now extend Algorithm 1 to support constraints besides queries. For that, we assume an additional set of deduction rules $\mathcal{D}_c$ defining intensional relations for the constraints, along with two sets of literals $\mathcal{C}_p$ and $\mathcal{C}_n$ to be given (see Section 2.2.6). Grounding the literals in $\mathcal{C}_p$ and $\mathcal{C}_n$ delivers the propositional lineage formulas resembling positive and negated constraints, respectively.

The detailed procedure is captured in Algorithm 2. First, in Line 1 we invoke Algorithm 1, which hence computes all query answers along with their lineage, but neglects the constraints[1]. Lines 2 to 7 refer the grounding of the constraints to Algorithm 1, again. It remains to combine all obtained lineage formulas, such that the answers' probabilities are computed via Equation (2.7). Hence, in Lines 8 to 11, we construct the conjunction of constraint lineage formulas ϕ_c. The following line precomputes $P(\phi_c)$, because it will be repeatedly invoked. In Line 14 we compute the probability of all query answers while respecting the constraints. To conclude, it is noteworthy to remark that Algorithm 1 results as a special case of Algorithm 2 if there are no constraints present.

Example 2.19. *Let us extend Example 2.18 by setting the constraint that all awards are certainly won, which we achieve by* $\mathcal{C}_p := \{C(M, A)\}$ *and by the deduction rule:*

$$C(M, A) \leftarrow WonAward(M, A)$$

First, in Line 1 we invoke Algorithm 2 which computes the query answers as in Example 2.18. In the next line, we ground the constraints yielding:

$$Constraints = \left\{ \begin{array}{c} C(ApocalypseNow, BestScript), C(Godfather, BestDirector), \\ C(Godfather, BestPicture),\ C(PulpFiction, BestPicture) \end{array} \right\}$$

[1]It suffices to run Algorithm 1 from Line 1 to 15 only.

Algorithm 2 QueryAnsweringWithConstraints($\mathcal{T}, \mathcal{D}_q, Q(\bar{X}'), \mathcal{D}_c, \mathcal{C}_p, \mathcal{C}_n$)

```
Input: Tuples $\mathcal{T}$, query deduction rules $\mathcal{D}_q$, query $Q(\bar{X}')$, constraint deduction
    rules $\mathcal{D}_c$, constraints $C(\bar{X})$
Output: Query answers with probabilities
 1: $Answers := QueryAnswering(\mathcal{T}, \mathcal{D}_q, Q(\bar{X}'))$                ▷ See Algorithm 1
 2: $Constraints_p := \emptyset$
 3: for $C_p(\bar{X}) \in \mathcal{C}_p$ do
 4:     $Constraints_p := QueryAnswering(\mathcal{T}, \mathcal{D}_c, C_p(\bar{X}))$       ▷ See Algorithm 1
 5: $Constraints_n := \emptyset$
 6: for $C_n(\bar{X}) \in \mathcal{C}_n$ do
 7:     $Constraints_n := QueryAnswering(\mathcal{T}, \mathcal{D}_c, C_n(\bar{X}))$       ▷ See Algorithm 1
 8: if $Constraints_p \neq \emptyset$ or $Constraints_n \neq \emptyset$ then
 9:     $\phi_c := \bigwedge_{C \in Constraints_p} \lambda(C) \wedge \bigwedge_{C \in Constraints_n} \neg\lambda(C)$   ▷ See Definition 2.15
10: else
11:     $\phi_c := true$                                                   ▷ See Section 2.2.6
12: Precompute $P(\phi_c)$
13: for $Q(\bar{a}) \in Answers$ do
14:     $P(Q(\bar{a})) := \frac{P(\lambda(Q(\bar{a})) \wedge \phi_c)}{P(\phi_c)}$                       ▷ See Definition 2.15
15: return $Answers$
```

Therefore, we obtain $\phi_c = I_4 \wedge I_5 \wedge I_6 \wedge I_7$ in Line 9. We then precompute the probability of ϕ_c as $P(\phi_c) = 0.1 \cdot 0.8 \cdot 0.9 \cdot 0.5$. Based on $P(\phi_c)$ we can determine the conditioned probability of the query answers in Line 14.

2.4 Related Approaches

In this section, we present further techniques from probabilistic databases (Section 2.4.1) and related approaches, such as probabilistic XML, statistical relational learning, and probabilistic programming.

2.4.1 Probabilistic Databases

Data Model A number of representation mechanisms for probabilistic databases were investigated in the last years. Most closely related to our data model are *x-tuples* [14], also called *block-independent disjoint databases* [30], which allow non-overlapping blocks of tuples to be mutually exclusive, and hence extend tuple-independent databases. More expressive are *pc-tables* [60], which annotate each tuple by a propositional formula over random variables. With specific restrictions on pc-tables, it is possible to store them in a relational database. Then they are called *U-relations* [10]. Moreover, *pvc-tables* [46] are an extension of pc-tables which adds further support for aggregations. An alternative to represent dependencies across different tuples are *decomposition trees* [47]. Their inner nodes represent independent-and, independent-or, or mutual exclusion operators over the leafs which correspond to database tuples. Storing and querying *correlated tuples* in a probabilistic database is considered in [76, 128]. The authors express correlations among database tuples by *junction trees* and additionally allow lineage formulas on top. Likewise, in [119] factor graphs are represented by *arithmetic circuits* to support correlations. Finally, based on c-tables, [79] features *continuous* probabilities, which are not supported by most works in the field.

Probability Computations Propelled by the existence of queries exhibiting $\#\mathcal{P}$-hard probability computations [58], one major focus of research on probabilistic databases has been on efficient probability computations of query answers. One solution are *safe query plans* [29], which alter the query plan to carry out efficient probability computations while performing data computations. In [31] a dichotomy result is shown which characterizes conjunctive queries' probability computations to be in either polynomial time or $\#\mathcal{P}$-hard. In addition, the difference of two safe conjunctive queries can be unsafe, as shown in [81]. For a larger set of queries, namely unions of conjunctive queries, [28, 32] establishes an efficient algorithm either computing the probabilities in polynomial time or characterizing the query as computationally hard. Another approach is taken by works not directly acting on queries, but relying on intensional semantics, e.g. on lineage formulas. There, *read-once formulas* [129] are a class of formulas which both can be detected efficiently and also allow efficient probability computations. Another line of works is *knowledge compilation* [74, 105], which transforms lineage formulas into various forms of binary decision diagrams. In particular, for unions of conjunctive queries, [74] introduced a hierarchy of compilation target languages while rating their expressiveness. Finally, if exact probabilities for query answers are not required, one can employ *approximation* techniques as in [47, 107]. These works incrementally create decomposition trees whose probability serves as bound on the probability of the full lineage formula. Supporting *aggregations* over probabilistic databases can yield exponentially sized results, which is studied in [117]. To avoid this, the authors of [100] maintain only bounds on the aggregates. Furthermore, in [46] arithmetic expressions representing aggregations are compiled into decomposition trees, which also limits the blow-up.

Probabilistic Database Systems Most of the above approaches are implemented in existing probabilistic database systems. Many of these have been released as open-source prototypes in recent years, including *MystiQ* [18], *MayBMS* [10], *SPROUT* [106], *Trio* [14], *PrDB* [128], and *ERACER* [97].

2.4.2 Probabilistic XML

The eXtensible Markup Language (XML) is a document encoding yielding tree shaped parses. If different branches of a node in a parse tree are conditioned by independent or mutually exclusive probabilities, or even logical formulas, then the notion of probabilistic XML arises. Within this field many results are shared with probabilistic databases, since one can encode probabilistic databases and probabilistic XML into each other [8]. Most results of both fields can be transferred to each other, where the differences arise from the tree shape of the XML documents. The complexity of the query evaluation problem of the various probabilistic XML models is characterized in [82]. Additionally, [4] studies the expressiveness of these models.

2.4.3 Statistical Relational Learning

Emerging from a mixture of artificial intelligence [123] and machine learning [99], the subfield of statistical relational learning [56] has many similarities with probabilistic databases. They both tackle uncertainty in relational data which is further described by logical statements.

Markov Logic Networks The most well-known statistical relational learning approach are Markov Logic Networks (MLNs) [120], which have been subject to many improvements, e.g. [102, 121, 132]. Their grounding mechanism proceeds via instantiating literals with all possible constants based on the literals' type signatures. This is commonly referred to as open-world assumption. In contrast, probabilistic databases rely on the closed world assumption with more selective grounding via deduction rules, which usually results in much fewer grounded tuples, and hence in much better scalability. In Markov logic networks, dependencies between grounded literals are induced by weighted logical clauses where all literals have an equal impact on the truth value of the clause. All these clauses are kept in a large conjunctive formula resembling an undirected graphical model. Conversely, probabilistic databases use unweighted lineage formulas, where all input tuples fully determine the truth value of the output tuple. Hence, in this case the lineage formulas take acyclic directed structures, as opposed to one conjunctive formula in Markov logic networks. A step to merge both worlds was taken in MarkoViews [73] where probabilistic databases are extended by uncertain views inspired by Markov logic networks. Finally, we compete against Markov logic networks implementations throughout the experiments in this thesis.

2.4.4 Probabilistic Programming

In probabilistic programming known programming frameworks, such as Prolog [90], are extended by probabilities.

ProbLog Most closely related to our data model, tuple-independent probabilistic databases with lineage, is ProbLog [33], where the similarities are discussed in [143]. Their input facts are annotated with independent probabilities and are correlated by rules known from Prolog [90]. After executing the rules, ProbLog keeps the SLD proofs to perform probability computations, which can be viewed as lineage tracing in probabilistic databases. The major difference is that probabilistic databases return all answers to a query with their respective probability, whereas ProbLog queries rather ask whether an answer exists at all. Because of these similarities, ProbLog will be a competitor in our experiments.

2.5 Application: Information Extraction

One of the major application domains of probabilistic databases is information extraction [155]. Its goal is to harvest factual knowledge from free-text to bring it into a more machine-readable format, i.e. into database tuples. For example, the sentence "Spielberg won the Academy Award for Best Director for Schindler's List (1993) and Saving Private Ryan (1998)" from Steven Spielberg's Wikipedia page[2], contains the knowledge that *Spielberg* won an *AcademyAward*, which we could represent as *WonAward*(*Spielberg*, *AcademyAward*). Due to the many ways of rephrasing in natural language, an automatic machinery mining the facts from text produces some erroneous facts. So, its resulting data is never clean, but rather uncertain. Since the web is full of text and facts, managing the vast amounts of extracted and hence uncertain facts is a prime application of probabilistic databases.

For illustration, we model the information extraction process in a probabilistic database. Usually candidates for facts in sentences are detected by textual patterns [19, 101]. For instance, for winning an award, the single word "won" might be a good indicator. In our probabilistic database we want to capture the different ingredients that yield to the extraction of a fact. Besides the textual pattern, this could be the web domain where we found the sentence of interest. Hence, we store these in separate probabilistic relations as shown in Figure 2.1. Therefore, the probabilities of the tuples of each domain and each pattern reflect our trust in this source and pattern, respectively. To reconcile the facts along with their resulting probabilities from the probabilistic database of Figure 2.1, we employ two deduction rules. In essence, they formulate a join on *Pid* and *Did* to connect the pattern and domain to the actual fact:

$$\mathit{WonPrize}(S,O) \leftarrow \begin{pmatrix} \mathit{WonPrizeExtraction}(S,O,\mathit{Pid},\mathit{Did}) \\ \wedge \mathit{UsingPattern}(\mathit{Pid},P) \\ \wedge \mathit{FromDomain}(\mathit{Did},D) \end{pmatrix} \tag{2.8}$$

[2]http://en.wikipedia.org/wiki/Steven_Spielberg (accessed December 22nd, 2013)

WonPrizeExtraction

	Subject	Object	Pid	Did	p
I_1	*Spielberg*	*AcademyAward*	1	1	1.0
I_2	*Spielberg*	*AcademyAward*	2	1	1.0

BornInExtraction

	Subject	Object	Pid	Did	p
I_3	*Spielberg*	*Cincinnati*	3	1	1.0
I_4	*Spielberg*	*LosAngeles*	3	2	1.0

UsingPattern

	Pid	Pattern	p
I_5	1	*Received*	0.8
I_6	2	*Won*	0.5
I_7	3	*Born*	0.9

FromDomain

	Did	Domain	p
I_8	1	*Wikipedia.org*	0.9
I_9	2	*Imdb.com*	0.8

Figure 2.1: An Example Probabilistic Database for Information Extraction

$$\mathit{BornIn}(S,O) \leftarrow \begin{pmatrix} \mathit{BornInExtraction}(S,O,\mathit{Pid},\mathit{Did}) \\ \wedge\, \mathit{UsingPattern}(\mathit{Pid},P) \\ \wedge\, \mathit{FromDomain}(\mathit{Did},D) \end{pmatrix} \tag{2.9}$$

If we execute a query on the *WonPrize* or *BornIn* relation, then the probabilities of the pattern and domain establish the probability of each fact.

Chapter 3

Temporal Probabilistic Data Model

3.1 Introduction

In recent years, temporal databases and probabilistic databases have both become highly developed research fields. In this chapter we focus on their intersection, i.e. data that is valid during a specific time with a given probability. Hence, our goal is to devise a closed and complete data model where tuples have probabilities over temporal annotations both in the database as well as in the query answers. To the present day in this setting only very few works exist which we briefly discuss next.

Uncertainty in Temporal Databases In previous works from the temporal databases field, the representation of uncertainty over time is referred to as *temporal indeterminacy* [9, 42, 142]. These works allow temporal annotations to be specified by precedence statements or by probability distributions over time. Still, running queries on top of these annotations produces either deterministic answers or, if the probabilities are propagated, the queries are restricted to joins over independent relations. Starting at temporal databases, Dekhtyar et al. [35] addresses specifically temporal probabilistic databases, but they approximate probabilities by coarse bounds. Thus, their data model is not closed and complete.

Time in Probabilistic Databases To the best of our knowledge, there has been only one work [126] on modeling time in probabilistic databases. Its focus is updating and versioning of probabilistic databases. However, this notion of time called *transaction-time* [71], is different. It captures when tuples are inserted or altered in the database. In this chapter, we consider the temporal validity of a tuple in the real world, called *valid-time* [71].

Sources of Uncertain Temporal Data There are many sources yielding both temporal and probabilistic data, which call for a system natively supporting uncertain temporal data. First, every *time measurement* is *imprecise* which results in uncertainty. For instance, every clock comes with a degree of precision. This argument holds even more for advanced techniques, such as radiocarbon dating, whose imprecision is in the order of several decades. In addition, in temporal information we often face uncertainty caused by *coarse granularities*. For many events, we might be aware of the year they took place, but might not know the exact day, for example. Likewise, in *project planning* intended completion dates tend to be loosely defined, such as "the project will complete in three to six months from now on". Even if the temporal data is precise, but it is given in an unstructured format (e.g. by written text) uncertainty can arise. The reason is that an *automated information extraction* machinery acting on the unstructured input commits errors which then induce uncertainty. Finally, *inconsistencies* over temporal data, such as finding more than one birth date for a person, produce several valid alternatives, which we can reason about probabilistically.

Our Approach In this chapter, we introduce a *closed and complete* temporal-probabilistic data model [39], which features queries as well as consistency constraints. We consider database tuples being annotated by time-intervals each of which in turn have a probability (Section 3.4.3). Over such tuples we execute queries expressed by *temporal deduction rules* (Section 3.4.5). Their results, the query answers, have the same structure as the database tuples, namely time-intervals with probabilities. Besides queries, we allow temporal *consistency constraints*, such as temporal precedence or disjointness requirements (Section 3.4.8). These constraints alter the probabilities of the time-intervals of the query answers. We perform all necessary probability computations via data *lineage* which we extend to capture both time and probabilities (Section 3.4.6).

Problem Statement In short, we are given as *input*:

- *data* which is *annotated* by both *time-intervals* and *probabilities*;
- a *query* over this data represented by deduction rules;
- and optionally consistency *constraints*.

As *output* we deliver:

- *query answers* with a probability at each time-interval.

We illustrate this setting by the following example.

BornIn

	Subject	Object	Valid Time	p
I_1	*DeNiro*	*Greenwich*	$[1943\text{-}08\text{-}17, 1943\text{-}08\text{-}18)$	0.9
I_2	*DeNiro*	*Tribeca*	$[1998\text{-}01\text{-}01, 1999\text{-}01\text{-}01)$	0.6

Wedding

	Subject	Object	Valid Time	p
I_3	*DeNiro*	*Abbott*	$[1936\text{-}11\text{-}01, 1936\text{-}12\text{-}01)$	0.3
I_4	*DeNiro*	*Abbott*	$[1976\text{-}07\text{-}29, 1976\text{-}07\text{-}30)$	0.8

Divorce

	Subject	Object	Valid Time	p
I_5	*DeNiro*	*Abbott*	$[1988\text{-}09\text{-}01, 1988\text{-}12\text{-}01)$	0.8

Figure 3.1: An exemplary Temporal Probabilistic Database

Example 3.1. *Our running example is centered around the actor "Robert De Niro" about whom the temporal probabilistic database of Figure 3.1 captures a number of facts. Tuple I_1 expresses that De Niro was born in Greenwich Village (New York) on August 17th, 1943, which is encoded into the time-interval* $[1943\text{-}08\text{-}17, 1943\text{-}08\text{-}18)$ *using the ISO standardized Internet date/time format. This tuple is* true *for the given time-interval with probability* 0.9*, and it is* false*, or does not exist, for this interval with probability* 0.1*. Furthermore, tuples are always* false *outside their attached time-intervals. Notice that another tuple, I_2, states that De Niro could have also been born in Tribeca in the interval* $[1998\text{-}01\text{-}01, 1999\text{-}01\text{-}01)$ *with probability* 0.6*. In the remainders of this chapter, we investigate how to evaluate queries over this kind of data, i.e. how to propagate time and probabilities from the database to the query answers. We also discuss consistency constraints. For instance the two tuples of* BornIn *state different birth places of De Niro, an inconsistency we should rule out by the use of constraints.*

3.2 Related Work

In computer science there is a long line of research on representing time, especially in artificial intelligence [51], logic [104, 92], scheduling [111], time series analysis [20], and temporal databases [71]. Additionally, there even exists a dedicated annual symposium, *TIME*[1], which accepts solely works on temporal concepts from some of the aforementioned fields. In this section, however, we focus on the facets relevant to this thesis, which are temporal databases, probabilistic databases and temporal constraints.

[1]`http://time.di.unimi.it/TIME_Home.html`

Temporal Datalog Research on temporal versions of Datalog mostly considers the temporal successor relation [23, 24]. That is, each relation is annotated by a time-point and the successor relation allows reasoning towards the future. Here, recursion induces theoretically interesting settings, where mostly conditions for finiteness are discussed. Recently, this idea was implemented in Dedalus [7] to model distributed systems. To the best of our knowledge, none of these works considers uncertainty.

Temporal Databases Managing temporal data in databases has been an active research field for many years, where Jensen's PhD thesis [71] still is a comprehensive reference. In general, there are two ways [5, 71] of modeling temporal annotations, either by time-points or by time-intervals. For the latter, Allen's interval algebra [6] is the most influential work, since it discusses all possible relationships between two intervals. Building on these data models, the most notable temporal query language is *TSQL2* [71, 134], which supports a variety of temporal features. It captures both *valid-time*, which describes when a tuple is true in the real world, and *transaction-time*, which captures when tuples were created or altered. Also, *coalescing* of consecutive temporal annotations and *time varying aggregates* are part of *TSQL2*. Likewise, *TSQL2* offers support for uncertainty about temporal annotations which is referred to as *temporal indeterminacy* [9, 42, 142]. The uncertainty in the tuples can be captured by inequalities or user-defined probability distributions. Over these tuples it is possible to run queries, which feature temporal arithmetic predicates such as comparing two uncertain temporal annotations. Still, we have to note that either their query results are deterministic or probabilities can only be multiplied, hence restricting the queries for which correct probabilities are computed to joins of independent relations. Finally, nowadays even the SQL 2011 standard contains some of *TSQL2*'s temporal properties [84].

Temporal Databases and Lineage Dignös et al. [37] describe how to enable the usage of sequenced semantics in temporal databases relying on lineage information. Intuitively, the sequenced semantics reduces a temporal operation to corresponding non-temporal operations over the individual snapshots of the database at each point in time. However, their approach does not support probabilistic data, and the non-sequenced semantics (see Section 3.5.4) we follow in this work is more expressive.

Temporal Probabilistic Databases There are relatively few works in the intersection of temporal databases and probabilistic databases. Dekhtyar et al. [35] introduce temporal probabilistic relations by adding time to a probabilistic database. Relations are defined via discrete probability distributions over time, and they also consider consistency constraints. However,

[35] does not provide a closed probabilistic database model since they do not support lineage information, and they approximate probabilities by coarse upper and lower bounds. Wang et al. [154] perform a simple form of probabilistic deduction of temporal annotations based on time histograms. They employ lineage but allow no constraints and their deduction capabilities are limited to overlapping temporal annotations. *LIVE* [126], which is an offspring of the *Trio* [14] probabilistic database system, employs a temporal database model by considering *transaction-time* annotations, thus focusing on efficiently supporting data modifications and versioning. In contrast, *valid-time* annotations, which we consider, refer to a time-interval during which a tuple is considered to be true in the real world. Research in uncertain spatio-temporal databases, such as [43], focuses on stochastically modeling trajectories through space and time. In contrast to our work, they do not employ concepts known from probabilistic databases, such as a possible-worlds model with data lineage. Finally, for time-series databases, the authors of [127] describe how to learn probability distributions by estimating densities, which is a very different setup compared to our relational database setup.

Temporal Constraints Imposing constraints on temporal data has a long history in research. Often constraints are formulated by Allen's interval relationships [6], which we support as well. Another common representation for temporal constraints are graphs [34]. The traditional way of solving the resulting problem are constraint satisfaction programs [11]. Recent works consider more sophisticated logical reasoning techniques, such as Satisfiability Modulo Theories (SMT) solving [75, 144]. With respect to different classes of temporal constraints, in [85] a study is presented characterizing the complexity of solving them. Also, scheduling problems [40, 111] encode temporal constraints and compute a solution to them. The common theme to all of the above approaches, however, is that they intend to compute only one possible solution for the given temporal constraints. Since we work on probabilities, we are counting all possible solutions, instead.

3.3 Contribution

This chapter presents a *unified temporal probabilistic database model* [39] in which both time and probability are considered as first-class citizens.

- Specifically, in Section 3.4.5 we define an expressive class of *temporal deduction rules*, before we study lineage tracing in the presence of both time and uncertain data in Section 3.4.6, and eventually in Section 3.4.8 devise how *temporal consistency constraints* can be incorporated.

- Moreover, we investigate the properties of the introduced data model from a theoretical perspective. In detail, we analyze the *complexity* of *data computations* (Section 3.5.1) as well as *probability computations* (Section 3.5.2). We show that our data model is *closed and complete* (Section 3.5.3), and we characterize its relationship to *sequenced semantics* (Section 3.5.4).

- Furthermore, in Section 3.6.2 we introduce an efficient *algorithm for temporal deduplication* of Section 3.4.6, which can be plugged into any probabilistic database system with lineage, hence enabling support of temporal data.

- Finally, in Section 3.7 we *experimentally evaluate* our data model and algorithms on two datasets from the domains of temporal information extraction and temporal knowledge bases. On these, we compete with state-of-the-art methods from probabilistic databases, statistical relation learning, and constraint-solving via integer linear programs.

3.4 Temporal Data Model

In this section we present our data model that supports queries and constraints over temporal and probabilistic data.

3.4.1 Time Domain

We start with the most important point, namely our model of time. As in a calendar, there are a number of choices to take. First, we have to decide on the granularity of time, which could be days, hours or minutes, for instance. Also, we should determine whether time is finite, and, if so, when it starts or ends, e.g. at the first or last day of a year, respectively.

Technically, we adopt the view of time as points which then can be coalesced to form intervals [71, 148]. We consider the *time universe* $\mathcal{U}^T$ as a linearly ordered finite sequence of *time points*, e.g. days, minutes or even milliseconds. Considering time to be finite and discrete later ensures that there are finitely many possible worlds. 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. For instance, a day can be viewed as an interval of hours. Moreover, we employ the two constants t_{min}, t_{max} to denote the earliest and latest time point in $\mathcal{U}^T$, respectively. Finally, temporal variables are written as T or $[T_b, T_e)$, if we refer to a time-interval.

We do not consider the finiteness of $\mathcal{U}^T$ to be any limitation of our model in practice, since we can always choose t_{min}, t_{max} as the earliest and latest time points we observe among the tuples and deduction rules. Also, discrete time-points of fixed granularity do not present any restraint, as we can resort to employing time-points of smaller granularity than observed in the data.

Example 3.2. *Regarding the database of Figure 3.1, $\mathcal{U}^T$ comprises the sequence of days starting at $t_{min} :=$ 1936-11-01 and ending at $t_{max} :=$ 1999-01-01. We could equally choose any more fine-grained unit for the time-points, but for presentation purposes we select days.*

3.4.2 Relations and Tuples

We now relate data to time, that is, tuples are considered valid during a specific time only and otherwise invalid. For this, we extend the relations introduced in Section 2.1.1 to temporal relations, following work by [1, 71, 145]. We annotate each tuple by a time-interval specifying the validity of the tuple over time, a technique, which is commonly referred to as *tuple timestamping* [71]. More detailed, a *temporal relation R^T* is a logical predicate of arity $r \geq 3$, whose latter two arguments are temporal. Hence, an instance of a temporal relation is a finite subset $\mathcal{R}^T \subseteq \mathcal{U}^{r-2} \times \mathcal{U}^T \times \mathcal{U}^T$. Therein, we require the temporal arguments t_b, t_e of a tuple $R^T(\bar{a}, t_b, t_e)$ to form a time-interval $[t_b, t_e)$. Choosing intervals over time-points has the advantage that the storage costs are independent of the granularity of the time-points.

Example 3.3. *The tuple* BornIn(DeNiro, Greenwich, 1943-08-17, 1943-08-18) *of Figure 3.1 is valid only at one day, namely on August 17th 1943.*

In general, a temporal relation instance can contain several tuples with equivalent non-temporal arguments $\bar{a}$, but with varying temporal arguments. For instance, assume we have two tuples describing De Niro's birthday, one timestamped with the year 1943 and one by the day 1943-08-18. Then, a database engine might conclude that he was born twice on August 18th, 1943 with different probabilities. To resolve this issue, we enforce the time-intervals of tuples with identical non-temporal arguments to be disjoint, where a relation instance obeying this is termed *duplicate-free* [37].

Definition 3.1. *A temporal relation instance $\mathcal{R}^T$ is called* duplicate-free*, if for all pairs of tuples $R^T(\bar{a}, t_b, t_e), R^T(\bar{a}', t'_b, t'_e) \in \mathcal{R}^T$ it holds that:*

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

The above definition does not affect tuples of different non-temporal arguments or with non-overlapping temporal arguments.

Example 3.4. *In Figure 3.1 the temporal relation instance* Wedding *is duplicate-free, as both tuples have equivalent non-temporal arguments, but their intervals are non-overlapping.*

3.4.3 Temporal Probabilistic Databases

In this section, we extend tuple-independent probabilistic databases of Definition 2.10 to temporal data as in [39]. Intuitively, each tuple has two annotations: a temporal and a probabilistic one. Hence, each tuple exists only during a given time and with a given probability. Supporting both probability and time annotations allows to represent data where we are unsure about the time when it is valid.

Definition 3.2. *For temporal relations $R_1^T, \dots, R_n^T$ a* tuple-independent temporal probabilistic database *$(\mathcal{T}, p, \mathcal{U}^T)$ is a triple, where*

1. *$\mathcal{T} := \mathcal{R}_1^T \cup \dots \cup \mathcal{R}_n^T$ is a finite set of tuples;*
2. *$\forall i \in \{1, \dots, n\} : \mathcal{R}_i^T$ is duplicate-free;*
3. *all tuples $R_i^T(\bar{a}, t_b, t_e)$ of all relation instances $\mathcal{R}_i^T$ share the time universe $\mathcal{U}^T$, that is, $t_b, t_e \in \mathcal{U}^T$;*
4. *p is a function $p : \mathcal{T} \to (0, 1]$ which assigns a non-zero* probability value *$p(I)$ to each tuple $I \in \mathcal{T}$;*
5. *the probability values of all tuples in $\mathcal{T}$ are assumed to be* independent*.*

Above, the first, fourth and fifth condition are analogous to Definition 2.10. Still, we here consider *temporal* relation instances $\mathcal{R}_i^T$ and require them to be duplicate-free (see Definition 3.1). Additionally, all time-points occurring in any relation instance $\mathcal{R}_i^T$ must be contained in the time universe $\mathcal{U}^T$. We highlight that the probabilities of two tuples $R(\bar{a}, t_b, t_e)$ and $R(\bar{a}, t'_b, t'_e)$, even if they share $\bar{a}$, are independent due to the fifth condition. In the remaining parts of the thesis we will often drop the term tuple-independent when we refer to a temporal probabilistic database. As in Chapter 2, correlations among tuples will be induced by constraints and queries.

Example 3.5. *The temporal relation instances of Figure 3.1, with their time universe defined in Example 3.2 form the temporal probabilistic database $(\{I_1, I_2, I_3, I_4, I_5\}, p, \langle 1936\text{-}11\text{-}01, \dots, 1999\text{-}01\text{-}01\rangle)$.*

Since tuple probabilities here are defined as in probabilistic databases, and $\mathcal{U}^T$ is finite as well as discrete, the possible worlds of Section 2.2.1 apply to temporal probabilistic databases as well. Now, we characterize the relationship between temporal probabilistic databases and (non-temporal) probabilistic databases.

Proposition 3.1. *Every probabilistic database instance $(\mathcal{T}, p)$ can be encoded in a temporal probabilistic database instance $(\mathcal{T}', p, \mathcal{U}^T)$.*

Proof. To achieve the encoding we create the time universe $\mathcal{U}^T := \langle 1, 2\rangle$, expand each relation R in $\mathcal{T}$ by two temporal arguments, and set $\mathcal{T}' := \{R^T(\bar{a}, 1, 2) \mid R(\bar{a}) \in \mathcal{T}\}$. □

3.4.4 Arithmetic Predicates

To express temporal statements, it is necessary to be able to compare temporal annotations in form of time-points. Hence, we support two temporal arithmetic predicates $=^T$ and $<^T$ [51, 104], which check for the equality and precedence of time-points, respectively.

Definition 3.3. *For* $t_1, t_2 \in \mathcal{U}^T$ *the* temporal arithmetic predicates $=^T$ *and* $<^T$ *are evaluated as follows:*

$$t_1 =^T t_2 \equiv \begin{cases} \text{true} & \textit{if } t_1 = t_2, \\ \text{false} & \textit{otherwise}, \end{cases}$$
$$t_1 <^T t_2 \equiv \begin{cases} \text{true} & \textit{if } t_1 \textit{ strictly before } t_2 \textit{ in } \mathcal{U}^T, \\ \text{false} & \textit{otherwise.} \end{cases}$$

In other words, $=^T$ is satisfied, whenever two time-points are identical, whereas $<^T$ compares the order of two time-points in $\mathcal{U}^T$.

Example 3.6. *Since* 1998-01-01 *is before* 1999-01-01*, we have* 1998-01-01 $<^T$ 1999-01-01 $\equiv$ true.

By utilizing conjunctions of $<^T$ and $=^T$ predicates over the temporal arguments, we are able to express all the 13 relationships between time-intervals defined in the seminal work of Allen [6], such as *overlaps*, *disjoint* or *starts*.

Proposition 3.2. *We can express all 13 relationships between two time intervals as defined by Allen [6] by relying on conjunctions of* $=^T$ *and* $<^T$*.*

Proof.

Allen's relation	Our encoding
$[T_b, T_e)$ before $[T'_b, T'_e)$	$T_e <^T T'_b$
$[T_b, T_e)$ equal $[T'_b, T'_e)$	$T_b =^T T'_b \wedge T_e =^T T'_e$
$[T_b, T_e)$ meets $[T'_b, T'_e)$	$T_e =^T T'_b$
$[T_b, T_e)$ overlaps $[T'_b, T'_e)$	$T_b <^T T'_b \wedge T'_b <^T T_e \wedge T_e <^T T'_e$
$[T_b, T_e)$ during $[T'_b, T'_e)$	$T'_b <^T T_b \wedge T_e <^T T'_e$
$[T_b, T_e)$ starts $[T'_b, T'_e)$	$T_b =^T T'_b \wedge T_e <^T T'_e$
$[T_b, T_e)$ finishes $[T'_b, T'_e)$	$T'_b <^T T_b \wedge T_e =^T T'_e$

The remaining 6 relationships are the inverse of one of the above ones, except for equality which is symmetric. □

3.4.5 Deduction Rules

Next, we devise temporal deduction rules, that is, general "if-then" rules which mention time. Formally, our temporal deduction rules [39] are logical implications over temporal relations and temporal arithmetic predicates, defined as follows.

Definition 3.4. *A* temporal deduction rule *is a logical rule of the form*

$$R_0^T(\bar{X}_0, T_b, T_e) \leftarrow \bigwedge_{i=1,\dots,n} R_i^T(\bar{X}_i, T_{i,b}, T_{i,e}) \ \wedge \ \bigwedge_{j=1,\dots,m} \neg R_j^T(\bar{X}_j, T_{j,b}, T_{j,e}) \ \wedge \Phi(\bar{X}, \bar{T}) \tag{3.1}$$

where

1. *all requirements of Definition 2.2 hold;*
2. T_b*,* T_e*,* $T_{i,b}$*,* $T_{i,e}$*,* $T_{j,b}$*,* $T_{j,e}$ *and* $\bar{T}$ *are temporal constants and variables, where* $Var(T_b, T_e)$*,* $Var(T_{j,b}, T_{j,e})$*,* $Var(\bar{T}) \subseteq \bigcup_i Var(T_{i,b}, T_{i,e})$*;*
3. $\Phi(\bar{X}, \bar{T})$ *is a conjunction of literals over the arithmetic predicates, such as* $=$ *and* $\neq$*, and the temporal arithmetic predicates* $=^T$ *and* $\leq^T$*.*

With respect to non-temporal arguments all restrictions of non-temporal deduction rules (see Definition 2.2) hold. Combining this fact with the second requirement above, we conclude that temporal deduction rules are *safe* [2]. Furthermore, the third condition allows the temporal arithmetic predicates of Definition 3.3 to occur in temporal deduction rules. Of course, also non-temporal relations are allowed in temporal deduction rules, hence inducing mixtures of temporal and non-temporal rules. We note that the above class of temporal deduction rules is very expressive, as it allows T_b, T_e to be constants or to be variables from different literals R_i^T. In particular, the rules do not obey the sequenced semantics (see Section 3.5.4). Finally, as in Chapter 2 we assume the rules to be *non-recursive*.

Example 3.7. *Given the tuples of Figure 3.1 about both De Niro's wedding and divorce with Dianne Abbott, we aim to deduce the time-interval of their marriage by temporal deduction rules. The first rule states that a couple stays married from the begin time point of their wedding (denoted by the variable* $T_{b,1}$*) until to the last possible time point we consider (denoted by the constant* t_{max}*), unless there is a divorce tuple.*

$$Marriage^T(P_1, P_2, T_{b,1}, t_{max}) \leftarrow \begin{pmatrix} Wedding^T(P_1, P_2, T_{b,1}, T_{e,1}) \wedge \\ \neg Divorce(P_1, P_2) \end{pmatrix} \tag{3.2}$$

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

$$Divorce(P_1, P_2) \leftarrow Divorce^T(P_1, P_2, T_b, T_e)$$

The second rule states that a couple stays married from the begin time point of their wedding till the end time point of their divorce.

$$Marriage^T(P_1, P_2, T_{b,1}, T_{e,2}) \leftarrow \begin{pmatrix} Wedding^T(P_1, P_2, T_{b,1}, T_{e,1}) \wedge \\ Divorce^T(P_1, P_2, T_{b,2}, T_{e,2}) \wedge \\ T_{e,1} <^T T_{b,2} \end{pmatrix} \tag{3.3}$$

Thereby, we consider only weddings that took place before divorces as stated by $T_{e,1} <^T T_{b,2}$*.*

3.4.6 Lineage and Deduplication

As in Section 2.2.4, we trace the deduction history of tuples via lineage, however, with the additional twist that it may vary over time. Since temporal deduction rules are safe, the groundings $G(D, \mathcal{T})$ of Definition 2.4 and the new tuples $\mathit{NewTuples}(D, \mathcal{T})$ of Definition 2.5 apply to temporal deduction rules as well. Hence, at first glance lineage tracing according to Definition 2.11 works in a temporal context, but with one random variable for each tuple with its time-interval. However, if we execute temporal deduction rules, the newly derived tuples may not necessarily define a duplicate-free relation instance. We illustrate this issue by the following example.

Example 3.8. *Let the deduction rules of Example 3.7 and the tuples of Figure 3.1 be given. Now, in Figure 3.2 we visualize both the tuples from database (on the bottom) and the deduced tuples (in the middle). Inspecting*

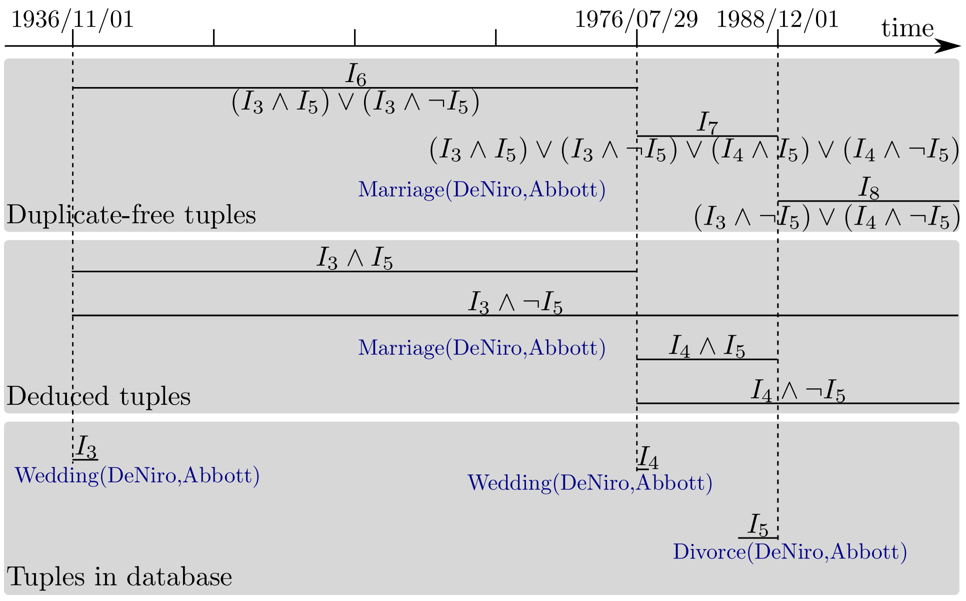


Figure 3.2: Deducing & deduplicating tuples with time-intervals

the deduced tuples, we realize that they have equivalent non-temporal arguments, i.e. DeNiro *and* Abbott, *but their time-intervals are overlapping, which contradicts Definition 3.1 of duplicate-free relation instances.*

Hence, in order to convert a temporal relation instance with duplicates (as shown in the middle box of Figure 3.2) into a duplicate-free temporal relation (as shown on the top of Figure 3.2), we provide the following definition.

Definition 3.5. *Let a temporal relation R^T, 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 $R^T(\bar{a}, t_b, t_e)$ that are valid at time point t:*

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

We create duplicate free tuples $I' = R^T(\bar{a}, t_b, t_e)$ such that for any pair of time points $t_0, t_1 \in [t_b, t_e)$ it holds that:

$$L(R^T, \bar{a}, t_0, \mathcal{T}) = L(R^T, \bar{a}, t_1, \mathcal{T}) \tag{3.4}$$

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

$$\lambda(I') := \bigvee_{\phi_i \in L(R^T, \bar{a}, t_b, \mathcal{T})} \phi_i \tag{3.5}$$

In short, for each time-point t we create the disjunction of all tuples being valid at t (see Equation 3.5). 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 (3.4)). We remark that for the equality of Equation (3.4), we focus on syntactical equivalence checks between lineage formulas. Nevertheless, we refrain from logical equivalence checks, as they are *co-$\mathcal{NP}$*-complete [25].

Example 3.9. *Applying Definition 3.5 to the tuples in the middle of Figure 3.2 yields the tuples shown at the top of the figure. For instance, if we inspect L at the time points* 1976-07-28 *and* 1976-07-29*, we notice that $\{I_3 \wedge I_5, I_3 \wedge \neg I_5\} \neq \{I_3 \wedge I_5, I_3 \wedge \neg I_5, I_4 \wedge I_5, I_4 \wedge \neg I_5\}$, so two differing tuples I_6 and I_7 have to be kept in the relation. In total, the resulting duplicate-free tuples are:*

Marriage

	Subject	*Object*	*Valid Time*
I_6	*DeNiro*	*Abbott*	[1936-11-01, 1976-07-29)
I_7	*DeNiro*	*Abbott*	[1976-07-29, 1988-12-01)
I_8	*DeNiro*	*Abbott*	[1988-12-01, t_{max})

Following Equation (3.5)*, their lineages are:*

$$\begin{aligned}
\lambda(I_6) &= (I_3 \wedge I_5) \vee (I_3 \wedge \neg I_5) \\
\lambda(I_7) &= (I_3 \wedge I_5) \vee (I_3 \wedge \neg I_5) \vee (I_4 \wedge I_5) \vee (I_4 \wedge \neg I_5) \\
\lambda(I_8) &= (I_3 \wedge \neg I_5) \vee (I_4 \wedge \neg I_5)
\end{aligned}$$

Hence, for temporal deduction rules the combination of Definitions 2.11 and 3.5 creates the lineage formulas. We want to remark that these lineage formulas are purely propositional formulas as captured below.

Observation 3.1. *Temporal deduction rules and deduplication produce propositional lineage formula without any explicit mention of time.*

Hence, any work on probabilistic databases with lineage can be applied, especially probability computations (see Section 2.2.5).

3.4.7 Queries and Answers

As a last step we introduce temporal queries which extend Definition 2.6 by a temporal component. A query resembles the body of a temporal deduction rule.

Definition 3.6. *Given temporal deduction rules $\mathcal{D}$ with their intensional relations a* temporal query *Q is a conjunction:*

$$Q(\bar{X}_0, \bar{T}_0) := \bigwedge_{i=1,\ldots,n} R_i^T(\bar{X}_i, T_{i,b}, T_{i,e}) \wedge \bigwedge_{j=1,\ldots,m} \neg R_j^T(\bar{X}_j, T_{j,b}, T_{j,e}) \wedge \Phi(\bar{X}, \bar{T})$$

where

1. *all requirements of Definition 2.6 hold;*
2. *$\bar{T}_0$, $T_{i,b}$, $T_{i,e}$, $T_{j,b}$, $T_{j,e}$ and $\bar{T}$ are temporal constants and variables, which satisfy:*
   (a) *$Var(\bar{T}_0) = \bigcup_{i=1,\ldots,n} Var(T_{i,b}, T_{i,e})$;*
   (b) *$\forall j \in \{1, \ldots, m\}$ $Var(T_{j,b}, T_{j,e}) \subseteq \bigcup_{i=1,\ldots,n} Var(T_{i,b}, T_{i,e})$;*
   (c) *$Var(\bar{T}) \subseteq \bigcup_{i=1,\ldots,n} Var(T_{i,b}, T_{i,e})$;*
3. *$Var(\bar{X}_0) \cup Var(\bar{T}_0)$ are the* query variables*;*
4. *$\Phi(\bar{X}, \bar{T})$ is a conjunction of (temporal) arithmetic literals.*

Temporal queries inherit all properties from non-temporal queries (see Definition 2.6), in particular that all relations are intensional. The first and second condition above ensure *safe* [2] queries. In this chapter, the query variables are formed by both the variables in $\bar{X}_0$ and in $\bar{T}_0$. With respect to arithmetic predicates we support non-temporal ones as in Section 2.1.4 and the temporal ones from Definition 3.3.

Example 3.10. *If we are interested in people who were married before* 1980*, we write the query*

$$Marriage(P_1, P_2, T_b, T_e) \ \wedge \ T_b <^T \text{1980-01-01}$$

where the intensional relation Marriage *is defined in Example 3.7.*

Since the restrictions on variables of Definition 2.6 and Definition 3.6 coincide, query answers can be obtained as in the non-temporal case of Definition 2.7.

3.4.8 Constraints

In Section 2.2.6 we introduced constraints as propositional lineage formula ϕ_c. Following Definition 2.14, we can create constraints by deduction rules. For this, we keep two sets of literals $\mathcal{C}_p$ and $\mathcal{C}_n$ which indicate constraints that always hold and never hold, respectively. Then, the literals of both sets induce the lineage formula ϕ_c (see Definition 2.14). Hence, in this chapter constraints are formulated as temporal deduction rules. As we support temporal arithmetic predicates (see Definition 3.3) in the temporal deduction rules, we can express any temporal precedence (ordering) constraint, and any temporal disjointness or containment constraint.

Example 3.11. *If we intend to enforce that persons are born before their marriage starts, we write*

$$Constraint(P_1, P_2, T_b, T_e, T'_b, T'_e) \leftarrow \left(\begin{array}{r} Born^T(P_1, T_b, T_e) \wedge \\ Marriage^T(P_1, P_2, T'_b, T'_e) \wedge \end{array} T'_b <^T T_e \right) \tag{3.6}$$

and add $\text{Constraint}(P_1, P_2, T_b, T_e, T'_b, T'_e)$ *to* $\mathcal{C}_n$. *To shorten notation we also write the above constraint as:*

$$\neg(Born^T(P_1, T_b, T_e) \wedge Marriage^T(P_1, P_2, T'_b, T'_e) \wedge T'_b <^T T_e)$$

Here, the negation resembles that the head literal of Equation (3.6) *is in* $\mathcal{C}_n$, *i.e. it should never hold. When we ground the above constraint, all pairs of* Marriage^T *and* Born^T *tuples contradicting the correct temporal ordering are excluded by* ϕ_c.

3.5 Theoretical Properties

In this section, we describe the properties of our temporal probabilistic database model [39] with respect to grounding, probability computations, and its representational power.

3.5.1 Grounding Complexity

We first investigate the computational complexity of grounding, that is instantiating all deduction rules and the query against a database to obtain answers with their lineage formulas.

Theorem 3.1. *For a temporal probabilistic database* $(\mathcal{T}, p, \mathcal{U}^T)$, *a fixed set of temporal deduction rules* $\mathcal{D}$ *and a fixed query* Q, *grounding has polynomial data complexity in* $|\mathcal{T}|$ *and* $|\mathcal{U}^T|$.

Because both queries and constraints are represented by deduction rules, the above theorem applies to both of them individually, which in combination yields polynomial complexity again.

Proof. Due to Lemma 2.1 grounding deduction rules with lineage tracing has polynomial data complexity. We proceed by a reduction from temporal deduction rules to non-temporal Datalog with inequalities. The latter has polynomial data complexity due to Lemma 2.1, which hence implies polynomial data complexity for the temporal case. During the proof we will deploy several indices at relation symbols R, hence, to avoid cluttering we drop ^{T} in R^T throughout the proof.

First, let $\mathcal{U}^T$ be isomorphic to a finite subset of the natural numbers. For each temporal (and probabilistic) relation $R(\bar{X}, T_b, T_e)$, we introduce an additional deterministic relation $R^d(\bar{X}, T_b, T_e)$. For extensional relations, we copy all tuples into the new relation, and we set the probabilities $\forall I \in \mathcal{R}^d : P(I) = 1.0$. We then rewrite a general rule of Equation (3.1) as follows. First, we create a deterministic version of the rule:

$$R^d(\bar{X}_0, T_b, T_e) \leftarrow \bigwedge_i R^d_i(\bar{X}_i, T_{b,i}, T_{e,i}) \wedge \bigwedge_j \neg R^d_j(\bar{X}_j, T_{b,j}, T_{e,j}) \wedge \Phi(\bar{X}, \bar{T})$$

Applying this rule yields all tuples as deterministic tuples, which are not necessarily duplicate-free. To counter this we follow Definition 3.5 using Datalog only. Therefore, we gather all limits of time-intervals by the following two rules over the deterministic relations:

$$\begin{array}{l} \mathit{Limit}^d_R(\bar{X}, T_b) \leftarrow R^d(\bar{X}, T_b, T_e) \\ \mathit{Limit}^d_R(\bar{X}, T_e) \leftarrow R^d(\bar{X}, T_b, T_e) \end{array}$$

To ensure that the intervals are non-overlapping, we create another deterministic relation which is instantiated for each pair of time points (denoted by the variables T_1, T_3), if there is at least one *Limit* tuple in between:

$$\mathit{Between}^d_R(\bar{X}, T_1, T_3) \leftarrow \begin{array}{l} \mathit{Limit}^d_R(\bar{X}, T_1) \wedge \mathit{Limit}^d_R(\bar{X}, T_2) \wedge \\ \mathit{Limit}^d_R(\bar{X}, T_3) \wedge T_1 < T_2 \wedge T_2 < T_3 \end{array}$$

Now, we deduce all adjacent, non-overlapping time-intervals (i.e., shown in the upper part of Figure 3.2) by the following rule:

$$\mathit{Interval}^d_R(\bar{X}, T_b, T_e) \leftarrow \begin{array}{l} \mathit{Limit}^d_R(\bar{X}, T_b) \wedge \mathit{Limit}^d_R(\bar{X}, T_e) \\ \wedge T_b < T_e \wedge \neg \mathit{Between}^d_R(\bar{X}, T_b, T_e) \end{array}$$

These intervals are duplicate-free as demanded by Definition 3.1. At this point, we are able to deduce the time-intervals as deterministic tuples, however, we are missing rules to induce the correct lineage, i.e. of Equation (3.5), in order to compute probabilities. Thus, in the final step, we create another copy of Equation (3.1) where R' is a new relation:

$$R'(\bar{X}_0, T_b, T_e) \leftarrow \bigwedge_i R_i(\bar{X}_i, T_{b,i}, T_{e,i}) \wedge \bigwedge_j \neg R_j(\bar{X}_j, T_{b,j}, T_{e,j}) \wedge \Phi(\bar{X}, \bar{T})$$

Again, its deduced time-intervals correspond to the ones on middle of Figure 3.2. We add a final deduction rule which combines the deduced tuples with their correct time-intervals from $\mathit{Interval}^d_R$ and creates the logical disjunction in the lineage as required by Definition 3.5, thus utilizing the uncertain tuples of R':

$$R(\bar{X}, T'_b, T'_e) \leftarrow R'(\bar{X}, T_b, T_e) \wedge \mathit{Interval}^d_R(\bar{X}, T'_b, T'_e) \wedge T_b \leq T'_b \wedge T'_e \leq T_e$$

We note that the time-interval $[T'_b, T'_e)$ of the head originates from the literal $\mathit{Interval}^d_R(\bar{X}, T'_b, T'_e)$, hence the tuples deduced from R' are duplicate-free. Also, all tuples in $R'(\bar{X}, T_b, T_e)$, whose time-interval contains $[T'_b, T'_e)$, deduce $R(\bar{X}, T'_b, T'_e)$, so the logical disjunction of Equation (3.5) is produced. Furthermore, since $\mathit{Interval}^d_R$ is deterministic, the probability of the deduced tuple entirely depends on R'.

If we apply the above procedure to all temporal deduction rules, we obtain a polynomial blow-up in the size of the data and the number of rules. Since the data complexity of non-recursive Datalog with inequalities and lineage tracing is polynomial (see Lemma 2.1), also our model has polynomial data complexity. $\square$

3.5.2 Probability Computations Complexity

We next study hard cases of computing probabilities which can result from temporal deduction rules of a query over a temporal probabilistic database.

Lemma 3.1. *For a temporal probabilistic database $(\mathcal{T}, p, \mathcal{U}^T)$ there are temporal deduction rules $\mathcal{D}$ for which probability computations are $\#\mathcal{P}$-hard in $|\mathcal{T}|$ and $|\mathcal{U}^T|$.*

Proof. Consider a temporal probabilistic database $(\mathcal{T}, p, \mathcal{U}^T)$ and the deduction rule

$$R^T(\bar{X}, T'_b, T'_e) \leftarrow R^T_1(\bar{X}, T_b, T'_e) \wedge R^T_2(\bar{X}, T_b, T_e) \wedge R^T_3(\bar{X}, T'_b, T_e)$$

which joins on the temporal variables T_b and T_e. Every answer $R(\bar{a}, t_b, t_e)$ has $\#\mathcal{P}$-hard probability computations in $|\mathcal{T}|$ and $|\mathcal{U}^T|$, as it resembles $H(1)$ of Lemma 2.2. $\square$

Of course, there are other ways of causing lineage formulas with hard probability computations, such as joins on non-temporal variables. Besides the deduction rules of a query, also the temporal constraints of Section 3.4.8 can cause expensive probability computations when evaluating Equation (2.7).

3.5.3 Closure and Completeness

Generally, a representation system is *complete* [60], if it can represent any finite instance of data, which is in our case temporal and probabilistic. Furthermore, a representations system is *closed* [60], if all query results can be expressed in the representation itself.

Theorem 3.2. *A temporal probabilistic database $(\mathcal{T}, p, \mathcal{U}^T)$ with lineage is* closed *and* complete *under all algebraic operations which are expressible by the temporal deduction rules $\mathcal{D}$.*

Because completeness implies closure, we provide a proof for the completeness of our temporal probabilistic database model.

Proof. We show that, given any finite instance $\mathcal{T}$ of temporal and probabilistic relational data, we can represent it in our temporal probabilistic database model. Without loss of generality, we are given only one relation instance $\mathcal{R}^T$ along with its possible worlds $\mathcal{W}_1, \ldots, \mathcal{W}_n$ and a probability $P(\mathcal{W}_i)$ for each of them. Now, to encode these in a temporal probabilistic database $(\mathcal{T}, p, \mathcal{U}^T)$, there are three points to show, namely (1) setting $\mathcal{U}^T$, (2) ensuring that $\mathcal{R}^T$ is duplicate free, and (3) determining $\mathcal{T}$ and p.

First, we select the earliest and latest time-points t_{min} and t_{max}, respectively, which occur in $\mathcal{R}^T$. From this, we create the sequence $\mathcal{U}^T := \langle t_{min}, \ldots, t_{max} \rangle$ where each time-point is of the smallest granularity of time-points that occurs in $\mathcal{R}^T$. Second, to guarantee that each $\mathcal{R}^T$ is duplicate-free (see Definition 3.1), we create a new relation instance $\mathcal{R}^{T'}$ which extends each tuple by a unique id, e.g. if $R^T(\bar{a}, t_b, t_e) \in \mathcal{R}^T$, then $R^{T'}(id, \bar{a}, t_b, t_e) \in \mathcal{R}^{T'}$. Third, regarding the probabilistic data, we follow [14, 139] by proving the statement via induction over the number of possible worlds. Let the possible worlds $\mathcal{W}_i$ range over $\mathcal{R}^{T'}$.

<u>Basis $n = 1$:</u> In this case, there is only one possible world $\mathcal{W}_1$ with $P(\mathcal{W}_1)$. We store $\mathcal{W}_1$ in the deterministic relation $R_1^{T',d}$ and create a uncertain relation $R^u(X)$ holding one tuple $R^u(1)$ with $p(R^u(1)) = 1$. Then, the rule

$$R_1^{T'}(\bar{X}) \leftarrow R_1^{T',d}(\bar{X}) \wedge R^u(1)$$

along with $\mathcal{T}_1 := \mathcal{W}_1$ encodes the temporal probabilistic database. Now, queries posed on $R_1^{T'}$ deliver the correct semantics.

<u>Step $n \to n + 1$:</u> We want to extend the temporal probabilistic database by a possible world $\mathcal{W}_{n+1}$ which should have $P(\mathcal{W}_{n+1}) = p_{n+1}$. For this, we create the deterministic relation $R_{n+1}^{T',d}$ containing the tuples of $\mathcal{W}_{n+1}$. Then, we insert the tuple $R^u(n + 1)$ into R^u and set its probability value to p_{n+1}. Now, we add the rules:

$$\begin{aligned} R_{n+1}^{T'}(\bar{X}) &\leftarrow R_{n+1}^{T',d}(\bar{X}) \wedge R^u(n + 1) \\ R_{n+1}^{T'}(\bar{X}) &\leftarrow R_n^{T'}(\bar{X}) \wedge \neg R^u(n + 1) \end{aligned}$$

Next, we set $\mathcal{T}_{n+1} := \mathcal{T}_n \cup \mathcal{W}_{n+1}$ to finalize the temporal probabilistic database. Again, queries formulated on $R^{T'}_{n+1}$ yield the intended semantics. □

3.5.4 Relationship to Sequenced Semantics

In temporal databases there is a distinction between sequenced and non-sequenced semantics [37, 71]. Intuitively, the sequenced semantics reduces a temporal database operation to corresponding non-temporal operations over the individual snapshots of the database at each point in time (see Figure 3.3). In general, sequenced semantics are easier to grasp, but less

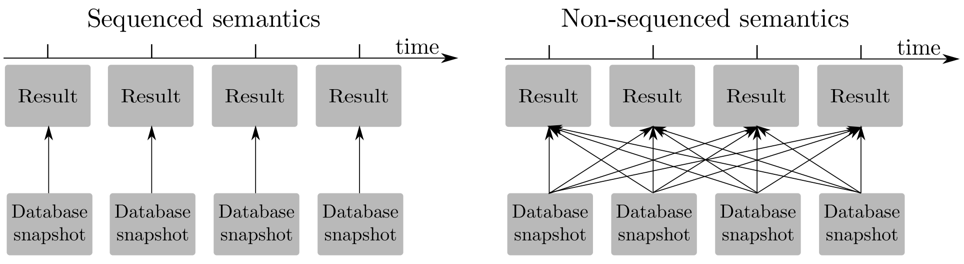


Figure 3.3: Sequenced and non-sequenced semantics

expressive than non-sequenced semantics, which allow operations involving database snapshots at more than one time point. Now, we discuss the relationship between sequenced semantics and our temporal probabilistic data model.

Corollary 3.1. *Deduplication can be expressed by operations obeying the sequenced semantics.*

Proof. The crucial part of Definition 3.5 is the set $L(R^T, \bar{a}, t, \mathcal{T})$. It captures lineages of tuples with relation R^T and non-temporal arguments $\bar{a}$ that are valid at time-point t. Since L can be computed on a per-time point base, it is expressible in sequenced semantics. □

Observation 3.2. *In general, our temporal deduction are not in sequenced semantics.*

The temporal deduction rules of Example 3.7 are both not executable under sequenced semantics, since their resulting tuples cover time points lying in between the time-intervals of the wedding and divorce tuples. One possible subset of temporal deduction rules that adheres the sequenced semantics is considering only deduction from overlapping time-intervals, as discussed for example in [154].

3.5.5 Temporal Coalescing

Temporal coalescing [15] is the process of merging tuples with identical non-temporal arguments and adjacent or overlapping time-intervals. For example, when storing the tuples of Figure 3.2 in a temporal database without probabilities, we could coalesce all marriage tuples into a single time-interval, hence reducing redundancy. Unfortunately, this is not always possible in our data model. The reason is that deduplication (see Definition 3.5) disallows the coalescing of intervals with non-equivalent lineage, as the difference in lineage can result in varying probabilities. Furthermore, checking whether two propositional lineage formulas ϕ and ψ are equivalent $\models \phi \leftrightarrow \psi$ is $co\text{-}\mathcal{NP}$-complete.

3.6 Algorithms

In this section we first discuss a transformation of propositional lineage formulas called decomposition (Section 3.6.1) which accelerates probability computations before we introduce an algorithmic solution to deduplication (Section 3.6.2).

3.6.1 Lineage Decomposition

In practice, the necessity for Shannon expansions (see Section 2.2.5) hinders the scalability of the probability computation step to larger datasets. Hence, we discuss two methods for lineage decomposition (executed before invoking the probability computations), which reduce the number of Shannon expansions [39, 47]. To quantify these, we introduce the following function.

Definition 3.7. *Let $\phi = \psi_1$ op $\ldots$ op ψ_n be a propositional lineage formula with operator op $\in \{\wedge, \vee\}$, and let ψ_0 to ψ_n be subformulas of ϕ. Then, we let the function Shannon return the number of Shannon expansions necessary at the top-level operator:*

$$\mathit{Shannon}(\phi) = |\{I \mid \exists i \neq j : I \in \mathit{Tup}(\psi_i), I \in \mathit{Tup}(\psi_j)\}|$$

The above definition applies to $\wedge, \vee$ only, because $\neg$ cannot induce new Shannon expansions. Moreover, if $\mathit{Shannon}(\phi) = 0$, then the second or third line of Equation (2.4) is applicable. When we attempt to calculate $P(\phi)$ naively, $\mathit{Shannon}(\phi)$ characterizes the exponent added to the runtime of $P(\phi)$ by the top level operator of ϕ.

Example 3.12. *As an example, we apply S to the propositional lineage formula $\phi := (I_1 \wedge I_2) \vee (I_2 \wedge \neg I_3) \vee (I_3 \wedge \neg I_1)$. That is $op = \vee$ and $\mathit{Shannon}(\phi) = |\{I_1, I_2, I_3\}| = 3$. Thus, a naive probability computation of $P(\phi)$ would take 2^3 iterations.*

Next, we characterize sets of propositional lineage formulas, which do not share any tuple, or in other words, are disjoint with respect to tuples. Connecting formulas from different sets by logical connectives hence does not induce any Shannon expansion.

Definition 3.8. *Given a set of propositional lineage formulas* $\{\phi_1, \dots, \phi_n\}$

$$\begin{array}{ll} \textit{Determine:} & S_1, \dots, S_m \subseteq \{\phi_1, \dots, \phi_n\} \\ \textit{Such that:} & \dot{\bigcup}_i S_i = \{\phi_1, \dots, \phi_n\} \\ \textit{and} & \forall i \neq j : \mathit{Tup}(S_i) \cap \mathit{Tup}(S_j) = \emptyset \end{array}$$

The first condition ensures that all formulas are contained in on of the disjoint sets S_i. The second condition disallows a tuple identifier to occur in more the one set. Moreover, the above problem can be solved in (almost) linear time by the union-find datastructure [25].

Example 3.13. *We consider the following propositional lineage formulas* $\{I_1 \wedge I_2, \neg I_2 \wedge I_3, I_4 \wedge I_5, I_4 \wedge \neg I_5 \wedge I_6\}$*. Then, there are two independent sets of lineage formulas, namely* $S_1 = \{I_1 \wedge I_2, \neg I_2 \wedge I_3\}$ *and* $S_2 = \{I_4 \wedge I_5, I_4 \wedge \neg I_5 \wedge I_6\}$*.*

Building on independent sets of formulas, we rewrite a lineage formula using associativity. This transformation reduces the necessary number of Shannon expansions, and hence accelerates probability computations.

Proposition 3.3. *Given a propositional lineage formula* $\psi := op_j\ \phi_j$ *with* $op \in \{\wedge, \vee\}$ *we equivalently rewrite* ψ *to*

$$\psi' := op_{S_i}(op_{\phi_j \in S_i} \phi_j)$$

where the independent sets $S_1, \dots, S_m$ *are from Definition 3.8. If* $m > 1$*, we have:*

1. $\mathit{Shannon}(\psi') = 0$
2. $\mathit{Shannon}(\psi) = \sum_i \mathit{Shannon}(op_{\phi_j \in S_i} \phi_j)$

Proof. First of all, ψ and ψ' are equivalent, as we applied merely associativity to derive ψ' from ψ. Now, the first statement $\mathit{Shannon}(\psi') = 0$ results from the construction of the sets S_i, whose formulas do not share any tuples across sets. This implies that at the top level operator of ψ' no Shannon expansions are required. Finally, since all tuples occurring in more than one formula ϕ_j must be contained in exactly one set S_i, the last statement is valid. □

The proposition provides an easy way, i.e. by applying associativity, to speed up probability computations. These take an exponential number of steps in the number of Shannon expansions, which we reduced as follows:

$$2^{\mathit{Shannon}(\psi)} \geq \sum_i 2^{\mathit{Shannon}(op_{\phi_j \in S_i} \phi_j)}$$

Example 3.14. *We continue Example 3.13 by considering*

$$\psi = (I_1 \land I_2) \lor (\neg I_2 \land I_3) \lor (I_4 \land I_5) \lor (I_4 \land \neg I_5 \land I_6)$$

which we rewrite to:

$$\psi' = ((I_1 \land I_2) \lor (\neg I_2 \land I_3)) \lor ((I_4 \land I_5) \lor (I_4 \land \neg I_5 \land I_6))$$

Calculating $P(\psi)$ *takes* $2^3 = 8$ *steps, whereas* $P(\psi')$ *consumes only* $2^1 + 2^2 = 6$ *steps.*

We capture the above result in Algorithm 3 implementing two strategies for lowering the number of Shannon expansions. These are repeatedly invoked by the loop in Line 1. In each iteration, we handle one subformula

Algorithm 3 Decompose(ϕ, θ)

```
Input: Lineage formula $\phi$, threshold $\theta$
Output: Adapted formula $\phi$ with reduced Shannon expansions
1: while $\exists \psi \in \phi : \mathit{Shannon}(\psi) > \theta$ do
2:     Select subformula $op_i\ \psi_i$ of $\phi$ with $\mathit{Shannon}(op_i\ \psi_i) > \theta$
3:     Gather independent sets $S_1, \ldots, S_m$, $S_i \subseteq \{\psi_1, \ldots, \psi_n\}$    ▷ Definition 3.8
4:     if $m > 0$ then
5:         Replace $op_i\ \phi_i$ in $\phi$ by $op_{S_i}(op_{\phi_j \in S_i} \phi_j)$    ▷ See Proposition 3.3
6:     else
7:         $I := \arg\max_I |\{\psi_i \mid I \in \mathit{Tup}(\psi_i), 0 \le i \le n\}|$
8:         Replace $op_i \psi_i$ in $\phi$ by $(I \land \psi_{[I/true]}) \lor (\neg I \land \psi_{[I/false]})$ ▷ Equation (2.6)
9: return $\phi$
```

$op_i\ \psi_i$ of ϕ which has more than θ Shannon expansions (Line 2). Then, we attempt to apply Proposition 3.3 which is covered in Lines 3 to 5. Otherwise, we fall back to materializing a single, targeted Shannon expansion. In Line 7, we pick the tuple I which occurs in the maximal number of lineage subformulas ψ_i, such that $\phi_{[I/true]}$, $\phi_{[I/false]}$ have high chances to be simplified.

Example 3.15. *We decompose the propositional lineage formula* $\phi := (I_1 \land I_3) \lor (I_1 \land \neg I_3) \lor (I_2 \land I_3) \lor (I_2 \land \neg I_3)$ *by Algorithm 3 with* $\theta = 0$*. Following Definition 3.7 we obtain* $\mathit{Shannon}(\phi) = 3$*, so we select our entire formula in Line 2. Then, in Line 3 we receive* $m = 0$*, that is there are no independent sets of lineage formulas. The reason is that* I_3 *occurs in all lineage subformulas. Hence, in Line 7, we choose* I_3 *and rewrite* ϕ *to* $(I_3 \land (I_1 \lor I_2)) \lor (\neg I_3 \land (I_1 \lor I_2))$ *in Line 8. Now, all Shannon expansions are gone, so if we compute the probability* $P(\phi)$ *we obtain* $(p(I_3) \cdot (1 - (1 - p(I_1) \cdot (1 - p(I_2)))) + ((1 - p(I_3)) \cdot (1 - (1 - p(I_1)) \cdot (1 - p(I_2)))) = 0.8$ *which can be computed via Equation* (2.4)*.*

In the overall framework, Algorithm 3 of this paragraph should be plugged into Algorithm 1 right after Line 15, such that the lineage formulas are simplified before probability computations are performed. We want to remark that Line 7 of Algorithm 3 is a heuristic only, because the optimal decision on which variable to expand is $\mathcal{NP}$-hard [16]. Also, repeated applications of Line 8 may increase the size of ϕ exponentially which is due to the $\#\mathcal{P}$-hardness of probability computations (see Lemma 2.2).

3.6.2 Temporal Deduplication

What Algorithm 1 of Section 2.3.1 misses to process temporal data is a form of deduplication (see Definition 3.5). Hence, in this paragraph we present Algorithm 4, which for a given set of deduced tuples, outputs duplicate-free tuples. Figure 3.2 again illustrates this by the running example of Chapter 3. For a set of tuples $\mathcal{T}_{R^T,\bar{a}}$ with relation R^T and non-temporal arguments $\bar{a}$ it performs a loop over the limits of the tuples' time-intervals. In more detail,

Algorithm 4 Deduplicate($\mathcal{T}_{R^T,\bar{a}}$)

```
Input: Tuples $\mathcal{T}_{R^T,\bar{a}}$ of relation $R^T$, non-temporal arguments $\bar{a}$
Output: Deduplicated tuples according to Definition 3.5
 1: $Limits := \{t_b, t_e \mid R^T(\bar{a}, t_b, t_e) \in \mathcal{T}_{R,\bar{a}}\}$
 2: $Begin := Map[t \rightarrow \{I \mid I = R^T(\bar{a}, t, t_e) \in \mathcal{T}_{R^T,\bar{a}}\}]$
 3: $End := Map[t \rightarrow \{I \mid I = R^T(\bar{a}, t_b, t) \in \mathcal{T}_{R^T,\bar{a}}\}]$
 4: $t_{last} := first(Limits)$
 5: $Active := \emptyset$
 6: $Result := \emptyset$
 7: for $t \in Limits$ in ascending order do
 8:     if $Active \neq \emptyset$ then
 9:         $I_{new} := R^T(\bar{a}, t_{last}, t)$
10:         $\lambda(I_{new}) := \bigvee_{I \in Active} \lambda(I)$        ▷ See Equation (3.5)
11:         $Result := Result \cup \{I_{new}\}$
12:         $t_{last} := t$
13:     $Active := (Active \setminus End(t)) \cup Begin(t)$
14: return $Result$
```

the set *Limits* contains all endpoints of intervals in $\mathcal{T}_{R^T,\bar{a}}$. *Begin* and *End* are maps pointing from a time point t to the set of tuples beginning from t and ending at t, respectively. During the execution of the loop (Line 7), the set *Active* contains all tuples whose interval contains $[t_{last}, t)$ (Line 13). In Lines 9 and 10, we produce a new tuple I_{new} whose lineage is the disjunction of the lineage of all tuples in *Active*, which resembles Equation (3.5).

Example 3.16. *We apply Algorithm 4 to the deduced tuples presented in the middle of Figure 3.2. First, we initialize* Limits *as* $\{1936\text{-}11\text{-}01, 1976\text{-}$

07-29, 1988-12-01, t_{max}} and set

$$\begin{aligned} \mathit{Begin} &:= [1936\text{-}11\text{-}01 \rightarrow \{I_3 \wedge I_5, I_3 \wedge \neg I_5\}, 1976\text{-}07\text{-}29 \rightarrow \{I_4 \wedge I_5, I_4 \wedge \neg I_5\}] \\ \mathit{End} &:= [1988\text{-}12\text{-}01 \rightarrow \{I_3 \wedge I_5, I_4 \wedge I_5\}, t_{max} \rightarrow \{I_3 \wedge \neg I_5, I_4 \wedge \neg I_5\}] \end{aligned}$$

Then, in the first iteration of the loop, Active *is empty. In the next iteration, we add* $I_6 = \text{AreMarried}(\text{DeNiro}, \text{Abbott}, 1936\text{-}11\text{-}01, 1976\text{-}07\text{-}29)$ *to* Result *and set its lineage to* $\lambda(I_6) = (I_3 \wedge I_5) \vee (I_3 \wedge \neg I_5)$*. The last two iterations produce* I_7*,* I_8 *as shown on top of Figure 3.2.*

In the full framework, we execute the algorithm of this paragraph in Algorithm 1 in Line 11. Furthermore, the runtime complexity of Algorithm 4 is in $O(|\mathcal{T}_{R^T,\bar{a}}| log |\mathcal{T}_{R^T,\bar{a}}|)$, since the loop requires the sorting of *Limits*. Finally, the worst-case size of the output is $2 \cdot |\mathcal{T}_{R^T,\bar{a}}| - 1$, which occurs when all tuples' time-intervals are stacked.

3.7 Experiments

We evaluate our data model and algorithm by focusing on three different aspects. In Section 3.7.1, we model a temporal information extraction setting to showcase the applicability of our data model. In Section 3.7.2, we focus on our runtime performance for query answering tasks. In Section 3.7.3 we study how we perform on large lineage and constraint instances. Finally, in Section 3.7.4 we analyze the detailed runtimes of the individual algorithmic tasks involved in these steps. In all these experiments our system is referred to as *TPDB*. Also, if not stated otherwise, the only parameter, θ of Algorithm 3, is set to 4.

3.7.1 Temporal Information Extraction

Temporal information extraction is centered around mining factual knowledge along with a time annotation from free text. Hence, all challenges of information extraction (see Section 2.5) arise, but additionally time expressions in text have to be recognized. For instance, the sentence "In 1997, De Niro married his second wife, actress Grace Hightower, at their Marbletown home."[2] contains the temporal fact that De Niro and Hightower got married in 1997. As case study for our data model, we implement this kind of temporal information extraction problem in it.

Competitors We compare our model against state-of-the-art methods for temporal tuple extraction, that is constraint solving via an integer linear program [109] and probabilistic inference via Markov logic networks [120].

[2] `http://en.wikipedia.org/wiki/Deniro` (accessed January 14th, 2014)

To solve integer linear programs we deploy the *Gurobi* software[3] implementing the constraints from [140, 153]. For Markov logic networks, we employ "Markov TheBeast"[4] (*TheBeast*), which performs cutting plane inference [121]. Since Markov logic networks do not natively support time, we encode time-intervals by adding two arguments on the relational predicates denoting their begin and end time points. Both competitors perform maximum-a-posteriori [64] inference, i.e. they return the single most likely possible world, rather than computing probabilities as in our model. Furthermore, we tried Markov logic networks implementations which compute probabilities [102, 120] but even after extensive tuning they either exceeded the available memory, disk space, or ran for several days without terminating. All systems operate on the same set of deduction rules and constraints.

Dataset Our dataset consists of 1,817 tuples which we extracted from free-text biographies of 272 celebrities crawled from `wikipedia.org`, `imdb.com`, and `biography.com`. The tuples correspond to nine temporal relations, namely $\mathit{AttendedSchool}^T$, Born^T, Died^T, $\mathit{Divorce}^T$, $\mathit{IsDating}^T$, $\mathit{Founded}^T$, $\mathit{GraduatedFrom}^T$, $\mathit{MovedTo}^T$, and $\mathit{Wedding}^T$. Our extractor employs textual patterns from [101] to recognize potential facts, regular expressions to find dates, and the Stanford named entity recognizer [50] to identify names in text, which we then disambiguate heuristically to entities of YAGO2 (see Appendix A.2.1). We assign each pattern and extraction step a probability, such that the probability of each extracted tuple is obtained by multiplying the probabilities of all steps involved in its extraction process.

Ground Truth We manually label 100 randomly chosen tuples from each relation by determining their correct time-intervals from the text and labeled them as false if the extracted tuple was erroneous. Additionally, we equally divide the labeled tuples into a training set for development and a test set for evaluation.

Deduction Rules and Constraints We employed a set of 11 hand-crafted temporal deduction rules, including the ones shown in Example 3.7. Besides deducing the $\mathit{Marriage}^T$ relation from $\mathit{Wedding}^T$ and $\mathit{Divorce}^T$, we aggregate the relations over their observations (as distinguished by their *id*) by writing for example:

$$\mathit{MovedTo}^T(P, L, T_b, T_e) \leftarrow \mathit{MovedToExtraction}^T(P, L, \mathit{Id}, T_b, T_e)$$

These different observations arise as the same fact can be extracted repeatedly, i.e. from different sources. Besides the deduction rules, there are 21

[3]`http://www.gurobi.com`

[4]`https://code.google.com/p/thebeast/`

constraints to enforce relations to be irreflexive, temporally preceding, or temporally disjoint. For instance, by writing

$$\neg(\mathit{Divorce}(P_1, P_2) \wedge P_1 = P_2)$$

we rule out that a person is getting divorced from him or herself, which assures that *Divorce* irreflexive. As for temporal constraints we ensure, for instance, that people are born before they attend a school:

$$\neg(\mathit{Born}^T(P, T_b, T_e) \wedge \mathit{AttendedSchool}^T(P, S, T_b', T_e') \wedge T_b' <^T T_e)$$

Besides the formulas above, all remaining deduction rules and constraints can be found in Appendix A.3.1.

Metrics To evaluate the result quality, we rely on the established metrics *precision*, *recall*, and F_1 [95], but tailored towards temporal data. Hence, our precision and recall metrics reflect the overlap between the obtained tuples' time-intervals and the time-intervals of the ground-truth. For this, we define the function *Valid*. For a set of tuples in a temporal relation instance $\mathcal{R}^T$ with identical non-temporal arguments $\bar{a}$ and a probability threshold θ_p, *Valid* returns the set of time points at which these tuples are valid with a probability of at least θ_p.

$$\mathit{Valid}(\mathcal{R}^T, \bar{a}, \theta_p) := \left\{ t \;\middle|\; \begin{array}{c} I = R^T(\bar{a}, t_b, t_e) \in \mathcal{R}^T, \\ t \in [t_b, t_e),\ P(\lambda(I)) \geq \theta_p \end{array} \right\} \tag{3.7}$$

Based on this set we define our quality metrics.

Definition 3.9. *For a temporal relation instance $\mathcal{R}^T$, a corresponding ground truth instance $\mathcal{R}^T_{truth}$, non-temporal arguments $\bar{a}$ and a probability threshold θ_p, we define* precision *and* recall *as follows:*

$$\begin{aligned} \mathit{Precision}(\mathcal{R}^T, \bar{a}, \theta_p) &:= \frac{|\mathit{Valid}(\mathcal{R}^T, \bar{a}, \theta_p) \cap \mathit{Valid}(\mathcal{R}^T_{truth}, \bar{a}, \theta_p))|}{|\mathit{Valid}(\mathcal{R}^T, \bar{a}, \theta_p)|} \\ \mathit{Recall}(\mathcal{R}^T, \bar{a}, \theta_p) &:= \frac{|\mathit{Valid}(\mathcal{R}^T, \bar{a}, \theta_p) \cap \mathit{Valid}(\mathcal{R}^T_{truth}, \bar{a}, \theta_p))|}{|\mathit{Valid}(\mathcal{R}^T_{truth}, \bar{a}, \theta_p)|} \end{aligned}$$

The harmonic mean of precision and recall establishes the F_1 measure:

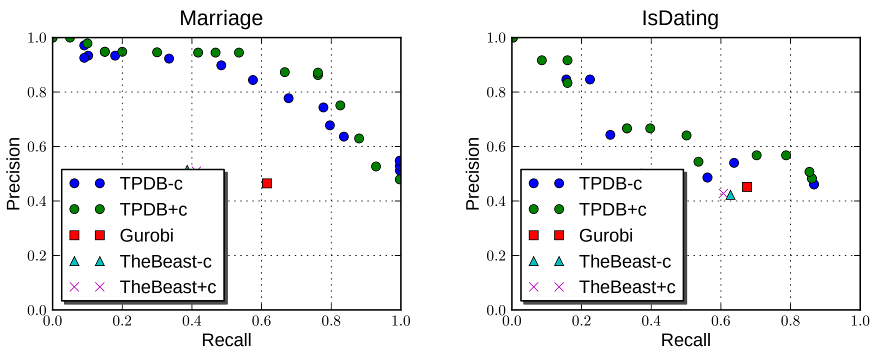
$$F_1(\mathcal{R}^T, \bar{a}, \theta_p) := \frac{2 \cdot \mathit{Precision}(\mathcal{R}^T, \bar{a}, \theta_p) \cdot \mathit{Recall}(\mathcal{R}^T, \bar{a}, \theta_p)}{\mathit{Precision}(\mathcal{R}^T, \bar{a}, \theta_p) + \mathit{Recall}(\mathcal{R}^T, \bar{a}, \theta_p)}$$

Intuitively, precision measures what fraction of time-points is correct. Recall calculates the fraction of correct time-points covered by the extracted data. Finally, F_1 combines both into one value.

Example 3.17. *Consider the ground truth tuple I_g :* Divorce^T(DeNiro, Abbott, 1988-09-01, 1988-09-02) *and I_5 from Figure 3.1. For $\theta_p = 0.7$, we obtain a precision of about* 0.01 *and recall of* 1.0 *which yields a F_1 value of approximately* 0.02*.*

To establish precision and recall for sets of tuples with different non-temporal arguments, we report the macro-average of the individual tuples' values.

Results In the resulting plots we distinguish between the setups of system with constraints (*+c*) and without constraints (*-c*). In Figure 3.4(a) we



(a) Precision & Recall (varying θ_p)

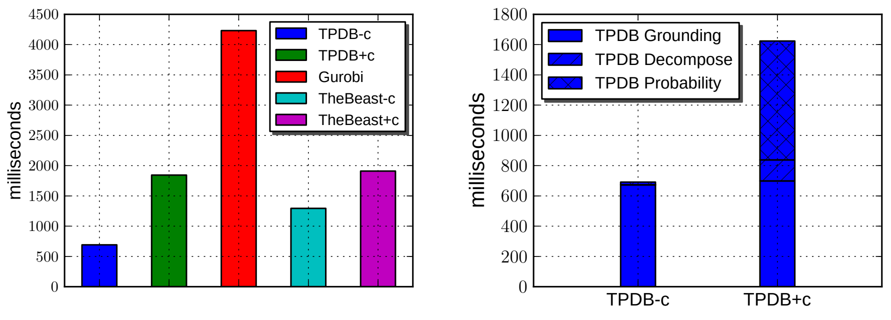
	TPDB-c	TPDB+c	Gurobi+c	TheBeast-c	TheBeast+c
Marriage	0.76	**0.81**	0.52	0.44	0.46
AttendedSchool	**0.72**	**0.72**	0.54	0.66	0.68
Born	0.83	0.84	**0.87**	0.80	0.80
Died	**0.70**	0.62	0.48	0.46	0.53
Founded	**0.80**	**0.80**	0.38	0.73	**0.80**
GraduatedFrom	**0.74**	**0.74**	0.70	0.68	0.67
IsDating	0.62	**0.66**	0.54	0.51	0.50
MovedTo	0.75	0.77	**0.79**	0.69	0.74
Average	0.74	**0.75**	0.61	0.62	0.65

(b) F_1 measure (best θ_p)

Figure 3.4: Temporal Information Extraction: Quality

depict precision over recall for the *Marriage*^{T} and *IsDating*^{T} relation, where each point of *TPDB* corresponds to a different threshold θ_p. The plots of the remaining six relations can be found in Appendix A.3.1. As an overview, we additionally provide Figure 3.4(b), which reports the best F_1 for each method within each relation. Moreover, we measured the runtimes of the competing systems which we display in Figure 3.5.

Discussion To demonstrate the challenges of the dataset, about 700 of the 1,817 tuples violated at least one constraint and about 500 of the 700 tuples contradicted more than one constraint. Considering the F_1 metric, *TPDB* performs best, where the addition of constraints shifts the focus from recall towards precision and sometimes yields major quality gains (see Fig-



(a) System Comparison (b) Our System Detailed

Figure 3.5: Temporal Information Extraction: Runtimes

ure 3.4(a)). *Gurobi* and *TheBeast* perform well, but the single possible world returned by their maximum-a-posteriori inference results in less stable F_1 values (see Figure 3.4(b)). Considering runtimes, in the case without constraints *TPDB* is the fastest system. When adding the constraints, *TheBeast* and *TPDB* take about the same time, whereas *Gurobi* is slower.

3.7.2 Querying

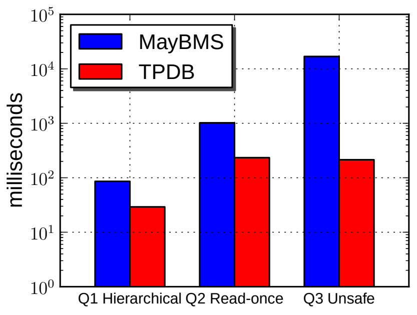
We first focus on answering specific queries over uncertain data, where we compare our implementation *TPDB* to the established probabilistic database system *MayBMS* [10].

Queries We focus on three established query classes in probabilistic databases, namely *hierarchical* [139], *read-once* [129], and *unsafe* [31] queries, which we run over the knowledge base YAGO2 (see Appendix A.2.1). We define three query patterns, one for each class, which we instantiate with different constants. Each pattern yields 1,000 distinct queries with varying lineages and numbers of answers.

The *hierarchical* query $Q1$ forms a union of two join queries and a single relation. Given the query $\mathit{Result}(P)$, we replace *Constant* by 1000 different entities:

$$\begin{array}{rcl}
\mathit{Result}(P) & \leftarrow & \mathit{IsMarriedTo}(\mathit{Id}, \mathit{Constant}, P) \\
\mathit{Result}(P_2) & \leftarrow & \mathit{Edited}(\mathit{Id}, \mathit{Constant}, P_1) \wedge \mathit{ActedIn}(\mathit{Id}_2, P_2, P_1) \\
\mathit{Result}(P_2) & \leftarrow & \mathit{IsLeaderOf}(\mathit{Id}, \mathit{Constant}, P_1) \wedge \mathit{LivesIn}(\mathit{Id}_2, P_2, P_1)
\end{array}$$

to obtain varying queries and lineage. The deduction rules of query $Q2$



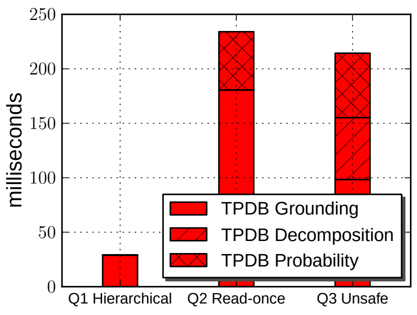


Figure 3.6: Querying Experiments

yielding *read-once* lineage look as follows:

$$\begin{array}{lcl} \mathit{Result}(P) & \leftarrow & \mathit{DiedIn}(\mathit{Id}, P, \mathit{Constant}) \wedge \mathit{HasGivenName}(\mathit{Id}_2, P, N) \\ \mathit{Result}(P) & \leftarrow & \mathit{DiedIn}(\mathit{Id}, P, \mathit{Constant}) \wedge \mathit{ActedIn}(\mathit{Id}_2, P, M) \\ \mathit{Result}(P) & \leftarrow & \mathit{DiedIn}(\mathit{Id}, P, \mathit{Constant}) \wedge \mathit{WasBornIn}(\mathit{Id}_2, L, \mathit{Constant}) \end{array}$$

We notice that *DiedIn* occurs in all three deduction rules. Hence, the resulting lineage formulas are not immediately in read-once form, but they can be transformed into it. Finally, the *unsafe* query $Q3$ is a Boolean query over a single deduction rule whose lineage alters with every of the 1000 constants we insert:

$$\mathit{Result}(0) \leftarrow \left(\begin{array}{c} \mathit{ActedIn}(\mathit{Id}_0, P_1, \mathit{Constant}) \wedge \mathit{WasBornIn}(\mathit{Id}_1, P_2, L) \\ \wedge \mathit{DiedIn}(\mathit{Id}_2, P_2, L) \wedge \mathit{WasBornOnDate}(P_2) \end{array} \right)$$

Results In Figure 3.6 we report the average runtime over the 1,000 queries for each class. The figure consists of two plots. The left one depicts the runtimes for *MayBMS* and *TPDB*, The right one how the runtime of *TPDB* is spent on the tasks performed by Algorithm 1 (*TPDB Grounding*), Algorithm 3 (*TPDB Decomposition*), and on probability computations of Section 2.2.5 (*TPDB Probability*).

Discussion The probability computations in hierarchical queries ($Q1$) are polynomial, i.e., they are completely captured by Equation (2.4). Hence both systems perform very well. Considering read-once lineage ($Q2$), probability computations are in polynomial time, however they might require a conversion of the lineage formula by the algorithm provided in [129]. This becomes evident by the time spent in probability computations (*TPDB Probability*). Also, *MayBMS* slows down slightly. Moving to unsafe queries ($Q3$), which are $\#\mathcal{P}$-hard, our lineage decomposition step (*TPDB Decomposition*) is heavily used. The reason is that there are on average there are about 4,000 database tuples in the lineage formula of each answer. We believe that the speed-ups in comparison to *MayBMS* result from the facts that our lineage is implemented in-memory as well as that our data structure can represent propositional lineage formulas taking forms of directed-acyclic graphs if grounding causes this (see Section 6.2).

3.7.3 Scalability

Instead of analyzing query classes, we now tackle queries with large result sizes that occur for instance when materializing major parts of a knowledge base. We designed two queries by focusing on two different challenges, namely producing large lineage formulas and having constraints over many tuples which we execute over YAGO2 (see Appendix A.2.1).

Large Lineage In this case, we run one temporal query $Q4$ by asking for $\mathit{Result}^T(P_1, P_2, T_b, T_e')$. Each answer shares the same $\#\mathcal{P}$-hard subquery (*Expensive*) which involves about 150,000 database tuples.

$$\begin{aligned}
\mathit{Expensive}(\mathit{result}) &\leftarrow \begin{pmatrix} \mathit{IsLocatedIn}(Id_0, L_1, L_2) \wedge \mathit{WasBornIn}(Id_1, P, L_1) \\ \wedge \mathit{LivesIn}(Id_2, P, L_3) \end{pmatrix} \\
\mathit{Result}^T(P_1, P_2, T_b, T_e') &\leftarrow \begin{pmatrix} \mathit{IsMarriedTo}(Id_0, P_1, P_2) \wedge \mathit{Expensive}(\mathit{result}) \\ \wedge \mathit{OccursSince}^T(Id_1, Id_0, T_b, T_e) \\ \wedge \mathit{OccursUntil}^T(Id_2, Id_0, T_b', T_e') \end{pmatrix}
\end{aligned}$$

Figure 3.7 (left part) depicts the runtimes of *MayBMS* and *TPDB*, where the lineage decompositions of *TPDB* pay off by a large margin.

Many Constraints In query $Q5$, we first instantiate six deduction rules to reconcile knowledge about persons, for example

$$\begin{aligned}
\mathit{Born}^T(P, T_b, T_e) &\leftarrow \mathit{WasBornOnDate}^T(Id, P, T_b, T_e) \\
\mathit{HasChild}^T(P_1, P_2, T_b, T_e) &\leftarrow \begin{pmatrix} \mathit{HasChild}(Id, P_1, P_2) \\ \wedge \mathit{WasBornOnDate}^T(Id_2, P_2, T_b, T_e) \end{pmatrix}
\end{aligned}$$

where the remaining rules are given in Appendix A.3.1. Then, we formulate 17 constraints on the intentional predicates of the deduction rules. These are either temporal precedence constraints, e.g.

$$\neg(\mathit{Born}^T(P_1, T_b, T_e) \wedge \mathit{HasChild}^T(P_1, P_2, T_b', T_e') \wedge T_b' <^T T_e)$$

or temporal disjointness constraints of the form:

$$\neg \begin{pmatrix} \mathit{Marriage}^T(P_1, P_2, T_b, T_e) \wedge \mathit{Marriage}^T(P_1, P_3, T_b', T_e') \\ \wedge P_2 \neq P_3 \wedge T_b <^T T_b' \wedge T_b' <^T T_e \end{pmatrix}$$

We let *TPDB* compete with *Gurobi*[5], a commercial solver for integer linear programs implementing the constraints of [153], and *TheBeast* [121], the

[5]`http://www.gurobi.com`

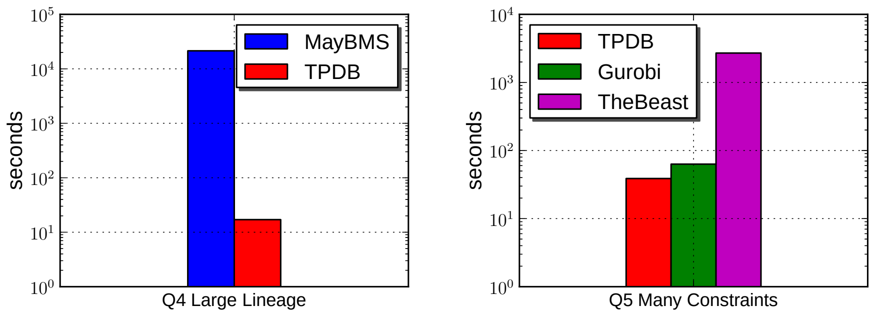


Figure 3.7: Scalability Experiments

fastest Markov logic networks [120] implementation we are aware of. Figure 3.7 (right part) holds the runtimes for the inference omitting grounding times for all systems. *TPDB* solves the problem in less than 40 seconds while *Gurobi* is not able to find the optimal solution anymore and finishes only after 60 seconds. Last, *TheBeast* spends about 40 minutes.

3.7.4 Algorithm Analysis

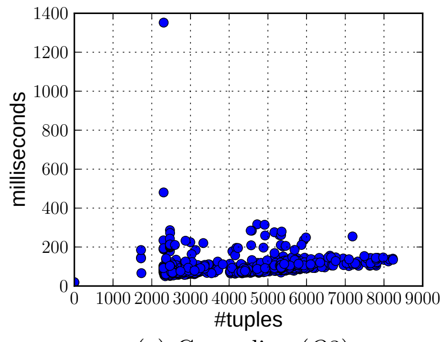
We conclude our evaluation by exploring how runtimes are spent among the steps of *TPDB*, i.e., grounding, lineage decompositions, probability computations, and deduplication. As for the queries, we mostly reuse $Q3$.

Grounding Figure 3.8(a) depicts the grounding time (Algorithm 1) of each of the 1,000 queries in $Q3$. We observe a smooth behavior as the number of base tuples grows (except for an outlier).

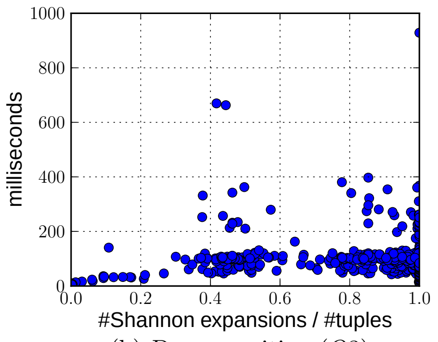
Decomposition Figure 3.8(b) shows the decomposition time (Algorithm 3) for each of the 1,000 queries in $Q3$. The x-axis shows the fraction of base tuples participating in a Shannon expansion over all touched database tuples. Hence, more difficult lineage formulas appear towards the right-hand side of the x-axis.

Probability Computations Relying on the same x-axis, Figure 3.8(c) depicts the runtime for probability computations (see Equations (2.4) and (2.5)) for each query of $Q3$. Due to the decomposition, most queries are handled very fast. Still, for more difficult queries the runtime grows as we perform more Shannon expansions.

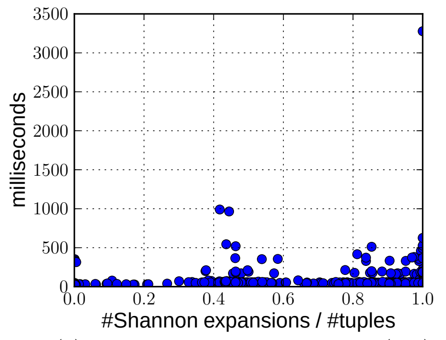
Deduplication For Algorithm 4, we created query pattern $Q6$ and instantiated it into 1,000 queries, such that varying numbers of time-intervals



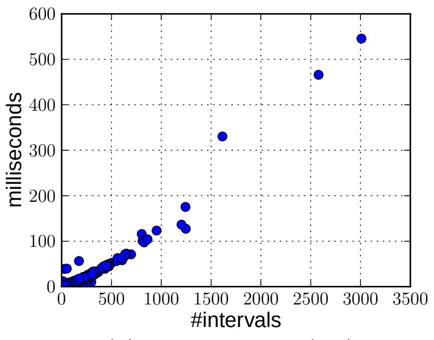
(a) Grounding ($Q3$)



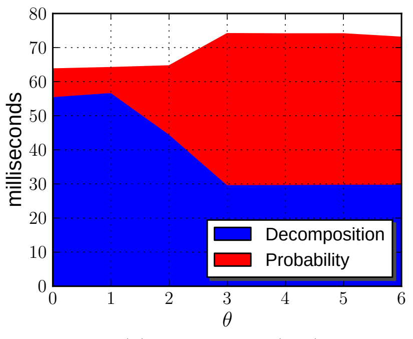
(b) Decomposition ($Q3$)



(c) Probability Computations ($Q3$)



(d) Deduplication ($Q6$)



(e) Varying θ ($Q3$)

Figure 3.8: Algorithm Analysis Experiments

are involved in the single answer of $Q6$. The corresponding deduction rules look as follows:

$$\begin{array}{rcl}
\mathit{Born}^T(P, T_b, T_e) & \leftarrow & \left(\begin{array}{c} \mathit{WasBornIn}(\mathit{Id}, P, \mathit{Constant}) \wedge \\ \mathit{WasBornOnDate}^T(\mathit{Id}', P, T_b, T_e) \end{array} \right) \\
\mathit{Died}^T(P, T_b, T_e) & \leftarrow & \left(\begin{array}{c} \mathit{DiedIn}(\mathit{Id}, P, \mathit{Constant}) \wedge \\ \mathit{DiedOnDate}^T(\mathit{Id}', P, T_b, T_e) \end{array} \right) \\
\mathit{Lives}^T(P, T_b, T_e') & \leftarrow & \mathit{Born}^T(P, T_b, T_e) \wedge \mathit{Died}^T(P, T_b', T_e') \\
\mathit{Lives}^T(P, t_{min}, T_e') & \leftarrow & \neg \mathit{Born}^T(P, T_b, T_e) \wedge \mathit{Died}^T(P, T_b', T_e') \\
\mathit{Lives}^T(P, T_b, t_{max}) & \leftarrow & \mathit{Born}^T(P, T_b, T_e) \wedge \neg \mathit{Died}^T(P, T_b', T_e')
\end{array}$$

We eventually query for $\mathit{Lives}(P, T_b, T_e)$. Figure 3.8(d) depicts the runtime of *TPDB* over the number of time-intervals in the query answers. As pointed out in Section 3.6.2, the runtime follows a $n \log n$ shape, which is confirmed by the plot.

Varying θ As a final experiment we investigate the impact of the parameter θ of Algorithm 3 on the runtime of both decompositions and probability computations. As queries we reuse the $\#\mathcal{P}$-hard queries $Q3$. In Figure 3.8(e) we display the cumulative runtime of both Algorithm 3 (*Decomposition*) and the probability computations of Section 2.2.5 (*Probability*). Setting $\theta = 0$, i.e. at the left of the x-axis, removes all Shannon expansions from all lineage formulas, which results in more runtime spent in decomposition. On the other hand, higher values for θ leave a number of Shannon expansions behind, which speeds up decomposition, but slows down probability computations. The sweet spot seems to be at $\theta = 0$ which reinforces work on decomposition trees [47].

3.8 Summary and Outlook

Contribution In this chapter we presented a temporal probabilistic database model that supports both time and probability as first class citizens. We introduced an expressive class of temporal deduction rules and temporal consistency constraints. Moreover, we extended lineage to be aware of both time and uncertainty and by that obtained a closed and complete representation formalism. Furthermore, we analyzed the properties of our data model from a theoretical perspective by characterizing its complexity properties besides its relationship to sequenced semantics and temporal coalescing. Also, we proposed an efficient algorithm which can be plugged into any probabilistic database with lineage to enable support for temporal data. Finally, we experimentally evaluated the data model and algorithm on an temporal information-extraction task as well as over a large temporal knowledge base.

Future Directions We propose to investigate the support for periodical temporal data, which could be achieved by extending deduction rules to support recursion or integer arithmetic. Additionally, we encourage to extend this work to non-independent database tuples where *block independent-disjoint* probabilistic databases [30] and *U-relations* [10] are prime targets.

Chapter 4

Top-k Query Processing

4.1 Introduction

Driven by the existence of queries whose probability computations are $\#\mathcal{P}$-hard [31, 29], the query evaluation problem in probabilistic databases has been studied extensively [31, 29, 32, 74, 81, 117, 129]. Except for two works [108, 115], each of these approaches aims for computing all answers along with their probabilities. Still, among these answers many feature low probability values, indicating for example that we are not very confident in them or that they are unlikely to exist. To avoid this, we can rely on top-k query answering where only the k most probable answers are computed. Besides the benefit of presenting only the high-confidence answers to the user, top-k approaches allow for runtime speed-ups. The reason is that we can save on computations for the neglected low probability answers.

Existing Top-k Approaches So far, there are only two methodologies [108, 115] for determining the k answers with the highest probabilities. Both of them first compute the lineage for all answers. Then, they invoke different strategies for binding the probabilities of each answer candidate. In [115] samples are drawn where increasing numbers of samples shrink the probability bounds of each answer. Instead, the authors of [108] employ incremental decompositions of the lineage formula of an answer candidate to derive probability bounds. Later, both works rely on these bounds to prune low probability answer candidates. The pruning via the bounds is achieved as in top-k algorithms on deterministic databases [44].

Our Approach In contrast to previous work, we drop the assumption that full data computations should be performed before developing probability bounds. In other words, we do not derive the lineage formulas of all answer candidates extensively, but rather expand them as necessary over the database while already pruning answer candidates. This technique generates another source of speed-ups both over the aforementioned top-k methodologies and over approaches that yield all query answers. We experimentally study this effect in Section 4.7.

Technically, our main tool are *first-order lineage* formulas as they capture any intermediate grounding state (see Section 4.4). These formulas can represent entire sets of answer candidates, for instance when we are starting to ground the query. Based on the insights of [108], we then devise *probability bounds* enclosing the probabilities of all answers in the set of candidates which is expressed by a first-order lineage formula (see Section 4.4.3). We prove that these bounds converge monotonically which enables us to formulate a *top-k algorithm* (see Section 4.5.2). As a subroutine we devise a *scheduling algorithm* which selects the next literal to be evaluated relying on both information about the database as well as the probability bounds (see Section 4.5.1). Finally, we provide *extensions* to sorted input lists, recursive deduction rules, and consistency constraints (see Section 4.6).

Problem Statement As *input* we are given:

- a *tuple-independent probabilistic database*;
- a *query* represented by deduction rules;
- an *integer* k;
- and optionally *constraints*.

Then, our top-k procedure delivers the *output*:

- the *set of cardinality* k or less containing the *query answers*, which have the *highest probabilities*.

We illustrate this by the following example.

Example 4.1. *Figure 4.1 depicts a probabilistic database in the movie domain. By the given deduction rules we intend to derive actors and directors who are known for working on movies in the crime genre as symbolized by the query* $\text{KnownFor}(X, \text{Crime})$. *When we execute traditional query evaluation, e.g. following Algorithm 1, we obtain the three answers:*

Answer	*Lineage*	*Probability*
$KnownFor(Coppola, Crime)$	$I_2 \wedge I_8 \wedge I_{12}$	0.36
$KnownFor(Tarantino, Crime)$	$I_{10} \wedge I_6 \wedge \neg I_3 \wedge I_{13}$	0.10
$KnownFor(Pacino, Crime)$	$I_9 \wedge I_5 \wedge I_{12}$	0.06

Now, imagine we are not interested in all answers, as in the table above, but rather in the most probable answer, e.g. $\text{KnownFor}(\text{Coppola}, \text{Crime})$. *This is the setting of the present chapter. We will elaborate on how to compute the k most likely answers efficiently by (1) not fully computing lineage and (2) pruning other less probable answers, i.e.* $\text{KnownFor}(\text{Tarantino}, \text{Crime})$ *and* $\text{KnownFor}(\text{Pacino}, \text{Crime})$, *as early as possible.*

Query:

$$KnownFor(X, Crime)$$

Deduction Rules:

$$\begin{array}{lll} KnownFor(X,Y) & \leftarrow & BestDirector(X,Z) \wedge Category(Z,Y) \\ KnownFor(X,Y) & \leftarrow & WonAward(Z,BestPicture) \wedge ActedOnly(X,Z) \wedge Category(Z,Y) \\ BestDirector(X,Z) & \leftarrow & Director(X,Z) \wedge WonAward(Z,BestDirector) \\ ActedOnly(X,Z) & \leftarrow & ActedIn(X,Z) \wedge \neg Directed(X,Z) \end{array}$$

Probabilistic Database:

Directed

	Director	Movie	p
I_1	*Coppola*	*ApocalypseNow*	0.8
I_2	*Coppola*	*Godfather*	0.9
I_3	*Tarantino*	*PulpFiction*	0.7

ActedIn

	Actor	Movie	p
I_4	*Brando*	*ApocalypseNow*	0.6
I_5	*Pacino*	*Godfather*	0.3
I_6	*Tarantino*	*PulpFiction*	0.4

WonAward

	Movie	Award	p
I_7	*ApocalypseNow*	*BestScript*	0.3
I_8	*Godfather*	*BestDirector*	0.8
I_9	*Godfather*	*BestPicture*	0.4
I_{10}	*PulpFiction*	*BestPicture*	0.9

Category

	Movie	Category	p
I_{11}	*ApocalypseNow*	*War*	0.9
I_{12}	*Godfather*	*Crime*	0.5
I_{13}	*PulpFiction*	*Crime*	0.9
I_{14}	*Inception*	*Drama*	0.6

Figure 4.1: Example Probabilistic Database with query and deduction rules

4.2 Related Work

The most influential work for top-k query evaluation on deterministic data is still given by the family of threshold algorithms by Fagin et al. [44]. Our pruning techniques of answer candidates also build on these. In this section, we discuss work on top-k queries over probabilistic databases, where we refer the interested reader to [69] for a survey on top-k algorithms in deterministic databases.

Top-k on Probabilistic Databases: Ranking by Scores The majority of works on top-k queries over probabilistic databases assumes the presence of an additional non-probabilistic score at each tuple, which is used to rank the query answers [70]. Our approach is very different, because we rank every answer by its probability and do not assume another score to exist. Given a non-probabilistic score for the ranking the main question is how to interpret the top-k answers with respect to the probabilities and possible worlds. There, numerous different semantics arise. Soliman et al. [135] introduce *U-topK* queries, which are the k tuples with the highest probability to be contained in the top-k list in all possible worlds. Also, they devise *U-kRanks*, where the k-th rank is filled by the tuple which has the highest probability to rank at this position over all possible worlds. Alternatively, [72] considers the expected rank over all possible worlds. Li et al. [88] propose a unified approach via parametrized ranking functions which subsumes most aforementioned approaches. In this setting, [151] develops pruning approaches to effectively compute the top-k answers. In [22] the parametrized ranking functions are extended to support attribute uncertainty. Moreover, in [67, 68] besides k a probability threshold is given. Then, only tuples which rank at least k with probability above the threshold are returned. Similarly, [89] lists all ranks at which the answer can be found with probability higher than a threshold. Recently, Ge et al. [55] studied the trade-offs between reporting tuples of a high score and tuples of a high probability. Furthermore, [86] considered ranking in the presence of both scores and continuous probabilities, i.e. occurring in sensor data. Finally, in [136] the scores used for ranking are uncertain, but the considered ranking function is still the sum of these scores.

Top-k on Probabilistic Databases: Ranking by Probabilities Few works consider top-k ranking by the probabilities of query answers as in our setting. The authors of [115] determine the top-k answers by first computing the lineage formulas of all answers and then approximating their probabilities by sampling. They devise probability bounds restricted to lineage formulas in disjunctive normal form, where the bounds tighten as more samples are drawn. For this, a multisimulation algorithm is introduced which aims to minimize the number of samples to draw before the ranking is known. We compete and outperform this approach in the experiments of Section 4.7.1. Extending works on approximation of probabilities of propositional lineage formulas [107], the authors of [108] derive probability bounds for answer candidates. Our bounding approach is related theirs, but they lack support for first-order logical formulas. In addition, in [108] a query class is established for which ranking is tractable, whereas absolute probability computations are not, an insight which is exploited by shared static query plans. We provide experiments on this query class in Section 4.7.1. Likewise, [87] computes the top-k answers with highest probabilities over probabilistic XML. Still, their model allows only independence and mutual exclusion nodes, which renders their model much simpler than ours. Similarly, in [114] static probability thresholds are incorporated into the query algebra, allowing for early pruning of low probability tuples. However, their approach does not support full relational algebra with duplicate elimination. Finally, in the case of continuous probabilities [110] thresholds bind probability distributions rather than being possible world based as in our case.

Bounds on Probabilities Bounds on probabilities are a major tool for running top-k algorithms, since they allow for pruning answer candidates without fully computing the candidates' probabilities. In this field, there are four major works, which we build upon for the propositional lineage case. In detail, [48] relies on (disjunctive) read-once lineage formulas as approximation for a given propositional lineage formula such that the probability bounds can be computed efficiently. Next, [107] presents error bounds which are obtained by incrementally transforming propositional lineage into a tree form. Each vertex in the tree corresponds to a Shannon expansion or independent conjunctions and disjunctions. Nevertheless, their work is restricted to formulas in disjunctive normal form. This restriction is consequently dropped in [49]. Finally, in [116] approximate propositional lineage is discussed which are smaller formulas, for instance implying the real formulas. By construction the probability of the approximate lineage can serve as probability bound on the full lineage. None of them, however, explicitly considers first-order logical formulas.

First-Order Lineage Although queries are in first-order form [139], work on lineage formulas in probabilistic databases focused on the propositional version. The first work on probability computations on first-order lineage formulas is yet to appear [12]. In contrast, within the artificial intelligence community inference on first-order logical formulas, i.e. probability computations, are established, for instance on belief networks [112] or via knowledge compilation [149].

4.3 Contribution

In this section we present a novel top-k querying approach [41] for probabilistic databases, where the query answers are ranked by their probabilities.

- To the best of our knowledge we present the first top-k approach on probabilistic databases supporting *partially grounded lineage* formulas, which we represent by *first-order lineage* formulas (Section 4.4). This method allows early pruning of entire sets of answer candidates, and hence can provide savings even on the data computation step (see Figure 4.3). Also, it does not assume any restriction on the query structure.

- We present a generic bounding approach for probability computations over first-order lineage formulas in Section 4.4.3, which provides (1) *lower* and *upper bounds* for the probability of an individual query answer, or for an entire set of query answers if not all query variables are bound to constants yet. We show that (2) both our lower and upper bounds *converge monotonically* to the final probabilities of the query answers as we gradually expand these formulas. Both (1) and (2) are key properties for building effective top-k-style pruning algorithms.

- Our approach allows for plugging in *different schedulers* which aim to select the first-order literal inside a lineage formula that is most beneficial for top-k pruning at each query processing step (Section 4.5.1). This benefit is estimated based on the expected selectivity, i.e., the expected number of resulting tuples, and the expected impact of literals on the probability bounds of query answers.
- Moreover, we extend our algorithm for the case when *sorted input lists* for extensional relations are available (Section 4.6.1), show how *recursive deduction rules* can be handled (Section 4.6.2), and discuss the usage of our top-k pruning techniques under both *queries and constraints* (Section 4.6.4).
- In Section 4.7 we present an extensive *experimental evaluation* and comparison to existing top-k pruning strategies in probabilistic databases.

4.4 First-Order Lineage

To handle partial grounding states, we now extend propositional lineage of Section 2.2.4 to first-order lineage [41], which hence 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.

Throughout this chapter, we assume relations to be duplicate-free. That is, there is no pair of tuples having the same arguments. This assumption facilitates the theoretical analysis which follows. Still, in practice, we can always remove the duplicates by an independent-project operation as a preprocessing step.

4.4.1 Deduction Rules with Quantifiers

To facilitate the construction of first-order lineage we write (some) quantifiers in deduction rules explicitly, which is captured in the following.

Definition 4.1. *A* first-order deduction rule *is a logical rule of the form*

$$R_0(\bar{X}_0) \leftarrow \exists \bar{X}_e \bigwedge_{i=1,\dots,n} R_i(\bar{X}_i) \wedge \bigwedge_{j=1,\dots,m} \neg R_j(\bar{X}_j) \quad \wedge \Phi(\bar{X})$$

where

1. *all requirements of Definition 2.2 hold;*
2. $\bar{X}_e = (\bigcup_{i=1,\dots,n} Var(\bar{X}_i)) \backslash Var(\bar{X}_0)$

The difference to Definition 2.2 might seem subtle, but we force all variables $\bar{X}_e$, which occur in positive literals $R_i(\bar{X}_i)$, but not in the head $R_0(\bar{X}_0)$, to be existentially quantified. This is in accordance to standard Datalog semantics [2]. Later, when constructing first-order lineage formulas, the existential quantifiers be explicitly kept. Universal quantifiers do not result from deduction rules, but can be introduced into first-order lineage formulas when handling constraints (see Section 4.6.4).

Example 4.2. *Let us adapt the deduction rules of Figure 4.1 to Definition 4.1 by writing the quantifiers explicitly:*

$$\begin{array}{lll} \mathit{KnownFor}(X,Y) & \leftarrow \exists Z & \mathit{BestDirector}(X,Z) \wedge \mathit{Category}(Z,Y) \\ \mathit{KnownFor}(X,Y) & \leftarrow \exists Z & \begin{array}{l}\mathit{WonAward}(Z,\mathit{BestPicture}) \wedge \mathit{ActedOnly}(X,Z) \\ \wedge\, \mathit{Category}(Z,Y)\end{array} \\ \mathit{BestDirector}(X,Z) & \leftarrow & \mathit{Director}(X,Z) \wedge \mathit{WonAward}(Z,\mathit{BestDirector}) \\ \mathit{ActedOnly}(X,Z) & \leftarrow & \mathit{ActedIn}(X,Z) \wedge \neg \mathit{Directed}(X,Z) \end{array}$$

4.4.2 Lineage Construction

First-order lineage can be constructed top-down and can express any intermediate state in this process. In a top-down approach, we start at the query and expand the deduction rules until we reach the database. In Section 2.2.4, the direction was reversed, since we started at the database until we ended up at the query. As first theoretical tool, we establish consistent vectors of constants $\bar{a}$ and mixtures of variables and constants $\bar{X}$. This technique enables us to match first-order literals against database tuples.

Definition 4.2. *Let X_i and a_i denote the i-th entry in the vector of variables and constants $\bar{X}$ and the vector of constants $\bar{a}$, respectively. We call $\bar{X}$ and $\bar{a}$* consistent, *if*

$$\forall X_i \in \bar{X} : X_i \text{ is a constant} \Rightarrow X_i = a_i$$

In other words, all constants in the vector $\bar{X}$ have to match the constant in $\bar{a}$ at the respective position.

Example 4.3. *The vectors* (X, Crime) *and* $(\text{Coppola}, \text{Crime})$ *are consistent, as the constant in the second entry occurs in both vectors.*

Based on consistent vectors we gather all constants binding a variable in a set of tuples. Later, this allows us to collect all tuples from the database, which match a first-order literal.

Definition 4.3. *Let $\mathcal{T}$ be a set of tuples and $R(\bar{X})$ be a literal with extensional relation R. Then, the set of constants from $\mathcal{T}$, which bind the variable X_i in $\bar{X}$ is:*

$$\mathit{Bindings}(X_i, R(\bar{X}), \mathcal{T}) := \{a_i \mid R(\bar{a}) \in \mathcal{T}, \bar{X} \text{ and } \bar{a} \text{ consistent}\}$$

We note that a_i and X_i refer to i-th entry of $\bar{a}$ and $\bar{X}$, respectively. In general, the above set can be empty or reach the same cardinality as $\mathcal{T}$.

Example 4.4. *Let the tuples of Figure 4.1 establish $\mathcal{T}$. Then, considering the literal* Directed(Coppola, Y) *we obtain the following bindings for the variable Y:*

$$\mathit{Bindings}(Y, \mathit{Directed}(\mathit{Coppola}, Y), \mathcal{T}) = \{\mathit{ApocalypseNow}, \mathit{Godfather}\}$$

The last technical prerequisite before introducing first-order lineage construction are logical equivalences which eliminate quantifiers. For this, say $a_1, \dots, a_n$ are all possible constants for the variable X, then following two equivalences [2, 146] hold:

$$\begin{aligned} \exists X \Phi &\equiv \sigma_{a_1}(\Phi) \vee \dots \vee \sigma_{a_n}(\Phi) \\ \forall X \Phi &\equiv \sigma_{a_1}(\Phi) \wedge \dots \wedge \sigma_{a_n}(\Phi) \end{aligned} \tag{4.1}$$

Here, σ_{a_i} is shorthand for $\sigma(X) = a_i$. Finally, we establish the top-down counterpart to Definition 2.11 for first-order lineage. We create first-order lineage starting at the query and then iteratively replacing first-order literals by deduction rules or, later on, tuples from database.

Definition 4.4. *Let a set of tuples $\mathcal{T}$, a set of deduction rules $\mathcal{D}$, a first-order lineage formula Φ, and a literal $R(\bar{X})$ which occurs in Φ be given. We define the expansion of $R(\bar{X})$ in Φ by a function:*

$$\mathit{SLD} : \mathit{Literals} \times \mathit{FirstOrderLineage} \to \mathit{Set}[\mathit{FirstOrderLineage}]$$

In detail:

1. *If R is* intensional*, then:*

$$\mathit{SLD}(R(\bar{X}), \Phi) := \left\{ \Phi[R(\bar{X}) / \textstyle\bigvee_{(R(\bar{X}') \leftarrow \Psi) \in \mathcal{D}} \sigma_{\bar{X}}(\Psi)] \right\}$$

   where $\sigma_{\bar{X}}$'s image coincides with $\bar{X}$.

2. *If R is* extensional*, we initialize:*

$$S_0 := \{\Phi\}$$

   and then iterate over all variables $X \in \mathit{Var}(\bar{X})$:

   (a) If X is a query variable:

$$S_i := \{\sigma_a(\Phi') \mid \Phi' \in S_{i-1}, a \in \mathit{Bindings}(X, R(\bar{X}), \mathcal{T})\}$$

   where $\sigma_a(X) = a$.

(b) *If X is bound by $\exists X$, then we replace the subformula $\exists X\ \Psi$ of Φ in S_i by $\sigma_{a_1}(\Psi) \vee \cdots \vee \sigma_{a_n}(\Psi)$ where all $a_i \in \mathit{Bindings}(X, R(\bar{X}), \mathcal{T})$.*

(c) *If X is bound by $\forall X$, then we replace the subformula $\forall X\ \Psi$ of Φ in S_i by $\sigma_{a_1}(\Psi) \wedge \cdots \wedge \sigma_{a_n}(\Psi)$ where all $a_i \in \mathit{Bindings}(X, R(\bar{X}), \mathcal{T})$.*

Finally, we replace all ground literals $R(\bar{a})$ in the last S_i by their tuple identifier I and assign $\mathit{SLD}(R(\bar{X}), \Phi) := S_i$.

3. *If there is* no match *to $R(\bar{X})$, neither in $\mathcal{T}$ nor in $\mathcal{D}$, then:*

$$\mathit{SLD}(R(\bar{X}), \Phi) := \{\Phi_{[R(\bar{X})/\mathit{false}]}\}$$

4. *If R is* arithmetic *and $\mathit{Var}(\bar{X}) = \emptyset$, then we evaluate $R(\bar{X})$ to V (*true *or* false*), and:*

$$\mathit{SLD}(R(\bar{X}), \Phi) := \{\Phi_{[R(\bar{X})/V]}\}$$

The above definition is admittedly involved. In the first case, we address *intensional* literals $R(\bar{X})$ where we exchange $R(\bar{X})$ for the disjunction of the deduction rules having R in their head literal. Since $\bar{X}$ can contain constants we propagate them to the rules' bodies by writing $\sigma_{\bar{X}}(\Psi)$. *Extensional* literals, which are the subject of the second case, can yield sets of first-order lineage formulas. We proceed by considering each variable individually and distinguish between query variables (see Definition 2.6), existentially bound variables and universally bound variables. If X is a *query variable* each constant a binding X produces a new distinct set of query answers represented by the lineage formula $\sigma_a(\Phi')$. Conversely, if X is *existentially quantified* we apply Equation (4.1) to expand the formula by introducing a disjunction ranging over the constants $a_1, \ldots, a_n$ which bind X. Analogously, a *universally quantified* X yields the conjunction over the constants $a_1, \ldots, a_n$. The third case reflects the closed world assumption [2], where we replace a literal with *no match* by *false*. Finally, if we have an *arithmetic* literal with only constants as arguments, we evaluate it. We can safely assume that all arguments of the arithmetic literal are constants, because these are bound in non-arithmetic literals (see Definition 2.2). What we omitted for brevity are constants in the head literal of a deduction rule. Since these constants bind variables as in extensional literals (the second case), a mixture of the first and second case arises.

Example 4.5. *We illustrate Definition 4.4 by providing an example for each case. As for $\mathcal{T}$ we assume it to comprise all tuples of Figure 4.1.*

1. *We expand the formula $\Phi :=$ KnownFor$(X,$ Crime$)$ over the deduction rules of Example 4.2. Since* KnownFor *is an intensional relation, we start with the first case of Definition 4.4. There, the substitution $\sigma_{\bar{X}}$ binds the second argument to* Crime*:*

$$\sigma_{\bar{X}}(Y) = \mathit{Crime}$$

Since there are two rules having KnownFor *in the head literal we apply the substitution to both bodies which yields:*

$$\left\{\begin{array}{c} (\exists Z\ \mathit{BestDirector}(X,Z) \wedge \mathit{Category}(Z,\mathit{Crime})) \\ \vee \\ \left(\exists Z \begin{array}{c} \mathit{WonAward}(Z,\mathit{BestPicture}) \wedge \\ \mathit{ActedOnly}(X,Z) \wedge \mathit{Category}(Z,\mathit{Crime}) \end{array}\right) \end{array}\right\}$$

2. (a) *Imagine, we are given the first-order lineage formula*

$$\Phi := \mathit{BestDirector}(X,Z) \wedge \mathit{Category}(Z,\mathit{Crime})$$

and we intend to expand the literal Category$(Z,$ Crime$)$. *Here,* Category *is an extensional relation. First, we determine the bindings of Z, which are* Godfather *and* PulpFiction. *Since Z is not quantified, but a query variable, we obtain several formulas, one for each of the constants:*

$$\left\{\begin{array}{c} (\mathit{BestDirector}(X,\mathit{Godfather}) \wedge \mathit{Category}(\mathit{Godfather},\mathit{Crime})), \\ (\mathit{BestDirector}(X,\mathit{PulpFiction}) \wedge \mathit{Category}(\mathit{PulpFiction},\mathit{Crime})) \end{array}\right\}$$

(b) *In this case, we quantify Z existentially and otherwise keep the previous formula:*

$$\Phi := \exists Z\ \mathit{BestDirector}(X,Z) \wedge \mathit{Category}(Z,\mathit{Crime})$$

Then, we expand the Category *literal by case 2(b) of Definition 4.4 which results in a disjunction over the two constants* Godfather *and* PulpFiction*:*

$$\left\{\begin{array}{c} (\mathit{BestDirector}(X,\mathit{Godfather}) \wedge \mathit{Category}(\mathit{Godfather},\mathit{Crime})) \\ \vee \\ (\mathit{BestDirector}(X,\mathit{PulpFiction}) \wedge \mathit{Category}(\mathit{PulpFiction},\mathit{Crime})) \end{array}\right\}$$

(c) *Let us consider a universal quantifier instead:*

$$\Phi := \forall Z\ \mathit{BestDirector}(X,Z) \wedge \mathit{Category}(Z,\mathit{Crime})$$

When applying a SLD step to the Category *literal, we instantiate Z by the two constants* Godfather *and* PulpFiction *to obtain the conjunction:*

$$\left\{\begin{array}{c} (\mathit{BestDirector}(X,\mathit{Godfather}) \wedge \mathit{Category}(\mathit{Godfather},\mathit{Crime})) \\ \wedge \\ (\mathit{BestDirector}(X,\mathit{PulpFiction}) \wedge \mathit{Category}(\mathit{PulpFiction},\mathit{Crime})) \end{array}\right\}$$

3. *Trying to resolve the second literal of*

$$\Phi := \exists Z \ \mathit{BestDirector}(X, Z) \land \mathit{Category}(Z, \mathit{Comedy})$$

over $\mathcal{T}$ delivers no result. Hence, we replace it by false, which yields:

$$\{\exists Z \ \mathit{BestDirector}(X, Z) \land \mathit{false}\}$$

4. *In the last case, we have an arithmetic literal, e.g.:*

$$I_1 \land I_2 \land \text{ApocalypseNow} \neq \text{Godfather}$$

which we then evaluate to $I_1 \land I_2 \land \text{true}$.

Analogously to the Disjunctive Normal Form (DNF) for propositional formulas, any first-order formula can equivalently be transformed into prenex normal form by pulling all quantifiers in front of the formula. The remaining formula can again be transformed into DNF, which is then called Prenex Disjunctive Normal Form (PDNF) [146].

Next, we devise two formal properties of first-order lineage formulas. First, the existence of at least one proof implies that all query variables are bound. Second, unbound query variables imply that a first-order lineage formula represents a set of query answers.

Proposition 4.1. *Expanding a query $Q(\bar{X})$ with query variables $\bar{X}$ to first-order lineage by repeatedly applying Definition 4.4 has the following properties:*

1. *If at least one clause in the disjunctive normal form of the lineage formula is propositional, then all query variables $\bar{X}$ are bound to constants.*

2. *If at least one query variable $X \in \bar{X}$ is unbound a lineage formula represents a (possibly empty) set of query answers.*

Proof. We prove both statements separately.

1. Without loss of generality, we assume the formula to be in PDNF. Then, every clause stands for one proof of the answer candidate. When one of these clauses is propositional, all query variables within this clause were bound and hence are bound in the entire formula.

2. Since a query variable can be bound to many constants, each potentially belonging to a particular answer, the first-order lineage formula represents all these answers.

□

4.4.3 Probability Bounds

In this section, we develop lower and upper bounds for the probability of any query answer that can be obtained from grounding a first-order lineage formula. We proceed by constructing two propositional lineage formulas ϕ_{low} and ϕ_{up} from a given first-order lineage formula Φ. Later, the probabilities of ϕ_{low} and ϕ_{up} serve as lower and upper bounds on the probabilities of all query answers captured by Φ. More formally, if $\phi_1, \ldots, \phi_n$ represent all query answers we would obtain by fully grounding Φ, then it holds that:

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

Building upon results of [48, 107, 124], we start by considering bounds for propositional formulas, from which we extend to the more general case of first-order lineage. Then, we show that these bounds converge monotonically to the probabilities $P(\phi_i)$ of each query answer ϕ_i, as we continue to ground Φ.

Bounds for Propositional Lineage Following [107], we relate the probability of two propositional lineage formulas ϕ and ψ via their sets of models $\mathcal{M}(\phi)$ and $\mathcal{M}(\psi)$ (see Equation (2.2)), i.e. the sets of possible worlds over which ϕ and ψ evaluate to *true*.

Proposition 4.2. *For two propositional lineage formulas ϕ and ψ it holds that:*

$$\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi) \quad \Rightarrow \quad P(\phi) \leq P(\psi)$$

Proof.

$$\begin{array}{rcl} P(\phi) & \overset{\text{Equation (2.3)}}{=} & \sum_{\mathcal{W} \in \mathcal{M}(\phi)} P(\mathcal{W}) \\ & \leq & \sum_{\mathcal{W} \in \mathcal{M}(\phi)} P(\mathcal{W}) + \sum_{\mathcal{W} \in \mathcal{M}(\psi) \backslash \mathcal{M}(\phi)} P(\mathcal{W}) \\ & \overset{\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)}{=} & \sum_{\mathcal{W} \in \mathcal{M}(\psi)} P(\mathcal{W}) \\ & \overset{\text{Equation (2.3)}}{=} & P(\psi) \end{array}$$

□

Since we assume $\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)$, the possible worlds satisfying ϕ fulfill ψ as well. However, there might be more worlds satisfying ψ but not ϕ. This might yield more terms over which the sum of Equation (2.3) ranges, and thus we obtain $P(\phi) \leq P(\psi)$.

Example 4.6. *Consider the two propositional formulas $\phi \equiv I_1$ and $\psi \equiv I_1 \vee I_2$. From $\mathcal{M}(I_1) \subseteq \mathcal{M}(I_1 \vee I_2)$ it follows that $P(I_1) \leq P(I_1 \vee I_2)$, which we can easily verify by Equation* (2.3).

To turn Proposition 4.2 into upper and lower bounds, we proceed by considering *conjunctive clauses* in the form of conjunctions of propositional literals. Then, following a result from [107], we obtain the following proposition.

Proposition 4.3. *Let ϕ, ψ be two propositional, conjunctive clauses. It holds that:*

$$\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi) \quad \Leftrightarrow \quad Tup(\phi) \supseteq Tup(\psi)$$

The above statement expresses that adding literals to a conjunction ϕ removes satisfying worlds from $\mathcal{M}(\phi)$.

Example 4.7. *For the two clauses $I_1 \wedge I_2$ and I_1 it holds that $Tup(I_1 \wedge I_2) \supseteq Tup(I_1)$ and thus Proposition 4.3 yields $\mathcal{M}(I_1 \wedge I_2) \subseteq \mathcal{M}(I_1)$.*

We now establish a relationship between two formulas in Disjunctive Normal Form (DNF) (see Definition 2.12) via their conjunctive clauses as in [107, 124]. Since any propositional formula can be transformed equivalently into DNF, this result is generally applicable.

Lemma 4.1. *For two propositional formulas in disjunctive normal form $\phi \equiv \phi_1 \vee \cdots \vee \phi_n$ and $\psi \equiv \psi_1 \vee \cdots \vee \psi_n$, it holds that:*

$$\forall \phi_i \; \exists \psi_j : \mathcal{M}(\phi_i) \subseteq \mathcal{M}(\psi_j) \quad \Rightarrow \quad \mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)$$

If we can map all clauses ϕ_i of a formula ϕ to a clause ψ_j of ψ with more satisfying worlds, i.e., $\mathcal{M}(\phi_i) \subseteq \mathcal{M}(\psi_j)$, then ψ has more satisfying worlds than ϕ. This mapping of clauses is established via Proposition 4.3.

Example 4.8. *For the propositional DNF formula $\phi \equiv (I_1 \wedge I_2) \vee (I_1 \wedge I_3) \vee I_4$, we can map each clause in ϕ to a clause in $\psi \equiv I_1 \vee I_4$. Hence, ψ has more models than ϕ, i.e., $\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)$.*

Thus, Lemma 4.1 enables us to compare the probabilities of propositional formulas in DNF based on their clause structure. When transforming any propositional formula into DNF, we can first iteratively apply De Morgan's law [146] which pushes negations down in a formula:

$$\begin{array}{c} \neg \bigwedge_i \Phi_i \equiv \bigvee_i \neg \Phi_i \\ \neg \bigvee_i \Phi_i \equiv \bigwedge_i \neg \Phi_i \end{array} \tag{4.2}$$

Thereafter, we apply the distributive law which allows the following observation.

Observation 4.1. *If a tuple I occurs exactly once in a propositional formula ϕ, then all occurrences of I in the DNF of ϕ have the same sign.*

The reason is that the sign of a tuple I changes only when De Morgan's law is applied. However, when applying De Morgan's law, no tuples are duplicated. When utilizing the distributive law, tuples are duplicated but preserve their signs.

Example 4.9. *Applying the distributive law to* $(I_1 \vee I_2) \wedge \neg I_3$ *yields* $(I_1 \wedge \neg I_3) \vee (I_2 \wedge \neg I_3)$*. Now,* I_3 *occurs twice, but its sign was preserved.*

Bounds for First-Order Lineage For our following constructions on first-order formulas, we assume the first-order formulas to be given in PDNF. 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 4.5. *Let* Φ *be a first-order lineage formula.*

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 the PDNF of* Φ*, or*
   - false *if* $R(\bar{X})$ *occurs negated in the PDNF of* Φ*.*
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 the PDNF of* Φ*, or*
   - true *if* $R(\bar{X})$ *occurs negated in the PDNF of* Φ*.*

The idea of the above definition is as follows. If we replace a positive literal by *true*, we add models to the resulting formula. Hence, due to Proposition 4.2 the resulting formula can serve as an upper bound on the probability, which we show formally later. The remaining three cases are analogous. We note that R can be intensional, extensional and even arithmetic.

Example 4.10. *We consider Figure 4.1 and the first-order lineage formula:*

$$\Phi := I_1 \wedge \exists X\, WonAward(X, BestPicture)$$

Then, the upper bound is given by $P(\phi_{up}) = P(I_1 \wedge true) = p(I_1) = 0.8$ *and the lower bound is* $P(\phi_{low}) = P(I_1 \wedge false) = P(false) = 0$*. If we execute one SLD step (see Definition 4.4) on* Φ *we obtain* $I_1 \wedge (I_9 \vee I_{10})$*. Its probability is* $P(I_1 \wedge (I_9 \vee I_{10}) = 0.8 \cdot (1 - (1 - 0.4) \cdot (1 - 0.9) = 0.752$ *which is correctly captured by the upper and lower bound.*

As a next step, we discuss the application of Definition 4.5 to general first-order lineage formulas which do not necessarily adhere any normal form.

Proposition 4.4. *By first exhaustively applying De Morgan's law of Equation* (4.2) *on a first-order lineage formula* Φ*, we can apply Definition 4.5 to* Φ*, even if* Φ *is not in PDNF. Hence, constructing* ϕ_{up} *and* ϕ_{low} *can be done in* $O(|\Phi|)$*.*

Proof. We can implement De Morgan by traversing the formula once, which thus is in $O(|\Phi|)$. Subsequently, we traverse the formula again and replace all first-order literals by *true* or *false* as devised in Definition 4.5. Observation 4.1 ensures the replacements to be unique for each literal. □

Convergence of Bounds Our last step is to show that, when constructing first-order lineage Φ (see Definition 4.4) for a fixed query answer ϕ resulting from Φ, the probability bounds converge monotonically to the probability of the propositional lineage formula $P(\phi)$ with each SLD step.

Theorem 4.1. *Let* $\Phi_1, \ldots, \Phi_n$ *denote a series of first-order formulas obtained from iteratively grounding a conjunctive query via SLD resolution of Definition 4.4 until we reach the propositional formula* ϕ*. Then, rewriting each* Φ_i *to* $\phi_{i,low}$ *and* $\phi_{i,up}$ *according to Definition 4.5 creates a monotonic series of lower and upper bounds* $P(\phi_{i,low})$*,* $P(\phi_{i,up})$ *for the probability* $P(\phi)$*. That is:*

$$\begin{aligned} 0 \leq P(\phi_{1,low}) \leq \cdots \leq P(\phi_{n,low}) \leq P(\phi) \\ \leq P(\phi_{n,up}) \leq \cdots \leq P(\phi_{1,up}) \leq 1 \end{aligned}$$

Proof. The proof proceeds inductively over the structure of Definition 4.4, where we show that each SLD step preserves the bounds. We assume the observed literal $R(\bar{X})$ to occur positively in the PDNF of Φ_i. The negated version is handled analogously.

1. We have an intensional literal $R(\bar{X})$ which was substituted in Φ_i by the disjunction of deduction rules' bodies, which we call Ψ here, to yield Φ_{i+1}. Because there are only literals and no tuple identifiers in Ψ, Definition 4.5 yields $\psi_{low} \equiv$ *false* and $\psi_{up} \equiv$ *true*. Hence, the bounds of Φ_{i+1} are not altered, which reads as $P(\phi_{i+1,up}) = P(\phi_{i,up})$ and $P(\phi_{i+1,low}) = P(\phi_{i,low})$.

2. As in Definition 4.4, we separate the cases of different variables.

   (a) In this case we consider an extensional literal $R(\bar{X})$ where $Var(\bar{X})$ are query variables. Now, $SLD(R(\bar{X}), \Phi_i)$ delivers a set of formulas. Let Φ_{i+1} be an arbitrary formula in this set. We obtain Φ_{i+1} by replacing $R(\bar{X})$ in Φ_i by a tuple identifier I. Hence, in the DNF of $\phi_{i+1,up}$ we added I to the clauses, whereas in the DNF of $\phi_{i,up}$ we replace $R(\bar{X})$ by *true*. Thus, Lemma 4.1 applies and we have $P(\phi_{i+1,up}) \leq P(\phi_{i,up})$. The reasoning for lower bounds is analogous.

(b) Again, we have an extensional literal $R(\bar{X})$, but all variables $Var(\bar{X})$ are bound by an existential quantifier. As a result, each Φ_{i+1} in the set $SLD(R(\bar{X}, \Phi_i)$ is constructed from Φ_i by the first line of Equation (4.1). Now, the DNF of $\phi_{i,up}$ has clauses where $R(\bar{X})$ was substituted by *true*. Then, in $\phi_{i+1,up}$ each clause featuring a new tuple identifier I can be mapped to one of these clauses in the DNF of $\phi_{i,up}$. Therefore, Lemma 4.1 gives us $P(\phi_{i+1,up}) \leq P(\phi_{i,up})$. Again, lower bounds are handled analogously.

(c) If the variables $\bar{X}$ in the extensional literal $R(\bar{X})$ are universally quantified, then $R(\bar{X})$ in Φ_i is replaced by a conjunction (as given in the second line of Equation (4.1)) to yield Φ_{i+1}. In the DNF of $\phi_{i,up}$, we employed *true* whenever $R(\bar{X})$ occurred. Conversely, in $\phi_{i+1,up}$ we replaced $R(\bar{X})$ by a conjunction of tuple identifiers. The resulting extended clauses of $\phi_{i+1,up}$ can be mapped to a clause of $\phi_{i,up}$, so Lemma 4.1 applies: $P(\phi_{i+1,up}) \leq P(\phi_{i,up})$. The lower bounds are addressed analogously.

3. Here, a literal $R(\bar{X})$ was replaced in Φ_i by *false* to yield Φ_{i+1}. Hence, for lower bounds constructed by Definition 4.5 we have $P(\phi_{i,low}) = P(\phi_{i+1,low})$. For the upper bounds Lemma 4.1 delivers $P(\phi_{i+1,up}) \leq P(\phi_{i,up})$, since the PDNF of Φ_{i+1} has fewer clauses as the PDNF of Φ_i.

4. In the last case, $R(\bar{X})$ is arithmetic and $\bar{X}$ consists of constants only. Now, if $R(\bar{X})$ evaluates to *true*, we have $\phi_{i,up} = \phi_{i+1,up}$ and hence also $P(\phi_{i,up}) = P(\phi_{i+1,up})$. For the lower bound, the DNF of $\phi_{i+1,low}$ can have more clauses than the DNF of $\phi_{i,low}$ and so Lemma 4.1 comes to our rescue again: $P(\phi_{i,low}) \leq P(\phi_{i+1,low})$. Conversely, if $R(\bar{X})$ evaluates to *false*, the reasoning for the upper and lower bounds is inverted.

□

4.5 Algorithms

So far, we shed light on the theoretical properties of first-order lineage formulas with respect to constructing them as well as for computing bounds on their answers' probabilities. In this section, we move to the algorithmic perspective. That is, we plug the obtained knowledge together to form a top-k algorithm (Section 4.5.2) which delivers the k answers with highest probabilities. To choose which literal to expand in each step of the top-k procedure, we also present a scheduling algorithm (Section 4.5.1).

4.5.1 Benefit-Oriented Literal Scheduling

We formally covered SLD steps to create first-order lineage formulas in Definition 4.4. However, if there is more than one first-order literal we could expand on, which one do we choose? The resulting scheduling problem is the subject of this section. Our approach does not adhere to any fixed query plan, but chooses the next literal to be expanded in each SLD step dynamically as follows.

In general, it is beneficial to prefer literals, which result in faster convergence of the probability bounds, since the top-k algorithm will rely on this to prune potential query answers more aggressively. For this, recall derivatives on probability computations (see Definition 2.13) and bounds on first-order lineage formulas (see Definition 4.5). We combine these as follows.

Definition 4.6. *Given a literal $R(\bar{X})$ occurring in a first-order lineage formula Φ, let $\Phi_{true} := \Phi[R(\bar{X})/true]$ and $\Phi_{false} := \Phi[R(\bar{X})/false]$. Then, we quantify the impact of $R(\bar{X})$ on the upper probability bound of Φ by:*

$$P(\phi_{true,up}) - P(\phi_{false,up})$$

and analogously for the lower bound:

$$P(\phi_{true,low}) - P(\phi_{false,low})$$

In the above definition, we first exchange the literal $R(\bar{X})$ for *true* and *false* to emulate the derivative. Afterward, we create the propositional formula for the upper or lower bound by substituting all remaining literals.

Example 4.11. *We consider the first-order lineage formula*

$$\Phi := I_1 \wedge ActedIn(P, M) \wedge WonAward(M, A)$$

with $p(I_1) = 0.8$. Then, the impact of the ActedIn *literal on the upper bound is:*

$$P(I_1 \wedge \underbrace{true}_{ActedIn} \wedge \underbrace{true}_{WonAward}) - P(I_1 \wedge \underbrace{false}_{ActedIn} \wedge \underbrace{true}_{WonAward}) = 0.8 - 0$$

In addition to the impact on the probability bounds, we can quantify the number of tuples matching a literal $R(\bar{X})$. Because fewer tuples yield smaller formulas or fewer potential query answers to consider, both ways provide a speed-up for top-k processing. For this, we utilize the groundings of Definition 2.4 and apply them to a single literal only.

Definition 4.7. *For a extensional literal $R(\bar{X})$ and tuples $\mathcal{T}$, we quantify the* selectivity *as the number of tuples matching $R(\bar{X})$ in $\mathcal{T}$:*

$$|G(R(\bar{X}), \mathcal{T})|$$

Unfortunately, in practice obtaining the exact selectivity of a literal $R(\bar{X})$ is nearly as expensive as running the actual database query, which we want to save on. Hence, we approximate $|G(R(\bar{X}), \mathcal{T})|$ by statistics, as it is common in databases [65]. For instance, instead of considering the exact constants in $\bar{X}$, we only keep track of the positions in $\bar{X}$ which hold constants and work with averages representing their expected resulted size. These statistics can be precomputed before running queries.

Example 4.12. *Regarding the* Category *relation of Figure 4.1, we approximate Definition 4.7 by:*

$$|G(\mathit{Category}(\bar{X}), \mathcal{T})| := \begin{cases} 4 & \text{if } \mathit{Var}(\bar{X}) = \emptyset \\ 1 & \text{if the first argument is bound} \\ \frac{4}{3} & \text{if the second argument is bound} \\ 1 & \text{if both arguments are bound} \end{cases}$$

Here, $\frac{4}{3} = \frac{1+1+2}{3}$ results from the fact that there are three constants with a total of four resulting tuples, if we bind the second argument.

Finally, for an extensional literal $R(\bar{X})$ occurring in a lineage formula Φ we combine both impact and selectivity to model the *benefit* of choosing the literal:

$$\mathit{ben}(\Phi, R(\bar{X}), \mathcal{T}) := \frac{|P(\phi_{true,up}) - P(\phi_{false,up})| + |P(\phi_{true,low}) - P(\phi_{false,low})|}{1 + |G(R(\bar{X}), \mathcal{T})|} \tag{4.3}$$

The above formula favors literals which have high impact on the probability bounds of Φ and low selectivity. Based on this formula, we prioritize positive extensional literals among each other.

Now, we are ready to present Algorithm 5 which performs the overall scheduling. In detail, whenever we find an arithmetic literal featuring only

Algorithm 5 Scheduling($A, \mathcal{T}$)

```
Input: Set of first-order lineage formulas $A$, and tuples $\mathcal{T}$
Output: Literal $R(\bar{X})$ to expand next and formula $\Phi$ it occurs in
1: $L := \langle (R(\bar{X}), \Phi) \mid \Phi \in A, R(\bar{X}) \in \Phi \rangle$    ▷ List of all literals with its formula
2: if $L$ contains an arithmetic literal $R(\bar{X})$ with $\mathit{Var}(\bar{X}) = \emptyset$ then
3:     return $(R(\bar{X}), \Phi)$
4: if $L$ contains an intensional literal $R(\bar{X})$ then
5:     return $(R(\bar{X}), \Phi)$
6: if $L$ contains a negated literal $R(\bar{X})$ with $\mathit{Var}(\bar{X}) = \emptyset$ then
7:     return $(R(\bar{X}), \Phi)$
8: Sort positive extensional literals in $L$ by $\mathit{ben}(\Phi, R(\bar{X}), \mathcal{T})$ ▷ See Equation (4.3)
9: return $\mathit{First}(L)$    ▷ Return best positive extensional literal
```

constants as arguments, we process it first, as it can be evaluated. Otherwise, we choose an intensional literal (see Line 5). As there are usually only few deduction rules $\mathcal{D}$, they can be kept in memory, and the resulting SLD step is fast. If there are only extensional literals to consider, we prefer negated literals which have all arguments bound to constants. Finally, we rely on the benefit function of Equation (4.3) to determine the best positive extensional literal. Implementationwise, it is of advantage to keep all extensional literals in a priority queue [25] ordered by the benefit of Equation (4.3).

4.5.2 Top-k with Dynamic Literal Scheduling

Our top-k algorithm primarily operates on the lineage formulas of answer candidates. Specifically, we maintain two disjoint sets of answer candidates A_{top} and A_{cand}, defined as follows. Following the seminal line of threshold algorithms [44], A_{top} comprises the current top-k answers with respect to the lower probability bounds, while A_{cand} consists of all remaining answer candidates. Later, we prune answer candidates from A_{cand} until it is empty.

Definition 4.8. *Given a set of answer candidates $A = \{\Phi_1, \ldots, \Phi_n\}$ represented by their lineage formulas Φ_i, we seek to partition them $A = A_{top} \dot{\cup} A_{cand}$ such that:*

- $|A_{top}| \leq k$
- $\forall \Phi \in A_{top} : P(\phi_{low}) > 0$
- $\forall \Phi \in A_{top}, \forall \Psi \in A_{cand} : P(\phi_{low}) \geq P(\psi_{low})$

As a constraint, the top-k set A_{top} consists only of query answers whose lower bound is greater than 0. This probability must be induced by a purely propositional clause which by the first case of Proposition 4.1 means that these answers must have all query variables bound. The candidate set, on the other hand, may also hold answer candidates with a lower probability bound of 0. These can be answer candidates for which the query variables are not yet bound to constants, hence representing sets of answers. Regarding the implementation of A_{top} and A_{cand}, priority queues keeping the answer candidates ordered by their lower probability bound are very effective, since they support updating the probability bounds efficiently.

Example 4.13. *If we intend to determine the top-1 answer and currently possess the two first-order lineage formulas*

$$I_1 \wedge \mathit{ActedIn}(X, \mathit{ApocalypseNow}) \quad \text{and} \quad I_2 \wedge \mathit{ActedIn}(X, \mathit{Godfather})$$

then both have lower bound 0. Therefore, the two are contained in A_{cand}.

The key to any top-k algorithm is pruning of answer candidates, which we achieve via the following threshold.

$$\theta_{low} := min\ (\{1\} \cup \{P(\phi_{low}) \mid \Phi \in A_{top}\}) \tag{4.4}$$

It captures the lowest lower bound that can be observed among any formula in A_{top}. Based on the θ_{low} we can prune answer candidates from A_{cand}, since they will never be among the top-k answers [44].

Proposition 4.5. *Given answer candidates A_{top} and A_{cand}, and the lower bound θ_{low}, all answer candidates in the following set can never become a top-k answer:*

$$Prune(A_{cand}, \theta_{low}) := \{\Phi \mid \Phi \in A_{cand}, P(\phi_{up}) < \theta_{low}\}$$

Proof. According to Theorem 4.1, the probability bounds of each answer candidate converge monotonically. Hence, with each SLD step the upper bound $P(\phi_{up})$ of Proposition 4.5 monotonically decreases, whereas θ_{low} monotonically increases. This prohibits the answer candidates to leave the set $Prune(A_{cand}, \theta_{low})$ during any future SLD step. □

Example 4.14. *We consider $A_{top} = \{I_1 \wedge I_4\}$, $A_{cand} = \{\text{Directed}(P, \text{Godfather}) \wedge I_5\}$ over the tuples of Figure 4.1 and $k = 1$. Since $p(I_1) = 0.8$, $p(I_4) = 0.6$ we have $P(I_1 \wedge I_4) = 0.48$, which hence is both the upper and lower bound. So, we set $\theta_{low} = 0.48$. The upper bound of* $\text{Directed}(P, \text{Godfather}) \wedge I_5$ *is calculated as* $P(\text{true} \wedge I_5) = 0.3$. *Because of* $0.3 < 0.48$, *we obtain:*

$$Prune(A_{cand}, \theta_{low}) = \{Directed(P,\ Godfather) \wedge I_5\}$$

Finally, we combine all building blocks in Algorithm 6 to form the top-k procedure, which proceeds as follows. First, we initialize the candidate set A_{cand} with the query (see Line 2), which has lower bound 0 and hence cannot be contained in A_{top}. Then, the loop of Line 3 is executed until the candidate set A_{cand} runs out of valid answer candidates, that is, A_{cand} is empty. At each processing step, the scheduler chooses the currently best literal $R_{best}(\bar{X})$ (see Line 4). Following Definition 4.4 we then expand the lineage formula of this literal by performing a single SLD step over both $\mathcal{D}$ and $\mathcal{T}$ (Line 5). Then, we update A_{top}, and A_{cand} (Line 7) due to the following. First, expanding $R_{best}(\bar{X})$ can change the probability bounds of the answer. Second, if there are no matches to $R_{best}(\bar{X})$, neither in $\mathcal{T}$ nor in $\mathcal{D}$, the answer candidates corresponding to $R_{best}(\bar{X})$ may be deleted (because lineage evaluates to *false*). And third, if a query variable was bound to more than one constant, one or more new top-k answer candidates are created. Finally, in Line 9 we prune answer candidates with a too low upper bound, before the next iteration of the loop starts.

Example 4.15. *Assume we query for* $\text{ActedOnly}(X, Z)$ *with $k = 1$ using the deduction rules and tuples of Figure 4.1. So, we first set $A_{top} := \emptyset$ and*

Algorithm 6 Top-$k(\mathcal{T}, \mathcal{D}, Q(\bar{X}), k)$

```
Input: Tuples $\mathcal{T}$, deduction rules $\mathcal{D}$, query $Q(\bar{X})$, and an integer $k$
Output: Top-$k$ answers $A_{top}$ for $Q(\bar{X})$ according to their lower probability bounds
 1: $A_{top} := \emptyset$                                   ▷ Current top-$k$ answers, see Definition 4.8
 2: $A_{cand} := \{Q(\bar{X})\}$                              ▷ Answer candidates, see Definition 4.8
 3: while $A_{cand} \neq \emptyset$ do
 4:     $(R_{best}(\bar{X}), \Phi_{best}) :=$ Scheduling$(A_{top} \cup A_{cand}, \mathcal{T})$      ▷ See Algorithm 5
 5:     $\{\Phi_1, \ldots, \Phi_n\} :=$ SLD$(R_{best}(\bar{X}), \Phi_{best})$                  ▷ See Definition 4.4
 6:     replace $\Phi_{best}$ in $A_{top}$ or $A_{cand}$ by $\Phi_1, \ldots, \Phi_n$
 7:     update $A_{top}$ and $A_{cand}$ to fulfill Definition 4.8
 8:     $\theta_{low} := min\ (\{1\} \cup \{P(\phi_{low}) \mid \Phi \in A_{top}\})$          ▷ See Equation (4.4)
 9:     $A_{cand} := A_{cand} \backslash Prune(A_{cand}, \theta_{low})$              ▷ See Proposition 4.5
10: return $A_{top}$
```

$A_{cand} := \{\text{ActedOnly}(X, Z)\}$. *Since* ActedOnly *is the only literal present, we expand it in Line 5 via its deduction rule, such that we receive in Line 7* $A_{cand} = \{\text{ActedIn}(X, Z) \wedge \neg\text{Directed}(X, Z)\}$. *As* A_{top} *is empty, we obtain* $\theta_{low} = 1$. *Hence, in Line 9 no pruning takes place. In the next iteration, the scheduler chooses* ActedIn*, because* Directed *is negated. Now, Line 5 delivers:*

$$\left\{ \begin{array}{c} I_4 \wedge \neg Directed(Brando, ApocalypseNow), \\ I_5 \wedge \neg Directed(Pacino, Godfather), \\ I_6 \wedge \neg Directed(Tarantino, PulpFiction) \end{array} \right\}$$

Because all their lower bounds are 0 (replace their Directed *literal for* true*), we place them in* A_{cand} *in Line 7. Again, no answer candidate is pruned, so we enter the following iteration. Now, we expand* Directed(Brando, ApocalypseNow)*, where no tuple is found, which results in* $I_4 \wedge \neg false \equiv I_4$. *As* $p(I_4) = 0.6 > 0$, *we receive* $A_{top} = \{I_4\}$ *and*

$$A_{cand} = \left\{ \begin{array}{c} I_5 \wedge \neg Directed(Pacino, Godfather), \\ I_6 \wedge \neg Directed(Tarantino, PulpFiction) \end{array} \right\}$$

Finally, since the lower bound of I_4 *is* 0.6 *and the two answer candidates above have upper bounds of* 0.3 *and* 0.4 *(replacing their* Directed *literal by* false*), we prune all of* A_{cand}*, which terminates the algorithm.*

Final Result Ranking Many applications that employ top-k queries require a complete ranking of the top-k answers. When Algorithm 6 terminates, the probabilities of the top-k answers may however not be known exactly—but only the bounds thereof. Similarly to the strategies in [108], we can tackle this either by iteratively running top-1,..., top-k queries, where an inspection of the k answer sets yields the desired ranking, or by continuing the grounding and decomposition steps until the probability bounds of the top-k answers do not overlap anymore.

4.6 Extensions

We continue with the presentation of extensions to the top-k algorithm being sorted input relations (Section 4.6.1), support for recursive deduction rules (Section 4.6.2), considerations on temporal data (Section 4.6.3), and finally top-k queries with constraints (Section 4.6.4).

4.6.1 Sorted Input Relations

A powerful technique in top-k approaches for extensional data [44, 69, 88] is to store each relation in decreasing order of local ranks, i.e. the tuple probability in our case. Then, the rank at the current scan position serves as an upper bound for the ranks of all remaining tuples. Along with a monotonic score aggregation function, this allows for the computation of monotonically decreasing upper bounds of answer candidates. In our setting, we rank query answers by their probabilities which generally does not resolve to such a monotonic form of score aggregation. However, in the following we characterize a case where decreasing upper bounds, i.e. not relying on *true* as upper bound from Definition 4.5 anymore, are allowed.

Definition 4.9. *Let $R(\bar{X})$ be an extensional literal occurring in Φ such that all variables in $Var(\bar{X})$ are query variables. Then, we can replace case 2(a) of Definition 4.4 by*

$$SLD(R(\bar{X}), \Phi) := \{\sigma_{\bar{a}}(\Phi)\} \cup \underbrace{\{\Phi \wedge \neg \bigwedge_{X_i \in Var(\bar{X})} X_i = a_i\}}_{(*)}$$

where

1. *$p(R(\bar{a}))$ is maximal among the matching tuples;*
2. *$p(R(\bar{a}))$ is the new upper bound for $R(\bar{X})$ in $(*)$.*

In other words, we process the tuple $R(\bar{a})$ in $\sigma_{\bar{a}}(\Phi)$ and capture all remaining tuples (but not $R(\bar{a})$) in $(*)$ with decreased upper bound for $R(\bar{X})$. Efficiently obtaining the maximal $p(R(\bar{a}))$ can be achieved by keeping $\mathcal{R}$ sorted by decreasing tuple probabilities.

Example 4.16. *In the simplest case a query is equivalent to a single extensional literal with all variables being query variables. Assume our query is* WonAward(M, A) *over the tuples of Figure 4.1 with $k = 1$. When we perform one SLD step following Definition 4.9, we gain:*

$$\{I_{10}\} \cup \underbrace{\{\mathit{WonAward}(M, A) \wedge \neg(M = \mathit{PulpFiction} \wedge A = \mathit{BestPicture})\}}_{(*)}$$

Now $p(I_{10})$ is the lower bound for I_{10} and also the upper bound for all answers in (*), which we obtain by setting the equalities to false *and* WonAward *to probability* 0.9*. Therefore, the set* Prune *of Proposition 4.5 contains* (*)*. So, we can stop and return I_{10} as answer.*

The key for Definition 4.9 is that binding a query variable yields a new query answer (see case 2(a) of Definition 4.4), while binding quantified variables can result in a disjunction in the lineage formula due to case 2(b) of Definition 4.4. This disjunction may result in a higher probability than that of the individual input tuples due to Equation (2.4). For example, if two independent tuples I_1, I_2 with a probability of 0.5 each match a single literal with existentially bound variables, then we obtain $1 - (1 - 0.5) \cdot (1 - 0.5) = 0.75 > 0.5$. Thus, an upper bound of 0.5 would be incorrect.

4.6.2 Recursion

In this subsection, we develop an extension for handling deduction rules with recursively defined intensional relations. To ensure a safe semantics for the SLD grounding steps, we require the set of recursive deduction rules $\mathcal{D}$ to be *stratifiable* [2]. That is, it is not allowed to deduce a tuple from its own negation. Stratifiability is a pure syntactic check on the rule structure and can be done prior to query processing. We remark that the combined complexity (in terms of the size of both the data and the deduction rules) for Datalog programs with a single, recursive, non-linear rule is known to be EXPTIME-complete [57]. Although we cannot improve upon this worst-case bound, we argue that top-k pruning may also help to improve the runtime for many recursive queries in practice.

Example 4.17. *If we want to obtain all predecessors of a politician P_1 we can write:*

$$Predecessors(P_1, P_2) \leftarrow \exists P_3\ HasPredecessor(P_1, P_3) \wedge Predecessors(P_3, P_2)$$

The above rule is recursive, since its head literal and the second literal in the body have the same intensional relation.

Recursion poses a challenging problem for any grounding algorithm. Conceptually, the lineage formula of an answer could grow infinitely large if a cycle arises within the deduction rules. To define cycles formally, let $SLD^+(R(\bar{X}), \Phi)$ denote the repeated application of the first case of Definition 4.4 which resolves intensional literals for their deduction rules.

Definition 4.10. *Given a first-order lineage formula Φ containing $R(\bar{X})$, we have a* cycle*, if $R(\bar{X})$ occurs in Ψ where $\Psi \in SLD^+(R(\bar{X}), \Phi)$.*

Hence, $R(\bar{X})$ forms a cycle if the resolving of it via deduction rules yields $R(\bar{X})$ again.

Example 4.18. *If we expand* Predecessors(P_1, P_2) *via the deduction rule of Example 4.17, we have*

$$\exists P_3\ \mathit{HasPredecessor}(P_1, P_3) \wedge \mathit{Predecessors}(P_3, P_2)$$

which features the Predecessors *literal again, and hence forms a cycle.*

Next, we develop a lemma ensuring the finiteness of a first-order lineage formula Φ, without altering the possible worlds that satisfy Φ.

Lemma 4.2. *On a set of recursive deduction rules let a first-order lineage formula Φ which contains the literal $R(\bar{X})$ be given. Then, for $\Psi \in SLD^+(R(\bar{X}), \Phi)$ and $\Psi' \in SLD^+(R(\bar{X}), SLD^+(R(\bar{X}), \Phi))$, it holds:*

$$R(\bar{X}) \textit{ occurs in } \Psi \textit{ and } \Psi' \quad \Rightarrow \quad \Psi \equiv \Psi'$$

In other words, expanding a cycle more than once does not change the validity of a lineage formula, which agrees with earlier results in the context of probabilistic extensions to Datalog [122].

Proof. Without loss of generality, we assume all formulas to be in prenex form. Furthermore, let $\Phi' \vee \bigvee_i (\Phi''_i \wedge R(\bar{X}))$ be the disjunctive normal form of Ψ. That is Φ' is a formula in disjunctive normal form, Φ''_i are conjunctions of literals and both do not contain $R(\bar{X})$. Due to stratification, $R(\bar{X})$ must occur positively in the above formula. Now, we can rewrite Ψ' through the following series of algebraic transformations:

$$\Psi' \equiv \Phi' \vee \bigvee_i (\Phi''_i \wedge (\Phi' \vee \bigvee_j (\Phi''_j \wedge R(\bar{X}))))$$

Here, Ψ' was expanded twice via the deduction rules. Now, we apply the distributive law:

$$\Phi' \vee \bigvee_i (\Phi''_i \wedge \Phi') \vee \bigvee_{i,j} (\Phi''_i \wedge \Phi''_j \wedge R(\bar{X}))$$

Then, each $(\Phi''_i \wedge \Phi')$ is subsumed by Φ':

$$\Phi' \vee \bigvee_{i,j} (\Phi''_i \wedge \Phi''_j \wedge R(\bar{X}))$$

Next, we separate the cases, when $i = j$ and $i \neq j$:

$$\Phi' \vee \bigvee_i (\Phi''_i \wedge \Phi''_i \wedge R(\bar{X})) \vee \bigvee_{i \neq j} (\Phi''_i \wedge \Phi''_j \wedge R(\bar{X}))$$

Since it holds that $\Phi_i'' \wedge \Phi_i'' \equiv \Phi_i''$, we write:

$$\Phi' \vee \bigvee_i (\Phi_i'' \wedge R(\bar{X})) \vee \bigvee_{i \neq j} (\Phi_i'' \wedge \Phi_j'' \wedge R(\bar{X}))$$

Now, $\Phi_i'' \wedge R(\bar{X})$ subsumes $\Phi_i'' \wedge \Phi_j'' \wedge R(\bar{X})$:

$$\Phi' \vee \bigvee_i (\Phi_i'' \wedge R(\bar{X})) \equiv \Psi$$

This again yields the form of the first expansion of $R(\bar{X})$. □

We extend Algorithm 5 to handle recursive deduction rules as follows. If a literal occurs in a cycle, we do not schedule it. The cycle can be broken by new constants binding a variable, and hence enabling scheduling the literal again. Furthermore, we add a new case to Definition 4.4: if a first-order lineage formula contains only literals in cycles, no new results can be obtained, so we replace all these literals by *false*.

4.6.3 Temporal Data

In principle, we can apply the top-k procedure to temporal data as follows. Given a temporal query $Q(\bar{X})$ and temporal deduction rules $\mathcal{D}$, we transform the deduction rules following the procedure of the proof of Theorem 3.1 to yield $\mathcal{D}'$. Afterwards, deduplication (see Definition 3.5) is encoded in $\mathcal{D}'$. Also, all mentions of time in $\mathcal{D}'$ are expressed by integer comparisons. Hence, executing Algorithm 6 on $\mathcal{D}'$ enables support for temporal data. As the proof of Theorem 3.1 duplicates all rules and data, this approach is feasible, but not very efficient.

4.6.4 Constraints

Let us extend the top-k algorithm to support constraints along with queries $Q(\bar{X})$ and their deduction rules $\mathcal{D}_q$.

Fixed Constraints If the constraints are fixed to the propositional lineage formula ϕ_c (see Section 2.2.6), then we invoke Algorithm 6 by:

$$\text{Top-k}(\mathcal{T}, \mathcal{D}_q, Q(\bar{X}) \wedge \phi_c, k)$$

This produces the correctly ranked query answers according to the probability defined in Equation (2.7). The reason is that $Q(\bar{X}) \wedge \phi_c$ yields the enumerator of Equation (2.7). Additionally, the denominator of Equation (2.7) is a constant which is shared by all query answers, and hence does not influence the ranking. Nevertheless, we have to stress that the probability bounds are correct for $Q(\bar{X}) \wedge \phi_c$, but do not necessarily bind the true probability of the query answers anymore, since we are neglecting the denominator.

Variable Constraints In this case, the constraints are altered along with the queries. Let the constraints be given by the set of literals $\mathcal{C}_p$ (see Section 2.2.6) with their deduction rules $\mathcal{D}_c$. We then start Algorithm 6 by

$$\text{Top-k}\left(\mathcal{T}, \mathcal{D}_q \cup \mathcal{D}_c, Q(\bar{X}) \wedge \forall \bar{Y}' \bigwedge_{C(\bar{Y}) \in \mathcal{C}_p} C(\bar{Y}) \ , k\right)$$

where $Var(\bar{Y}') = \bigcup_{C(\bar{Y}) \in \mathcal{C}_p} Var(\bar{Y})$. Here, $\forall \bar{Y}' \bigwedge_{C(\bar{Y}) \in \mathcal{C}_p} C(\bar{Y})$ will be expanded by case 2(d) of Definition 4.4 into the conjunction of all grounded constraints, hence delivering once again the enumerator of Equation (2.7). As in the case of fixed constraints, the ranking is correct and hence the correct answers are computed. Still, the bounds might not capture the true probabilities of the answers anymore.

Example 4.19. *We run a top-k query with constraints relying on the database $\mathcal{T}$ and deduction rules $\mathcal{D}_q$ from Figure 4.1. Let the query be* Acted-Only(X, Z) *where the constraints are given by* Constraints(D, M). *Furthermore, we define $\mathcal{D}_c$, that is the constraints, to comprise the deduction rule:*

$$Constraints(D, M) \leftarrow Directed(D, M) \wedge Category(M, Crime)$$

It expresses that we fully trust into the category and director of crime movies. Hence, we call Algorithm 6 by:

$$\textit{Top-k}(\mathcal{T}, \mathcal{D}_q \cup \mathcal{D}_c, ActedOnly(X, Z) \wedge \forall D \ \forall M \ \ Constraints(D, M), k)$$

If we apply the first case of Definition 4.4 to replace the literals in the above query by their deduction rules we obtain:

$$\begin{gathered} ActedIn(X, Z) \wedge \neg Directed(X, Z) \wedge \\ \forall D \ \forall M \ \ Directed(D, M) \wedge Category(M, Crime) \end{gathered}$$

In the next steps, we exchange the literals in the lower line for their tuple identifiers by repeatedly applying case 2(c) of Definition 4.4.

$$ActedIn(X, Z) \wedge \neg Directed(X, Z) \wedge \underbrace{(I_2 \wedge I_{12}) \wedge (I_3 \wedge I_{13})}_{(*)}$$

Then, each query answer as distinguished by X and Z has the conjunction of all constraints $(*)$ attached. This coincides with the enumerator of Equation (2.7).

4.7 Experiments

The empirical evaluation of the presented top-k algorithm is fourfold. We investigate the runtimes on established query classes in probabilistic databases (Section 4.7.1), followed by queries which we consider to characterize well the strengths and weaknesses of our algorithm (Section 4.7.2), before we experiment with recursive queries (Section 4.7.3), and then conclude with an internals analysis, such as our scheduling algorithm (Section 4.7.4).

Competitors We employ two well-known probabilistic database engines for comparison purposes, *MayBMS*[1] [10] and *Trio*[2] [14], both computing all answers and their probabilities. Additionally, we implemented the multi-simulation algorithm of [115], which we refer to as *MultiSim*. We also include comparisons against a purely deterministic database, denoted as *PostgreSQL*, by storing probability values in all relations but omitting the actual probability computations. The *PostgreSQL* baseline thus serves also as a lower bound for any probabilistic top-k approach, including [108, 115], that requires full data materialization (or full lineage tracing for intensional query evaluations). Finally, we refer to our top-k implementation as *ProbTop-k*.

4.7.1 Query Classes

We first focus on four established query classes in probabilistic databases, which are performed on the IMDB dataset (see Appendix A.2.2) comprising knowledge about movies. The tuple probabilities are synthetically created by drawing them from the uniform distribution.

Queries Here, we present four query patterns which are instantiated by constants to deliver 1000 queries each. The first query class $Q1$ is *non-repeating hierarchical* [31, 139], which may use every relation symbol at most once. For this reason, probability computations are efficient, i.e. fully captured by Equation (2.4). We instantiate $\mathit{Query}(P, \mathit{Constant})$ to 1000 queries over the deduction rules:

$$\begin{aligned}
\mathit{Query}(P, M) &\leftarrow \mathit{ActedIn}(P, M)\\
\mathit{Query}(P, M) &\leftarrow \mathit{Subquery}(P, M)\\
\mathit{Subquery}(P, M_1) &\leftarrow \exists M_2\ \mathit{Edited}(P, M_1) \wedge \mathit{Directed}(P, M_2)\\
\mathit{Subquery}(P, M_1) &\leftarrow \exists M_2\ \mathit{Produced}(P, M_1) \wedge \mathit{Written}(P, M_2)
\end{aligned}$$

In $Q2$, we focus on *repeating hierarchical* queries [139], where we ask for $\mathit{Query}(P, \mathit{Constant})$ defined as follows:

$$\begin{aligned}
\mathit{Query}(P, M) &\leftarrow \mathit{ActedIn}(P, M) \wedge \mathit{HasCategory}(P, \mathit{Action})\\
\mathit{Query}(P, M) &\leftarrow \begin{pmatrix} \mathit{Produced}(P, M) \wedge \mathit{HasCategory}(P, \mathit{Action}) \\ \wedge \mathit{HasCategory}(P, \mathit{Drama}) \end{pmatrix}\\
\mathit{Query}(P, M) &\leftarrow \mathit{Written}(P, M) \wedge \mathit{HasCategory}(P, \mathit{Drama})
\end{aligned}$$

The resulting lineage formulas generally require Shannon expansions, but at most two of them. Then, we work on *non-repeating head-hierarchical*

[1] `http://maybms.sourceforge.net/`

[2] `http://infolab.stanford.edu/trio/`

queries [108], whose probabilities are known to be computable in polynomial time for ranking, but not for fully computing the probability of each answer. The query pattern $Q3$ hence has a subquery *Expensive* which may not be evaluated for ranking, when we run $\mathit{Query}(\mathit{Constant}, M)$ over:

$$\begin{aligned}
\mathit{Query}(P, M) &\leftarrow \mathit{Edited}(P, M)\\
\mathit{Query}(P, M) &\leftarrow \mathit{Produced}(P, M)\\
\mathit{Query}(P, M) &\leftarrow \exists M_2\ \exists C\ \mathit{Directed}(P, M) \wedge \mathit{Expensive}(M_2, C)\\
\mathit{Expensive}(M_1, C) &\leftarrow \exists P\ \exists M_2 \begin{pmatrix} \mathit{ActedIn}(P, M_1) \wedge \mathit{Written}(P, M_2)\\ \wedge \mathit{HasCategory}(M_2, C) \end{pmatrix}
\end{aligned}$$

Finally, general *unsafe* queries [31, 139], as used in our query pattern $Q4$, can produce very large lineage formulas with many Shannon expansions:

$$\begin{aligned}
\mathit{Query}(P, M) &\leftarrow \exists M_2\ \exists P_2\ \exists M_3\ \exists C \begin{pmatrix} \mathit{Edited}(P, M_2) \wedge \mathit{Produced}(P_2, M_2)\\ \wedge \mathit{Written}(P_2, M)\\ \wedge \mathit{HasCategory}(M_3, C) \end{pmatrix}\\
\mathit{Query}(P, M) &\leftarrow \mathit{Directed}(P, M)\\
\mathit{Query}(P, M) &\leftarrow \mathit{ActedIn}(P, M)
\end{aligned}$$

Here, we execute the query $\mathit{Query}(\mathit{Constant}, M)$. Additionally, we note that sorted input lists can be employed for the latter two deduction rules.

Results We measure the average runtime over the 1000 queries and show them in Figure 4.2. Moreover, the table in Figure 4.3 depicts the fraction of database tuples our top-k approach reads in comparison to the number of database tuples necessary for computing all answers. For presentation purposes, we depict runtimes of only up to 100 seconds for all systems.

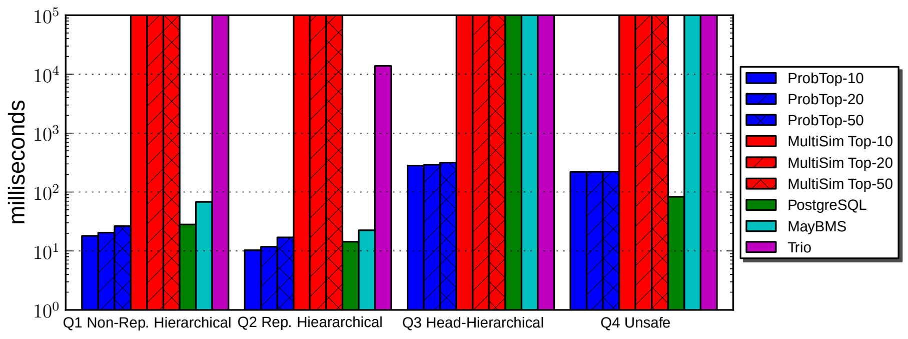


Figure 4.2: Query Classes Experiments

Discussion For the non-repeating hierarchical queries ($Q1$), our top-k approach outperforms all systems including the deterministic one. We mainly

	ProbTop-10	ProbTop-20	ProbTop-50
$Q1$	32.8%	43.8%	76.3%
$Q2$	17.5%	28.5%	60.6%
$Q3$	0.001%	0.001%	0.001%
$Q4$	0.2%	0.2%	0.2%
$Q5$	14.6%	23.2%	48.4%
$Q6$	21.9%	35.1%	74.8%
$Q7$	8.7%	14.1%	30.0%
$Q8$	23.7%	36.9%	74.9%

Figure 4.3: Percentage of Scanned Database Tuples

benefit, because by far not all database tuples need to be scanned, and probabilities, and hence bounds, can be computed efficiently, i.e. by Equation (2.4). $Q2$ contains repeated relations and the gains in data computations are partially diminished by the Shannon expansions needed for computing the bounds. $Q3$ includes expensive data computations caused by a subquery that is shared among all answers. Since the subquery is not required to rank the results, our approach reads only very few tuples (Figure 4.3). Here, our top-k algorithm successfully terminates and even outperforms the deterministic *PostgreSQL* baseline. $Q4$ has a subquery with both expensive probability computations and major data computations. However, we can prune answers even before the expensive subqueries are fully evaluated. *PostgreSQL* exhibits a low runtime for the unsafe query $Q4$. In all queries, *Trio* and *MultiSim* show an at least 100 times slower performance. For *MultiSim*, the majority of the runtime is spent in sampling (except for $Q3$). The runtime does not significantly increase with k, but rather depends on the distance of the k-th and $k+1$-th answers' probabilities. The smaller the distance, the longer it takes to run *MultiSim*.

4.7.2 Performance Factors

We next highlight the different factors that impact how our top-k approach performs against the competitors. For this, we create for four additional query patterns $Q5$, $Q6$, $Q7$, and $Q8$, which are again instantiated into $1,000$ queries each, and their average runtime is depicted in Figure 4.4. As previously, we run on the IMDB dataset (see Appendix A.2.2) with synthetic uniformly drawn probabilities.

The first query pattern $\mathit{Query}(M, \mathit{Constant})$, which we refer to as $Q5$, allows exactly one proof for each answer candidate:

$$\mathit{Query}(M_1, M_2) \leftarrow \exists P\ \mathit{ActedIn}(P, M_1) \wedge \mathit{Directed}(P, M_2)$$

Hence, pruning in terms of omitting a proof is impossible for our system. Also, the single proof involves an existentially quantified variable $\exists P$

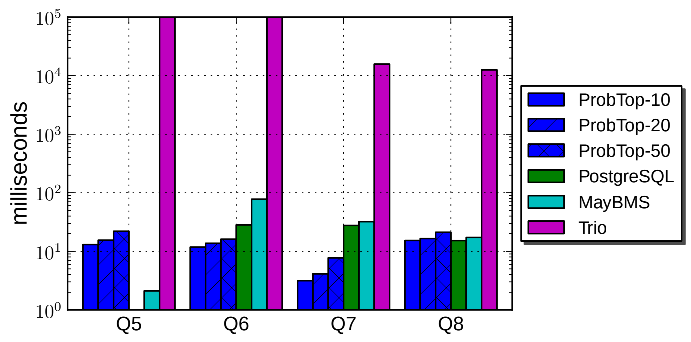


Figure 4.4: Performance Factors Experiments

which limits the use of sorted input lists. Inspecting Figure 4.4, *MayBMS*' bottom-up grounding and probability computation of all answers is much more efficient, since our system computes most answers. In $Q6$, we run $Query(P, Constant)$ with the deduction rules:

$$
\begin{aligned}
&Query(P, M) \leftarrow Written(P, M) \wedge Directed(P, M)\\
&Query(P, M) \leftarrow ActedIn(P, M) \wedge HasCategory(M, Action)\\
&Query(P, M) \leftarrow Produced(P, M)
\end{aligned}
$$

Here, the possibility of three proofs per answer enables pruning and the lack of existential quantifiers puts our approach in favor of the others. In the next query pattern $Q7$, the query $Query(P, Constant, M)$ is a join of two existential relations:

$$Query(P, M_1, M_2) \leftarrow Produced(P, M_1) \wedge Written(P, M_2)$$

This, is the prime target of sorted input lists which hence provided a significant performance gain for our approach. Finally, in $Q8$ we execute $Query(P, Constant)$ which is defined by:

$$
\begin{aligned}
&Query(P, M) \leftarrow ActedIn(P, M) \wedge HasCategory(M, Action)\\
&Query(P, M) \leftarrow Produced(P, M) \wedge HasCategory(M, Action)\\
&Query(P, M) \leftarrow \exists C\ Produced(P, M) \wedge HasCategory(M, C) \wedge C \neq Action
\end{aligned}
$$

Here, each answer has up to three proofs, however the joined relations overlap and hence require Shannon expansions. Since we repeatedly invoke these expansions to determine the bounds, the advantages of top-k pruning even out with the competitors.

4.7.3 Recursion

We investigate how our top-k approach performs for recursive deduction rules over the YAGO knowledge base (see Appendix A.2.1). The recursive query patterns are instantiated to 50 queries each. First, $Q12$ attempts to collect all probability mass from the children

$$\begin{aligned}&\mathit{Ancestor}(P_1, P_2) \leftarrow \mathit{HasChild}(P_1, P_2)\\&\mathit{Ancestor}(P_1, P_2) \leftarrow \exists P_3\ \mathit{HasChild}(P_1, P_2) \wedge \mathit{Ancestor}(P_2, P_3)\end{aligned}$$

where we query for $\mathit{Ancestor}(\mathit{Constant}, P)$. The second query pattern $Q13$ is given by the query $\mathit{Politician}(P, \mathit{Constant})$, and asks for politicians of nations by transitively following the *HasPredecessor* relation:

$$\begin{aligned}&\mathit{Politician}(P, N) \leftarrow \mathit{PoliticianOf}(P, N)\\&\mathit{Politician}(P, N) \leftarrow \exists P_2\ \mathit{HasPredecessor}(P, P_2) \wedge \mathit{Politician}(P_2, N)\end{aligned}$$

In Figure 4.5 we display the runtimes averaged runtimes over the 50 queries for our top-k algorithm along with the *FullGrounding* approach. The latter corresponds to an SLD grounding algorithm which computes all answers with their lineage, but does not perform any probability computations. For

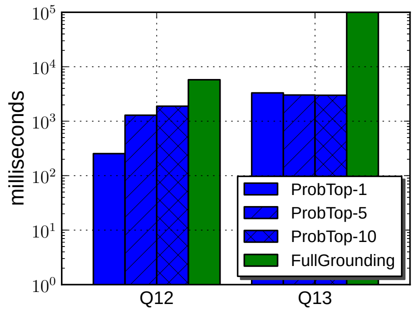


Figure 4.5: Recursion Experiments

$Q12$, the runtime increases with k, since more children being generations away from the queried person have to be computed. For $Q13$ our top-k algorithm takes the same amount of time for all k's, because more than 10 politicians are known per country and are ranked in the top-10 results. For the full grounding, the lineage computation of all answers becomes very expensive.

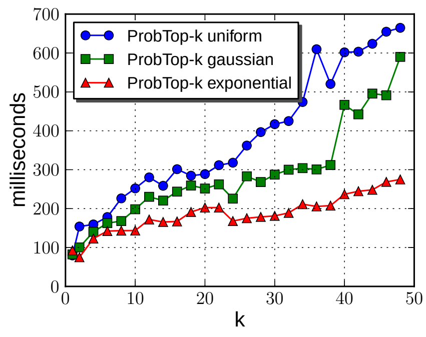
4.7.4 Algorithm Analysis

In this section we first evaluate our algorithms with respect to the impact of different probability distributions in the tuples. Then, we determine the break-even point between running a top-k query versus computing all answers in a bottom-up manner. Finally, we compare different scheduling strategies.

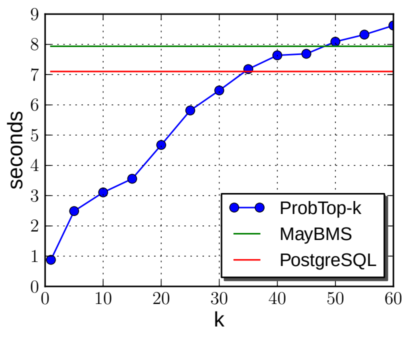
Probability Distributions So far, we focused on a uniform distribution of tuple probabilities. Instead, we now explore how different distributions can affect our performance by comparing uniform, Gaussian, and exponential distributions. Figure 4.6(a) shows the results for a join query $Query(M, P, M)$ on two existential relations over the IMDB dataset:

$$Query(M, P, M) \leftarrow HasCategory(M, \mathit{Talk\text{-}Show}) \wedge Produced(P, M)$$

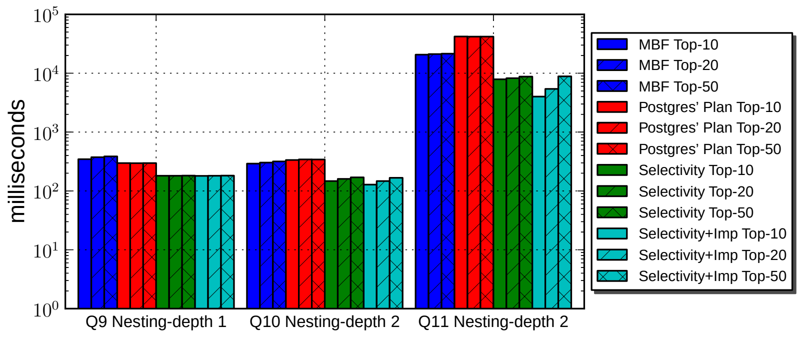
As k grows, the uniform distribution yields the highest increase in runtime while the exponential distribution has the slowest grow, as only few tuples have high probabilities. The Gaussian distribution shows a jump at $k = 40$ as more answer candidates with similar probabilities are found.



(a) Probability Distributions



(b) Break-Even Point



(c) Scheduling

Figure 4.6: Algorithm Analysis Experiments

Break-Even Point In this experiment, we compare our top-k approach against a full materialization of all answers, i.e. investigating for which values of k it is beneficial to compute all answers rather than running our top-k procedure. The query $Query(P, M)$ asks for directors of comedies:

$$Query(P, M) \leftarrow Directed(P, M) \wedge HasCategory(M, Comedy)$$

It is again performed on the IMDB dataset (see Appendix A.2.2) with synthetic uniform probabilities. Our top-k system computes the top-ranked answers for different ks. In contrast, *MayBMS* and *PostgreSQL* fully materialize a table containing all results, where *PostgreSQL* ignores the probability computation. Inspecting Figure 4.6(b), we notice that for $k < 50$, our top-k approach outperforms the others, but for larger values, the bookkeeping overhead starts dominating and it is more efficient to compute all answers by a full join.

Scheduling Finally, we evaluate our scheduling techniques based on *selectivity* estimation (see Definition 4.7) and *impact* (see Definition 4.6). Our first baseline for dynamic literal scheduling, called "most-bound-first" (*MBF*) (aka. "bound-is-easier" [2]), chooses the literal with the maximum number of arguments bound at each SLD grounding step. For the second baseline, we obtained PostgreSQL's static query plan for these query patterns and forced our system to adhere to this plan (denoted as *Postgres' Plan*). Using the YAGO dataset (see Appendix A.2.1), the three query patterns $Q9$, $Q10$, and $Q11$ were instantiated by 100 constants each (see Appendix A.3.2). We order the query patterns by increasing nesting depth of their deduction rules, such that $Q9$, $Q10$, and $Q11$ come with nesting depths of 1, 2, and 3, respectively.

In Figure 4.6(c) we display the average runtime over the 100 queries for each setup. For $Q9$, *MBF* is outperformed by both *Postgres' Plan* and our scheduler relying on selectivity estimates (*Selectivity*). Here, adding the impact calculations to the selectivity estimation does not yield any performance gains, but even results in slight losses. However, when moving to the higher nesting depths of $Q10$ and $Q11$, the impact calculations start improving the performance of the selectivity and impact based scheduler. The reason is that the first-order lineage formulas get more complicated, and hence the impact for the different literals varies more.

4.8 Summary and Outlook

Contribution We presented efficient processing strategies for top-k queries over tuple-independent probabilistic databases. Our approach does neither assume safe query plans nor read-once lineage formulas, and it is able to return the exact top-k answers according to their probabilities in many cases even when exact probability computations for these answers are difficult. To capture intermediate states in top-k processing, we introduced first-order lineage formulas, which represent sets of answer candidates. Based on theoretically derived probability bounds of first-order lineage formulas, we are able to prune (sets of) answer candidates without fully grounding the lineage formula. Hence, this is the first top-k work in the context of probabilistic databases which can save on the data computation step. This is an often neglected issue when querying a probabilistic database. Extensions of our framework allow us to adopt sequential access patterns known from managing extensional data, querying under constraints, and they even help to improve the runtime for recursive rules.

Future Directions Besides sorted input lists, there are other promising cases in which the probability bounds on a first-order literal could be tightened. These can arise from constraints, i.e. if a relation is functional, or from correlations in the database tuples, e.g. block-disjoint probabilistic databases can rule out the co-occurrence of two literals. Moreover, since probabilities are computed repeatedly in order to obtain bounds, this step can form a bottleneck in our approach. One solution is to investigate rearranging the structure of the first-order logical formulas such that less Shannon expansions are caused. Another interesting approach is to approximate the probability bounds to speed up the probability computations of the bounds.

Chapter 5

Learning Tuple Probabilities

5.1 Introduction

Most works in probabilistic databases assume the data along with its probabilities to be given as an input. Also the preceding chapters of this work followed this line. Nevertheless, when creating, updating or cleaning a probabilistic database, the tuple probabilities have to be altered or even newly created, that is learned. This is the subject of the present chapter.

Applications Consider a user who *labels* a *query answer* to be correct. This additional information should be incorporated into the probabilistic database by updating the probabilities stored therein. Also, when we design *consistency constraints* for the probabilistic database, these constraints rule out specific possibilities. Since this reduces uncertainty, the probabilities of the data have to be altered. Moreover, on top of the probabilistic database an automated process might be running. This process can hand back *large amounts of feedback* to the database, maybe in the form of confidence values. Furthermore, if we wish to turn an *incomplete database* into a probabilistic database, we have to learn probability values for each of the possible completions. Finally, the process of updating the probabilities of the tuples can be employed to *clean* a given probabilistic database. The reason is when we add feedback or constraints to a probabilistic database the uncertainty degrades. This can yield tuples which certainly exist or are invalid at all, as in conventional database systems.

Existing Approaches Although this has been stated as a challenge already in [27], to this date, there are only few works [83, 137] which aim for creating or updating probabilistic databases. The authors of [137] derive a probabilistic database from an incomplete database by estimating probabilities from the complete part of the incomplete database. In Section 5.7.5, we devise how to emulate their approach in our setting. In [83] *condition-*

ing a probabilistic database onto a given set of consistency constraints is introduced. These constraints are logical formulas over the probabilistic database which must either be true or false. This thesis heavily relies on their methodology (see Section 2.2.6). In Section 5.7.3, we discuss the relationship between learning and conditioning in detail.

Our Approach We integrate feedback into a probabilistic database as follows. As feedback we consider a set of *propositional lineage formulas* over a probabilistic database, where the formulas are *labeled by target probabilities.* Each propositional lineage formula of the feedback can represent a query answer or a consistency constraint, for instance. We then *learn* the *probability values* associated with the *tuples* in this probabilistic database. The tuples' probabilities are set such that the probabilities of the lineage formulas again yield the given probability labels. This problem can be seen as the inverse problem to probability computations in probabilistic databases, where tuple probabilities are known, and then produce probabilities of lineage formulas.

Problem Statement In short, as *input* we are given:

- a tuple-independent probabilistic database, where some or all *tuple probabilities are missing*;
- *propositional lineage formulas* over the database each of which is labeled by a *target probability.*

Then, our approach delivers as *output*:

- *new tuple probability values* such that the probability of the lineage formulas meets the given target probability.

The resulting probabilistic database instance encodes the additional information as provided by the labeled lineage formulas in the tuple probabilities. We allow probability labels for lineage formulas, rather than the special case of Boolean labels, because they occur in practice (see Section 5.9.1). Next, we illustrate our setting by the following running example.

Example 5.1. *Our running example resembles the information-extraction setting of Section 2.5 in which we employ a set of textual patterns to extract facts from various Web domains. However, instead of knowing all probabilities of all tuples the respective values in the* UsingPattern *and* FromDomain *relations are missing as indicated by the question marks in Figure 5.1. We thus are unsure about the reliability—or trustworthiness—of the textual patterns and the Web domains that led to the extraction of our remaining facts, respectively. Grounding the deduction rules of Equation* (2.8) *and Equation* (2.9) *against the database tuples of Figure 5.1 yields*

WonPrizeExtraction

	Subject	Object	Pid	Did	p
I_1	*Spielberg*	*AcademyAward*	1	1	0.6
I_2	*Spielberg*	*AcademyAward*	2	1	0.3

BornInExtraction

	Subject	Object	Pid	Did	p
I_3	*Spielberg*	*Cinncinati*	3	1	0.7
I_4	*Spielberg*	*LosAngeles*	3	2	0.4

UsingPattern

	Pid	Pattern	p
I_5	1	*Received*	?
I_6	2	*Won*	?
I_7	3	*Born*	?

FromDomain

	Did	Domain	p
I_8	1	*Wikipedia.org*	?
I_9	2	*Imdb.com*	?

Figure 5.1: Example Probabilistic Database with Missing Probability Values

the new tuples BornIn(Spielberg, Cinncinati)*,* BornIn(Spielberg, LosAngeles)*, and* WonPrize(Spielberg,AcademyAward)*. Figure 5.2 shows these new tuples along with their propositional lineage formulas. A closer look at the*

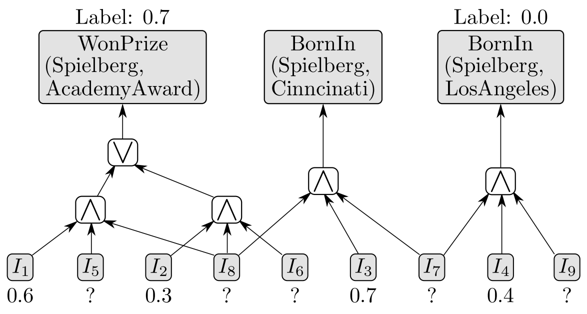


Figure 5.2: Example Lineages and Labels

new tuples reveals, however, that not all of them are correct. For instance, BornIn(Spielberg,LosAngeles) *is wrong, so we label it with the probability of* 0.0*. Moreover,* WonPrize(Spielberg,AcademyAward) *is likely correct, but we are unsure, hence we label it with the probability of* 0.7*, as shown on top of Figure 5.2. Given the probability labels of the query answers, the goal of the learning procedure is to learn the database tuples' unknown probability values for* UsingPattern *and* FromDomain*, such that the lineage formulas again produce the given probability labels. Instead, the probabilities of the tuples of* WonPrizeExtraction *and* BornInExtraction *remain unchanged.*

5.2 Related Work

Many machine learning approaches have been applied to large scale datasets where an overview is provided in [13]. Nonetheless, the scalable methods tend to not offer a declarative language (similar to deduction rules or constraints) in order to induce correlations among tuples, as queries and lineage do in probabilistic databases. Hence, in this section we review approaches which address learning in the presence of correlations induced by logical formulas and, at the same time, have the ability to scale to database like instance sizes.

Learning in Markov Logic Networks In [91, 131] a number of learning algorithms for Markov logic networks are studied, where some of them employ a per-weight learning rate as in this work. Still, the objective function for learning in Markov logic networks (as well as in relational Markov networks [56, 141]) is very different to ours. For each first-order clause they count how many true literals there are in the ground truth. Then, their learning procedure attempts to set the weights such that the expected number of true groundings equals the number of true groundings in the ground truth. In other words, the true groundings after weight learning might be very different from the true ones in the ground truth, but their cardinality will be the same. Nevertheless, in our setting labels are attached directly to propositional lineage formulas hence enforcing the involved tuples to fulfill their specific labels.

Learning in ProbLog The learning procedure of *ProbLog* [63] allows for labels in the form of partial interpretations. These labels are logical formulas over tuples which follow the semantics of tuple-independent probabilistic databases. The authors devise a expectation maximization algorithm for learning the tuple probability values. Still, our stochastic gradient descent implementation is several orders of magnitude faster (see Section 5.9.1 and Section 5.9.2). Within the ProbLog framework, [62] proposes an objective similarly to ours, however lacking both a theoretical analysis and large scale experiments.

Creating Probabilistic Databases There are very few works on the creation of probabilistic databases. The authors of [137] induce a probabilistic database by estimating probabilities from a given incomplete database. We can emulate their approach in our setting, which we discuss in detail in Section 5.7.5. Enforcing consistency constraints by conditioning the database tuples of a probabilistic database [83] onto these constraints allows for altering the tuple probabilities. In comparison to this work, conditioning lacks support for non-Boolean or inconsistent constraints. On the other hand, conditioning supports correlations between tuples by storing world-set-trees, whereas the result of our learning problem are independent tuple probabilities (see Section 5.7.3 for a detailed technical discussion).

Probability Distributions over Databases There are several works [94, 96, 118] which estimate a probability distribution over an entire database, e.g. over all possible worlds. To derive this distribution they often employ the maximum entropy principle which guarantees an unbiased distribution incorporating the evidence. In [96, 118] the methodology is used for selectivity estimation of queries over a database given the selectivities of previous queries. Also, a similar approach is employed in data mining [94] to predict frequencies of item sets in a database. Still, this distribution is expensive to compute and does not take advantage of the independence present in a probabilistic database model. In our setting, we specifically compute a distribution adhering the tuple-independence.

Data Integration Imagine we are given two or more sources of data and attempt to integrate them into one database. The resulting inconsistencies and alternatives can be modeled probabilistically [93], hence creating a probabilistic database. In this setting [150] presented an approach to merge XML documents such that probabilistic XML documents arise. Nevertheless, their way of creating probabilities is rather simple. For example, they employ 0.5 if there are two alternatives. As an extension [80] incorporates user feedback in the information integration process to reduce the uncertainty. Anyhow, they rely on consistent Boolean labels only.

Optimization Methods Our algorithm learning the tuple probabilities is based on stochastic gradient descent, an optimization procedure. Due to the various applications of optimization, there is an entire zoo of gradient-based optimization techniques [103]. Approaches, such as Newton's method, which are based on the Hessian, do not to scale to database-like instance sizes. This disadvantage is circumvented by Quasi-Newton methods, for instance limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) [103], which estimates the Hessian. We compare our approach to L-BFGS and plain gradient descent in the experiments of Section 5.9.2.

Polynomials for Probability Computations The theoretical analysis of our learning problem (Section 5.4) is based on computing probabilities of lineage formulas via polynomials. Similarly, the authors of [59] used semirings over polynomials to model provenance, where lineage is a special case. Likewise, [46] employs arithmetic expressions to compute aggregations over probabilistic databases. However, all of the above works do not consider learning or altering of tuple probabilities. Finally, *Sum-Product*

Networks [113] investigate tractable graphical models by representing these as polynomial expressions with polynomially many terms. Still, in this work polynomials serve as theoretical tool only having no restriction on size.

5.3 Contribution

In this chapter we present the first work to tackle the problem of learning unknown (or missing) tuple probabilities from labeled lineage formulas in the context of probabilistic databases.

- More specifically, in Section 5.5, we formally define the *learning problem* and analyze its properties from a theoretical perspective by characterizing its *computational complexity* as well as the nature of its *solutions.*

- We formulate the learning problem as an *optimization problem*, devise two different objective functions for solving it, and discuss both in Section 5.6.

- In Section 5.8, we present a *learning algorithm* based on stochastic gradient descent which scales to problem instances with hundreds of thousands of labels and millions of tuples to learn the probability values for (Section 5.9.3).

- In Section 5.7, we show that the learning problem supports *prior* probabilities of database tuples which can be incorporated to *update and clean* probabilistic databases. Also, we demonstrate that the learning problem supports the creation of *probabilistic databases from incomplete databases* and applying *constraints* to probabilistic databases.

- Additionally, we perform an *experimental evaluation* on three different real world datasets as well as on synthetic data, where we compare our approach to various techniques based on statistical relational learning, reasoning in information extraction, and optimization (Section 5.9).

5.4 Preliminary

For the theoretical analysis of the learning problem presented next, we devise an alternative way of computing probabilities of lineage formulas via polynomial expressions. First, we reduce the number of terms in the sum of Equation (2.3) by considering just tuples $\mathit{Tup}(\phi)$ that occur in the propositional lineage formula ϕ.

Proposition 5.1. *We can compute $P(\phi)$ relying on tuples in $Tup(\phi)$, only, by writing:*

$$P(\phi) = \sum_{\mathcal{V} \in \mathcal{M}(\phi, Tup(\phi))} \underbrace{P(\mathcal{V}, Tup(\phi))}_{\textit{Definition 2.10}} \tag{5.1}$$

Proof.

$$\begin{aligned} P(\phi) &= \textstyle\sum_{\mathcal{W} \in \mathcal{M}(\phi, \mathcal{T})} P(\mathcal{W}, \mathcal{T}) \\ &= \Big(\textstyle\sum_{\mathcal{V} \in \mathcal{M}(\phi, Tup(\phi))} P(\mathcal{V}, Tup(\phi))\Big) \cdot \underbrace{\Big(\textstyle\sum_{\mathcal{V} \subseteq (\mathcal{T} \setminus Tup(\phi))} P(\mathcal{V}, \mathcal{T} \setminus Tup(\phi))\Big)}_{=1 \text{ by Proposition 2.1}} \end{aligned}$$

□

So, Equation (5.1) expresses $P(\phi)$ as a polynomial. Its terms are defined as in the third item of Definition 2.10, and the variables are $p(I)$ for $I \in Tup(\phi)$. The degree of the polynomial is limited as follows.

Corollary 5.1. *The probability $P(\phi)$ of a propositional lineage formula ϕ can be expressed by a multi-linear polynomial over variables $p(I)$, for $I \in Tup(\phi)$, with a degree of at most $|Tup(\phi)|$.*

Proof. By inspecting Proposition 5.1, we note that the sum ranges over subsets of $Tup(\phi)$ only, hence each term has a degree of at most $|Tup(\phi)|$. □

Example 5.2. *Considering the propositional lineage formula $\phi \equiv I_1 \vee I_2$, the occurring tuples are $Tup(\phi) = \{I_1, I_2\}$. Then, it holds that $\{I_1, I_2\} \models \phi$, $\{I_1\} \models \phi$, and $\{I_2\} \models \phi$. Hence, we can write $P(\phi) = p(I_1) \cdot p(I_2) + p(I_1) \cdot (1 - p(I_2)) + (1 - p(I_1)) \cdot p(I_2)$. Thus, $P(\phi)$ is a polynomial over the variables $p(I_1)$, $p(I_2)$ and has degree $2 = |Tup(\phi)| = |\{I_1, I_2\}|$.*

5.5 Learning Problem

We now move away from the case where the probability values of all database tuples are known, which was true for all previous chapters. Instead, we intend to learn the unknown probability values of (some of) these tuples (e.g. of I_5–I_9 in Example 5.1). More formally, for a tuple-independent probabilistic database $(\mathcal{T}, p)$, we consider $\mathcal{T}_l \subseteq \mathcal{T}$ to be the set of base 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 $I \in \mathcal{T} \setminus \mathcal{T}_l$. 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. We formally define the resulting learning problem as follows.

Definition 5.1. *We are given a probabilistic database* $(\mathcal{T}, p)$*, a set of tuples* $\mathcal{T}_l \subseteq \mathcal{T}$ *with unknown probability values* $p(I_l)$ *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 follows:*

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

Intuitively, we aim to set the probability values of the base tuples $I_l \in \mathcal{T}_l$ such that the labeled lineage formulas ϕ_i yield the probability l_i. We want to remark that 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. Hence, Boolean labels resolve to a special case of the labels of Definition 5.1.

Example 5.3. *Formalizing the problem setting of Example 5.1, we obtain* $\mathcal{T} := \{I_1, \ldots, I_9\}$*,* $\mathcal{T}_l := \{I_5, \ldots, I_9\}$ *with labels* $((I_1 \wedge I_5 \wedge I_8) \vee (I_2 \wedge I_6 \wedge I_8), 0.7)$*, and* $((I_3 \wedge I_7 \wedge I_9), 0.0)$*.*

5.5.1 Complexity

We next discuss the complexity of solving the learning problem. Unfortunately, it exhibits hard instances. First, computing $P(\phi_i)$ may be $\#\mathcal{P}$-hard (see Lemma 2.2), which would require many Shannon expansions. But even for cases when all $P(\phi_i)$ can be computed in polynomial time (i.e., when Equation (2.4) is applicable), there are combinatorially hard cases of the above learning problem.

Lemma 5.1. *For a given instance of the learning problem of Definition 5.1, 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.*

Proof. We encode the 3-satisfiability problem (3SAT) [54] for a Boolean formula $\psi \equiv \psi_1 \wedge \cdots \wedge \psi_n$ in conjunctive normal form into the learning problem of Definition 5.1. For each variable $X_i \in \mathit{Var}(\psi)$, we create two tuples I_i, I'_i whose probability values will be learned. Hence, $2 \cdot |\mathit{Var}(\psi)| = |\mathcal{T}_l| = |\mathcal{T}|$. Then, for each X_i, we add the label $((I_i \wedge I'_i) \vee (\neg I_i \wedge \neg I'_i), 1.0)$. The corresponding polynomial equation $p(I_i)p(I'_i) + (1 - p(I_i))(1 - p(I'_i)) = 1.0$ has exactly two possible solutions for $p(I_i), p(I'_i) \in [0,1]$, namely $p(I_i) = p(I'_i) = 1.0$ and $p(I_i) = p(I'_i) = 0.0$. Next, we replace all variables X_i in ψ by their tuple I_i. Now, for each clause ψ_i of ψ, we introduce one label $(\psi_i, 1.0)$. Altogether, we have $|\mathcal{L}| = |\mathit{Var}(\psi)| + n$ labels for the problem of Definition 5.1. Each labeled lineage formula ϕ has at most three variables, hence $P(\phi)$ takes at most 8 steps. Still, Definition 5.1 solves 3SAT, where the learned values of each pair of $p(I_i)$, $p(I'_i)$ (either 0.0 or 1.0) correspond to truth value of X_i for a satisfying assignment of ψ. From this, it follows that the decision problem formulated in Lemma 5.1 is $\mathcal{NP}$-hard. $\square$

5.5.2 Solutions

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 $p : \mathcal{T}_l \to [0,1]$ such that all labels are satisfied.

Example 5.4. *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.*

From a practical point of view, there remain a number of questions regarding Definition 5.1. 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., $|Tup(\phi_i)|$)? These questions are addressed by the following theorem. It is based on the computation of probabilities of lineage formulas via their polynomial representation as in Corollary 5.1. We write the conditions of the learning problem $P(\phi_i) = l_i$ as polynomials over variables $p(I_l)$ of the form $P(\phi_i) - l_i$, where $I_l \in \mathcal{T}_l$ and the probability values $p(I)$ for all $I \in \mathcal{T} \backslash \mathcal{T}_l$ are fixed and hence represent constants.

Theorem 5.1. *If the labeling is consistent, the problem instances of Definition 5.1 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 $\prod_i |Tup(\phi_i) \cap \mathcal{T}_l|$ 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.*

Proof. The first case is a classical under-determined system of equations. In the second case, without loss of generality, there are two polynomials $P(\phi_i) - l_i$ and $P(\phi_j) - l_j$ with a common zero, say $p(I_k) = c_k$. Setting $p(I_k) = c_k$ satisfies both $P(\phi_i) - l_i = 0$ and $P(\phi_j) - l_j = 0$, hence we have $\mathcal{L}' := \mathcal{L} \backslash \langle (\phi_i, l_i), (\phi_j, l_j)\rangle$ and $\mathcal{T}_l' := \mathcal{T}_l \backslash \{I_k\}$ which yields the first case of the theorem again ($|\mathcal{L}'| < |\mathcal{T}_l'|$). Regarding the third case, Bezout's theorem [36], a central result from algebraic geometry, is applicable: for a system of polynomial equations, the number of solutions (including their multiplicities) over variables in $\mathbb{C}$ is equal to the product of the degrees of the polynomials. In our case, the polynomials are $P(\phi_i) - l_i$ with variables $p(I_l)$ where $I_l \in \mathcal{T}_l$. So, according to Corollary 5.1 their degree is at most $|\mathit{Tup}(\phi_i) \cap \mathcal{T}_l|$. Since our variables $p(I_l)$ range only over $[0,1] \subset \mathbb{R}$, and Corollary 5.1 is an upper bound only, $\prod_i |\mathit{Tup}(\phi_i) \cap \mathcal{T}_l|$ is an upper bound on the number of solutions. In the fourth case, the system of equations is over-determined, such that redundancies like common zeros reduces the problem to one of the previous cases. $\square$

Example 5.5. *We illustrate the theorem by providing examples for each of the four cases.*

1. *In Example 5.3's formalization of Example 5.1, we have* $|\mathcal{T}_l| = 5$ *and* $|\mathcal{L}| = 2$*. So, the problem is under-specified and has infinitely many solutions, since assigning* $p(I_7) = 0.0$ *enables* $p(I_9)$ *to take any value in* $[0,1] \subset \mathbb{R}$*.*

2. *We assume* $\mathcal{T}_l = \{I_5, I_6, I_7\}$*, and* $\mathcal{L} = \langle (I_5 \wedge \neg I_6, 0.0), (I_5 \wedge \neg I_6 \wedge I_7, 0.0), (I_5 \wedge I_7, 0.0) \rangle$*. This results in the equations* $p(I_5) \cdot (1 - p(I_6)) = 0.0$*,* $p(I_5) \cdot (1 - p(I_6)) \cdot p(I_7) = 0.0$*, and* $p(I_5) \cdot p(I_7) = 0.0$*, where* $p(I_5)$ *is a common zero to all three polynomials. Hence, setting* $p(I_5) = 0.0$ *allows* $p(I_6)$ *and* $p(I_7)$ *to take any value in* $[0,1] \subset \mathbb{R}$*.*

3. *Let us consider* $\mathcal{T}_l = \{I_7, I_8\}$*.*

   (a) *If* $\mathcal{L} = \langle (I_7, 0.4), (I_8, 0.7) \rangle$*, then there is exactly one solution as predicted by* $|\mathit{Tup}(I_7)| \cdot |\mathit{Tup}(I_8)| = 1$*.*

   (b) *If* $\mathcal{L} = \langle (I_7 \wedge I_8, 0.1), (I_7 \vee I_8, 0.6) \rangle$*, then there are two solutions, namely* $p(I_7) = 0.2$*,* $p(I_8) = 0.5$ *and* $p(I_7) = 0.5$*,* $p(I_8) = 0.2$*. Here,* $\prod_i |\mathit{Tup}(\phi_i) \cap \mathcal{T}_l| = |\mathit{Tup}(I_7 \wedge I_8)| \cdot |\mathit{Tup}(I_7 \vee I_8)| = 4$ *is an upper bound.*

4. *We extend the second case of this example by the label* $(I_5, 0.0)$*, thus yielding the same solutions but having* $|\mathcal{L}| > |\mathcal{T}_l|$*.*

In general, a learning problem instance has many solutions, where Definition 5.1 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 $|\mathit{Tup}(\phi_i) \cap \mathcal{T}_l|$). Hence, to achieve more uniquely specified probabilities for all tuples $I_l \in \mathcal{T}_l$, in practice we should obtain the same number of labels as the number of tuples for which we learn their probability values, i.e., $|\mathcal{L}| = |\mathcal{T}_l|$, and label those lineage formulas with fewer tuples in $\mathcal{T}_l$.

Now that we characterized the number of solutions, we furthermore provide an insight on their nature. We give conditions on learning problems which imply the existence of an integer solution, i.e. that assigns only 0 or 1 as tuple probabilities. Hence, the resulting tuples are either non-existent or deterministic as in conventional databases.

Proposition 5.2. *For a learning problem, where*

1. $\forall I \in \mathcal{T} \backslash \mathcal{T}_l : p(I) \in \{0, 1\}$
2. $(\phi_i, l_i) \in \mathcal{L} : l_i \in \{0, 1\}$
3. $\bigwedge_{(\phi_i,1)\in\mathcal{L}} \phi_i \ \wedge \bigwedge_{(\phi_i,0)\in\mathcal{L}} \neg\phi_i$ *is satisfiable,*

there exists an integer solution p'*, that is for all* $I_l \in \mathcal{T}_l : p'(I_l) \in \{0, 1\}$.

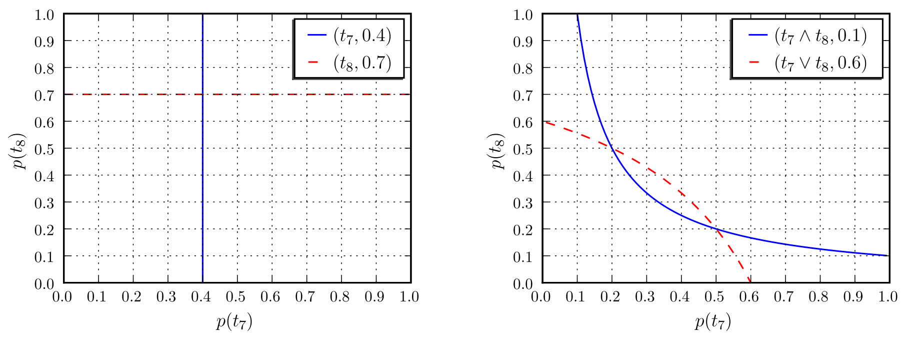
Proof. Due to the first requirement we can remove all tuples in $\mathcal{T} \backslash \mathcal{T}_l$ from the labels' formulas ϕ, since these tuples correspond to either *true* or *false*. Likewise, the second condition allows the construction of the formula $\bigwedge_{(\phi_i,1)\in\mathcal{L}} \phi_i \wedge \bigwedge_{(\phi_i,0)\in\mathcal{L}} \neg\phi_i$. As we require the existence of a satisfying assignment for this formula, precisely this assignment is the integer solution. □

5.5.3 Visual Interpretation

Based on algebraic geometry, the learning problem allows for a visual interpretation. All possible definitions of probability values for tuples in $\mathcal{T}_l$, that is, $p : \mathcal{T}_l \to [0, 1]$, span the hypercube $[0, 1]^{|\mathcal{T}_l|}$. In Example 5.5, cases 3(a) and 3(b), the hypercube has two dimensions, namely $p(I_7)$ and $p(I_8)$, as depicted in Figures 5.3(a) and 5.3(b). Hence, one definition of p specifies exactly one point in the hypercube. Moreover, all definitions of p that satisfy a given label define a curve (or plane) through the hypercube (e.g., the two labels in Figure 5.3(a) define two straight lines). Also, the points, in which all labels' curves intersect, represent solutions to the learning problem (e.g., the solutions of Example 5.5, case 3(b), are the intersections in Figure 5.3(b)). If the learning problem is inconsistent, there is no point in which all labels' curves intersect. Furthermore, if the learning problem has infinitely many solutions, the labels' curves intersect in curves or planes, rather than points.

5.6 Gradient Based Solutions

In the previous section, we formally characterized the learning problem and devised the basic properties of its solutions. From a visual perspective, Definition 5.1 established curves and planes whose intersections represent the solutions (see, e.g., Figure 5.3(b)). In this section, we introduce different objective functions that describe surfaces whose optima correspond to these solutions. For instance, the problem of Figure 5.3(b) has the surface of Figure 5.4(a) if we the employ mean squared error (MSE) as the objective,



(a) Labels of case 3(a) of Example 5.5 (b) Labels of case 3(b) Example 5.5

Figure 5.3: Visualization of the Learning Problem

which will be defined in this section. Calculating a gradient on such a surface thus allows the application of an optimization method to solve the learning problem.

Alternative Approaches In general, based on the polynomial equations, an exact solution to an instance of the learning problem can be computed in exponential time [36], which is not acceptable in a database setting. Also, besides gradient-based optimization methods, other approaches, such as expectation maximization [61, 63], are possible and represent valuable targets for future work.

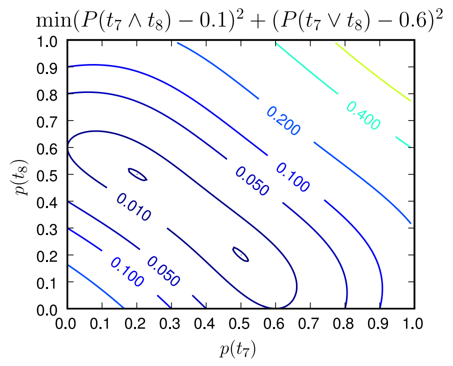
5.6.1 Desired Properties

Before we define objective functions for solving the learning problem, we establish a list of desired properties of these (which we do not claim to be complete). Later, we judge different objectives based on these properties.

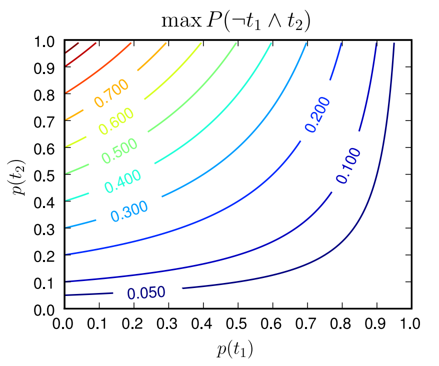
Definition 5.2. *An objective function to the learning problem should satisfy the following three* desired properties*:*

1. *All instances of the learning problem of Definition 5.1 can be expressed, including inconsistent ones.*
2. *If all $P(\phi_i)$ are computable in polynomial time, then also the objective is computable in polynomial time.*
3. *The objective is stable, that is $\mathcal{L} := \langle(\phi_1, l_1), \dots, (\phi_n, l_n)\rangle$ and $\mathcal{L} \cup \langle(\phi'_i, l_i)\rangle$ with $\phi'_i \equiv \phi_i$, $(\phi_i, l_i) \in \mathcal{L}$ define the same surface.*

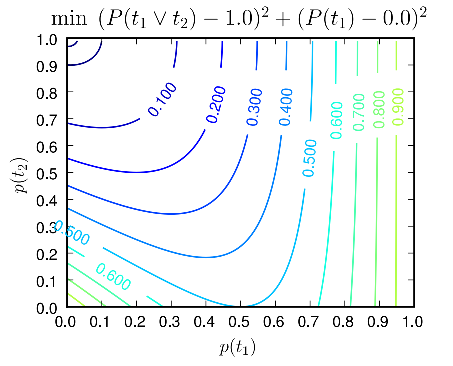
Here, the first case ensures that the objective can be applied to all instances of the learning problem. We insist on inconsistent instances, because they occur often in practice (see Table A.1). The second property restricts a blow-up in computation, which yields the following useful characteristic: if we can compute $P(\phi)$ for all labels, e.g., for labeled query answers, then we can also compute the objective function. Finally, the last of the desiderata reflects an objective function's ability to detect dependencies between labels. Since $\phi_i \equiv \phi'_i$ both $\mathcal{L}$ and $\mathcal{L} \cup \langle(\phi'_i, l_i)\rangle$ allow exactly the same solutions, the surface should be the same. Unfortunately, including convexity of an objective as an additional desired property is not possible. For example Figure 5.3(b) has two disconnected solutions, which induce at least two optima, thus prohibiting convexity. In the following, we establish two objective functions, which behave very differently with respect to the desired properties.



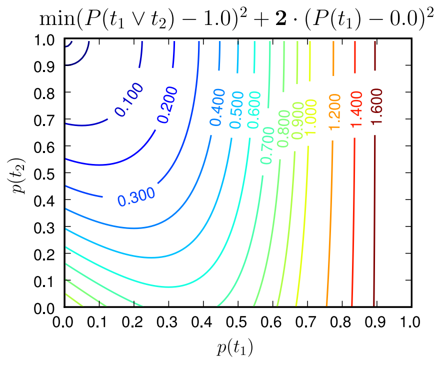
(a) Example 5.5: 3(b): MSE objective



(b) Example 5.6: Logical objective



(c) Example 5.8: MSE objective



(d) Example 5.8: MSE objective, unstable

Figure 5.4: Visualization of the Objective Functions

5.6.2 Logical Objective

If we restrict the probability labels of the learning problem to $l_i \in \{0.0, 1.0\}$, we can define a objective function based on computing probabilities of lineage formulas as follows.

Definition 5.3. *Let an instance of the learning problem of Definition 5.1 be given by a probabilistic database* $(\mathcal{T}, p)$*, tuples with unknown probability values* $\mathcal{T}_l \subseteq \mathcal{T}$*, and labels* $\mathcal{L} = \langle(\phi_1, l_1), \ldots, (\phi_n, l_n)\rangle$ *such that all* $l_i \in \{0.0, 1.0\}$*. Then, the* logical objective *is formulated as follows:*

$$Logical(\mathcal{L}, p) := P\left(\bigwedge_{(\phi_i,l_i)\in\mathcal{L},l_i=1.0} \phi_i \wedge \bigwedge_{(\phi_i,l_i)\in\mathcal{L},l_i=0.0} \neg\phi_i\right) \tag{5.2}$$

The above definition is a maximization problem, and its global optima are identified by $Logical(\mathcal{L}, p) = 1.0$. Moreover, from Definition 2.13, we may obtain its derivative.

Example 5.6. *Let* $\mathcal{T} = \mathcal{T}_l := \{I_1, I_2\}$ *and* $\mathcal{L} := \langle(I_1 \vee I_2, 1.0), (I_1, 0.0)\rangle$ *be given. Then,* Logical$(\mathcal{L}, p)$ *is instantiated as* $P((I_1 \vee I_2) \wedge \neg I_1) = P(\neg I_1 \wedge I_2)$*. Visually, this defines a surface whose optimum lies in* $p(I_1) = 0.0$ *and* $p(I_2) = 1.0$*, as shown in Figure 5.4(b).*

With respect to Definition 5.2, the third desired property is fulfilled, as $P(\phi_i' \wedge \phi_i) = P(\phi_i)$. Hence, the surface of the logical objective, shown for instance in Figure 5.4(b), is never altered by adding equivalent labels. Still, the first property is not given, since the probability labels are restricted to $l_i \in \{0.0, 1.0\}$ and inconsistent problem instances collapse Equation (5.2) to $P(\mathit{false})$, thus rendering the objective non-applicable. Also, the second property is violated, because in the spirit of the proof of Lemma 5.1, we can construct an instance where for each label $P(\phi_i)$ on its own is computable in polynomial time, whereas the computation of the probability for Equation (5.2) is $\#\mathcal{P}$-hard.

5.6.3 Mean Squared Error Objective

Another approach, which is common in machine learning, lies in using the mean squared error (MSE) to define the objective function.

Definition 5.4. *Let an instance of the learning problem of Definition 5.1 be given by a probabilistic database* $(\mathcal{T}, p)$*, tuples with unknown probability values* $\mathcal{T}_l \subseteq \mathcal{T}$*, and labels* $\mathcal{L} = \langle(\phi_1, l_1), \ldots, (\phi_n, l_n)\rangle$*. Then, the* mean squared error objective function *is formulated as:*

$$MSE(\mathcal{L}, p) := \frac{1}{|\mathcal{L}|} \sum_{(\phi_i,l_i)\in\mathcal{L}} (P(\phi_i) - l_i)^2$$

Moreover, its partial derivative with respect to the probability value $p(I)$ *of the tuple is:*

$$\frac{\partial MSE(\mathcal{L}, p)}{\partial p(I)} := \frac{1}{|\mathcal{L}|} \sum_{(\phi_i,l_i)\in\mathcal{L}, I\in Tup(\phi_i)} 2 \cdot (P(\phi_i) - l_i) \cdot \underbrace{\frac{\partial P(\phi_i)}{\partial p(I)}}_{Definition\ 2.13}$$

The above formulation is a minimization problem whose solutions have 0.0 as the value of the objective.

Example 5.7. *Example 5.5, case 3(b), is visualized in Figure 5.3(b). The corresponding surface induced by the MSE objective is depicted in Figure 5.4(a) and has its minima at the the solutions of the learning problem.*

Judging the above objective by means of Definition 5.2, we realize that the first property is met, as there are no restrictions on the learning problem, and inconsistent instances can be tackled (but deliver objective values larger than zero). Furthermore, since the $P(\phi_i)$'s occur in separate terms of the sum of the objective, the second desired property is maintained. However, the third desired property is violated, as illustrated by the following example.

Example 5.8. *In accordance to Example 5.6 and Figure 5.4(b), we set* $\mathcal{T} = \mathcal{T}_l := \{I_1, I_2\}$ *and* $\mathcal{L} := \langle (I_1 \vee I_2, 1.0), (I_1, 0.0) \rangle$*. Then, the MSE objective defines the surface in Figure 5.4(c). However, if we replicate the label* $(I_1, 0.0)$*, thus resulting in Figure 5.4(d) (note the "times two" in the objective), its surface becomes steeper along the* $p(I_1)$*-axis, but has the same minimum. Thus, MSE's surface is not stable. Instead, it becomes more ill-conditioned* [103].

5.6.4 Discussion

Both the logical objective and the MSE objective have optima exactly at the solutions of the learning problem of Definition 5.1. With respect to the desired properties of Definition 5.2, we summarize the behavior of both objectives in the following table:

	Properties		
Objective	1.	2.	3.
Logical	×	×	✓
MSE	✓	✓	×

The two objectives satisfy opposing desired properties, and it is certainly possible to define other objectives behaving similarly to one of them. Unfortunately, there is little hope for an objective that is adhering to all three properties. The second property inhibits computational hardness. However, Lemma 5.1 and the third property's logical tautology checking (i.e., $\models \phi_i \leftrightarrow \phi'_i$, which is co-$\mathcal{NP}$-complete) require this. In this regard the logical objective addresses both computationally hard problems by computing probabilities, whereas the MSE objective avoids them.

In the remainder of this chapter, we will favor the MSE objective, as it is more practical. In reality, many learning problem instances are inconsistent or have non-Boolean labels (see Table A.1), and the probability computations of Equation (5.2) are often too expensive (see Section 5.9.4).

5.7 Extensions and Applications

In this section we present extensions to the learning problem being priors for tuple probabilities (see Section 5.7.1), support of temporal data in the learning problem (see Section 5.7.2) and encoding of constraints in form of labels (see Section 5.7.3). Furthermore, we devise how tuple-independent probabilistic databases can be updated or cleaned using the learning problem (see Section 5.7.4), and how tuple-independent probabilistic databases can be derived from incomplete databases (see Section 5.7.5).

5.7.1 Priors

In order to explicitly incorporate preferences in the form of prior probabilities of base tuples $I_l \in \mathcal{T}_l$ into our learning objective (instead of just considering them to be "unknown"), we can extend the MSE objective of Definition 5.4 as follows.

Definition 5.5. *Given a function prior* : $\mathcal{T}_l \rightarrow [0,1] \subset \mathbb{R}$*, the MSE objective function of Definition 5.4 can be extended to*

$$\frac{c}{|\mathcal{L}|} \cdot \sum_{(\phi_i, l_i) \in \mathcal{L}} (P(\phi_i) - l_i)^2 + \frac{1-c}{|\mathcal{T}_l|} \cdot \sum_{I_l \in \mathcal{T}_l} (P(I_l) - \mathit{prior}(I_l))^2$$

where $c \in [0,1]$ *is a constant.*

Utilizing c, we can control the trade-off between the impact of the lineage labels and the *prior* function.

Example 5.9. *We extend the setting of Example 5.1 by utilizing priors encoding our intuition on the correctness for each textual pattern. For instance, an information extraction expert could deliver* $\text{prior}(I_5) = 0.6$ (Received)*,* $\text{prior}(I_6) = 0.9$ (Won)*,* $\text{prior}(I_7) = 0.9$ (Born) *and for each domain* $\mathit{prior}(I_8) = 0.8$ (Wikipedia.org)*,* $\mathit{prior}(I_9) = 0.9$ (Imdb.com)*. Therefore, in contrast to the problem lacking priors which was under-determined (see Example 5.3), we end up in an over-determined learning problem, thus encoding more information.*

Expressiveness Definition 5.5 is not more general than the original MSE objective. We can express priors in Definition 5.4 by creating a label (I_l, $\mathit{prior}(I_l)$) for each tuple $I_l \in \mathcal{T}_l$, which then produces $\sum_{I_l \in \mathcal{T}_l} (P(I_l) - \mathit{prior}(I_l))^2$ also in the objective of Definition 5.4. The coefficients preceding the sums can be emulated by replicating labels in $\mathcal{L}$. Thus, priors are a special case of lineage labels.

5.7.2 Temporal Data

We turn to the support of temporal data within the learning problem. As stated in Observation 3.1 temporal deduction rules and temporal constraints produce purely propositional lineage formulas, which hence can be employed directly within labels. Of course, these lineage formulas can be induced by temporal deduction rules and temporal constraints. In Section 5.9.1, we will encode a real-life temporal information extraction instance in the learning problem.

5.7.3 Constraints

In the following, we investigate the relationship between constraints, that is, conditioning [83] of Section 2.2.6, and the learning problem, by encoding one in to the other in both ways.

Conditioning by Learning In Section 2.2.6 we represented constraints as the propositional formula ϕ_c over the tuples of a probabilistic database. We can encode the constraint ϕ_c with the label $(\phi_c, 1.0)$ into an instance of the learning problem.

Example 5.10. *In Example 5.1 we can require the BornIn relation to be functional, since people are born in only one place. Hence,* BornIn(Spielberg, Cinncinati) *and* BornIn(Spielberg, LosAngeles) *can not co-exist, which we express via their lineage formulas as* $\neg(I_3 \wedge I_7 \wedge I_8) \vee \neg(I_4 \wedge I_7 \wedge I_9)$*. If we label this formula by* 1.0*, we can enforce it during learning.*

Next, we show that after learning the resulting tuple-independent database is conditioned as of Definition 2.15.

Proposition 5.3. *Given a probabilistic database* $(\mathcal{T}, p)$ *and constraints in the form of a satisfiable propositional formula* ϕ_c *over* $\mathcal{T}$*. Then, if we create a learning problem instance by setting* $\mathcal{T}_l := \mathcal{T}$ *and* $\mathcal{L} := \langle(\phi_c, 1.0)\rangle$*, its solution* p' *conditions the probabilistic database* $(\mathcal{T}, p)$ *with respect to* ϕ_c*. Hence, for a query answer represented by its propositional lineage formula* ψ*, over* $(\mathcal{T}, p')$ *it holds that:*

$$P(\psi \mid \phi_c) = P(\psi)$$

Proof. We observe that in the solution p' of the learning problem, we receive $P(\phi_c) = 1.0$. As ϕ_c holds in p' we can rewrite the probability of a query answer ψ as follows.

$$\begin{array}{rcl} P(\psi) & \overset{(2.3)}{=} & \sum_{\mathcal{W} \in \mathcal{M}(\psi)} P(\mathcal{W}) \\ & = & \sum_{\mathcal{W} \in \mathcal{M}(\psi) \cap \mathcal{M}(\phi_c)} P(\mathcal{W}) \\ & = & \sum_{\mathcal{W} \in \mathcal{M}(\psi \wedge \phi_c)} P(\mathcal{W}) \\ & = & P(\psi \wedge \phi_c) \end{array}$$

By combining both equations, we obtain over $(\mathcal{T}, p')$:

$$P(\psi \mid \phi_c) = \frac{P(\psi \wedge \phi_c)}{P(\phi_c)} = \frac{P(\psi)}{1.0} = P(\psi)$$

□

Therefore, after learning the constraints are satisfied in the resulting tuple-independent probabilistic database. However, the original work on conditioning [83] considers world-set decompositions [10], which are more expressive than tuple-independent databases (without lineage). Hence, conditioning and learning can yield different results as illustrated by the following example.

Example 5.11. *Consider $\mathcal{T} = \{I_1, I_2\}$ with $p(I_1) = 0.6$, $p(I_2) = 0.5$ and the constraint $\phi_c = I_1 \vee I_2$. Then, learning assigns $p'(I_1) = 1$ or $p'(I_2) = 1$ to fulfill the constraint. In contrast, conditioning computes:*

$$\frac{P(I_1 \wedge (I_1 \vee I_2))}{P(I_1 \vee I_2)} = \frac{P(I_1)}{P(I_1 \vee I_2)} = \frac{0.6}{0.8} = 0.75$$
$$\frac{P(I_2 \wedge (I_1 \vee I_2))}{P(I_1 \vee I_2)} = \frac{P(I_2)}{P(I_1 \vee I_2)} = \frac{0.5}{0.8} = 0.625$$

In conclusion, both approaches can tackle constraints, but do not necessarily produce the same result.

Learning by Conditioning We now cover the inverse direction, namely encoding a learning problem into a conditioning problem. Following the logical objective of Definition 5.3, we can solve a subset of possible learning problem instances by conditioning. The subset is characterized by instances with consistent labels, having $\mathcal{T} = \mathcal{T}_l$, and by restricting the lineage labels to $l_i \in \{0.0, 1.0\}$. We create a single constraint in the form of the conjunction of Equation (5.2), initially set all tuple probabilities to the same value, e.g. 0.5, and solve the resulting conditioning problem [83]. Again, since conditioning keeps world-set decompositions in the result, the produced probability values might differ from the learned ones.

5.7.4 Updating and Cleaning Probabilistic Databases

In this section, we discuss two applications of the learning problem, namely updating and cleaning tuple-independent probabilistic databases.

Updating If we are given an existing probabilistic database $(\mathcal{T}, p)$ and knowledge in the form of labeled lineage formulas $\mathcal{L} := \langle(\phi_1, l_1), \ldots, (\phi_n, l_n)\rangle$, we can update the tuples' probability values via the learning problem. We produce a new probabilistic database $(\mathcal{T}, p')$, whose probability values p' are updated according to the information provided in $\mathcal{L}$. To achieve this, we create a learning problem instance (whose solution is p') by using $\mathcal{L}$, setting $\mathcal{T}_l := \mathcal{T}$ and defining a prior $\mathit{prior}(t) := p(I)$ (see Section 5.7.1).

Example 5.12. *Assume our probabilistic database consists of two tuples $\mathcal{T} := \{I_1, I_2\}$ with $p(I_1) = 0.4$ and $p(I_2) = 0.8$. Now, a user tells us that $P(I_1 \vee I_2) = 1.0$ must hold for his application, so we update the database, by solving the learning problem instance $\mathcal{T}_l = \mathcal{T}$ and $\mathcal{L} := \langle (I_1, 0.4), (I_2, 0.8), (I_1 \vee I_2, 1.0) \rangle$. The first two labels, encode the old tuple probabilities as priors, whereas the last label is the information provided by the user. The resulting learning problem is inconsistent, but a local optimum is given by the updated tuple probabilities of $p'(I_1) = 0.4$ and $p'(I_2) = 1.0$.*

Cleaning After updating the new probability values p' allow for cleaning the probabilistic database as follows. If p' defines the probability value of a tuple to be 0.0, we can delete it from the database. Conversely, if p' yields 1.0 for the probability value of a tuple, we can move it into a new, deterministic relation.

Example 5.13. *With respect to Example 5.12 after updating I_2 is a deterministic tuple, which we might want to move to a deterministic relation.*

5.7.5 Incomplete Databases

A field that is related to probabilistic databases are incomplete databases. Intuitively, in an incomplete database some attributes values or entire tuples may be missing in the given database instance. A completion of an incomplete database can be seen as a possible world, where in a probabilistic database we associate these worlds with a probability.

Missing Attribute Values In [137], a probabilistic database is derived from an incomplete database which exhibits missing attributes in some of its tuples. Their idea is to estimate the probability of a possible completion of an incomplete tuple from the complete part of the database. We emulate their approach by our learning problem on tuple-independent databases.

Let an incomplete database be given by a set of complete tuples $\mathcal{T}_c$ and a set of incomplete tuples $\mathcal{T}_i$. We consider an incomplete tuple $R(\bar{X}) \in \mathcal{T}_i$ of relation R, where $\bar{X}$ besides constants comprises variables for the missing attribute values. Assuming a finite universe of constants $\mathcal{U}$, let $\mathcal{T}_\times := \mathcal{U}^r$ be its crossproduct, where r is the arity of R. Hence, $G(R(\bar{X}), \mathcal{T}_\times)$ are the substitutions standing for all possible completions of $R(\bar{X})$. Then, we create a new uncertain relation $R' := \{\sigma(\bar{X}) \mid \sigma \in G(R(\bar{X}), \mathcal{T}_\times)\}$ and add one deduction rule per substitution $\sigma_i \in G(R(\bar{X}), \mathcal{T}_\times)$:

$$\sigma_i(R(\bar{X})) \leftarrow \sigma_i(R'(\bar{X})) \wedge \bigwedge_{\substack{\sigma_j \in G(R(\bar{X}), \mathcal{T}_\times), \\ \sigma_i \neq \sigma_j}} \neg \sigma_j(R'(\bar{X}))$$

The above rules allow at most one completion σ_i of $R(\bar{X})$ to be true within a possible world, since the other completions are ruled out by the negation.

Now, we create labels following the approach of [137]. We consider a subset of argument values $\bar{X}' \subset \bar{X}$, that is $Var(\bar{X}') \supset Var(\bar{X})$. Then, we compute the frequency of how often completions of $\bar{X}'$ feature the constants bound by σ_i.

$$f_{\bar{X}'}(\sigma_i) := \frac{|\{\sigma_j \mid \sigma_j \in G(R(\bar{X}'), \mathcal{T}_c), \sigma_j(Var(\bar{X})) = \sigma_i(Var(\bar{X}))\}|}{|G(R(\bar{X}'), \mathcal{T}_c)|}$$

Then, for each completion σ_i, we generate the label $(\sigma_i(R(\bar{X})), f_{\bar{X}'}(\sigma_i))$. Besides these labels, the resulting learning problem instance is given by $\mathcal{T}_l$ comprising the tuples of the new relation R'.

Example 5.14. *Assume we are given a database on living places of movie directors:*

LivesIn

	Director	Place
I_1	*Nolan*	*LosAngeles*
I_2	*Lynch*	*LosAngeles*
I_3	*Coppola*	*NapaCounty*
I_4	*Tarantino*	?

However, for Tarantino we are not aware where he is residing, which we estimate from the complete part of the database $\mathcal{T}_c = \{I_1, I_2, I_3\}$ as follows. The two possible completions of I_4 are $I_5 = \text{LivesIn}(\text{Tarantino}, \text{LosAngeles})$ and $I_6 = \text{LivesIn}(\text{Tarantino}, \text{NapaCounty})$. By counting how often each place occurs in $\mathcal{T}_c$, we create the labels $(I_5 \wedge \neg I_6, 0.666)$ and $(I_6 \wedge \neg I_5, 0.333)$, which we can turn to the learning problem machinery.

Missing Tuples Generally, any deterministic relation instance can be seen as a finite subset of the cross-product of its attributes' domains. We now consider an incomplete database, whose (finite sets of) existing tuples and potentially missing tuples are $\mathcal{T}_c$ and $\mathcal{T}_m$, respectively. Assume we intend to enforce logical formulas $\phi_1, \ldots, \phi_n$ over tuples $\mathcal{T}_c \cup \mathcal{T}_m$, which could for example result from constraints or user feedback. We create a learning problem instance by setting $\mathcal{T} := \mathcal{T}_c \cup \mathcal{T}_m$, $\mathcal{T}_l := \mathcal{T}_m$ and $\mathcal{L} := \langle(\phi_1, 1.0), \ldots, (\phi_n, 1.0)\rangle$. Thus, a solution to the learning problem completes $\mathcal{T}_c$ with (possibly uncertain) tuples from $\mathcal{T}_m$, such that the logical formulas $\phi_1, \ldots, \phi_n$ are fulfilled. In addition, if $\phi_1 \wedge \cdots \wedge \phi_n$ is satisfiable, then Proposition 5.2 teaches us that we can find a learning problem solution p', where all tuple probabilities $p'(I_m) \in \{0, 1\}$. Otherwise, the we can find an approximating solution, which hence has uncertain tuples from $\mathcal{T}_m$.

5.8 Algorithm

Given the surface of a learning problem (see, e.g., Figure 5.4(a)), as it is defined by the choice of the objective function, the learning algorithm of this section determines how to move over this surface in order to reach an optimum, that is to find a solution to the learning problem.

Learning Algorithm Our learning algorithm is based on stochastic gradient descend (SGD) [17], which we demonstrate to scale to instance sizes with millions of tuples and hundreds of thousands of labels (see Section 5.9.3). It is initialized at a random point and repeatedly moves into the direction of a partial derivative until convergence. Visually, we start at a random point (e.g., somewhere in Figure 5.4(a)), and then in each step we move in parallel to an axis (e.g., $p(I_1)$ or $p(I_2)$), until we reach an optimum.

In Algorithm 7 *best*, represents the best known value of the objective, where p holds the corresponding probability values of tuples in $\mathcal{T}_l$. Also, η_l is the learning rate, which exists and may differ for each tuple in $\mathcal{T}_l$. The loop of Line 4 is executed until convergence to the absolute error bound of ϵ_{abs}. Then, Line 5 shuffles the order of the tuples in $\mathcal{T}_l$ for the inner loop of Line 6. Within each iteration, Line 8 updates the probability value of one tuple, which yields the updated definition p' of p. If p' is an improvement over p with respect to the objective (as verified in Line 11), we assign p' to p and double the learning rate η_l of the tuple. Otherwise, p' is discarded, and the learning rate η_l is halved.

Algorithm 7 Learning$((\mathcal{T}, p), \mathcal{T}_l, \mathcal{L}, \epsilon_{abs})$

```
Input: Probabilistic database $(\mathcal{T}, p)$, tuples $\mathcal{T}_l$ to learn the probability values for,
    labeled propositional lineage formulas $\mathcal{L}$, error bound $\epsilon_{abs}$
Output: $p$ with learned probability values, best value of objective
 1: $\forall I_l \in \mathcal{T}_l : p(I_l) := Rand(0,1)$                ▷ Random initialization
 2: $\forall I_l \in \mathcal{T}_l : \eta_l := 1.0$                       ▷ Per-tuple learning rate
 3: $best := MSE(\mathcal{L}, p)$                                          ▷ See Definition 5.4
 4: while $best > \epsilon_{abs}$ do
 5:     $sequence := Shuffle(\mathcal{T}_l)$                               ▷ Permuted sequence
 6:     while $\neg IsEmpty(sequence)$ do
 7:         $I_l := Pop(sequence)$                                         ▷ Get first element
 8:         $p'(I_l) := p(I_l) - \eta_l \cdot \frac{\partial MSE(\mathcal{L},p)}{\partial p(I_l)}$   ▷ See Definition 5.4
 9:         $p' := \begin{cases} p(I) & \text{if } I \neq I_l \\ p'(I_l) & \text{otherwise} \end{cases}$
10:         $newVal := MSE(\mathcal{L}, p')$                               ▷ See Definition 5.4
11:         if $newVal < best$ then
12:             $\eta_l := 2 \cdot \eta_l$                                 ▷ Increase $I_l$'s learning rate
13:             $p := p'$                                                  ▷ Keep new value of $p(I_l)$
14:             $best := newVal$
15:         else
16:             $\eta_l := \frac{1}{2} \cdot \eta_l$                       ▷ Decrease $I_l$'s learning rate
17: return $p$, $best$
```

Example 5.15. *We execute Algorithm 7 on the example of Figure 5.4(c).*

Following Definition 5.4 the corresponding partial derivatives are:

$$\begin{aligned}\frac{\partial MSE}{\partial p(I_1)} &:= (P(I_1 \vee I_2) - 1.0) \cdot (P(\mathit{true} \vee I_2) - P(\mathit{false} \vee I_2)) \\ &\quad + (P(I_1) - 0.0) \cdot (P(\mathit{true}) - P(\mathit{false})) \\ \frac{\partial MSE}{\partial p(I_2)} &:= (P(I_1 \vee I_2) - 1.0) \cdot (P(I_1 \vee \mathit{true}) - P(I_1 \vee \mathit{false}))\end{aligned} \tag{5.3}$$

Assuming that Line 1 delivers $p(I_1) = 0.7$ *and* $p(I_2) = 0.5$*, we obtain* $\text{best} = (-0.15)^2 + (0.7)^2 \approx 0.512$ *in Line 3. If* $\epsilon_{abs} = 0.01$ *we enter the loop of Line 4, where Line 5 randomly orders* I_2 *before* I_1*. Then, the partial derivative of* $p(I_2)$ *evaluates as follows* $\frac{\partial MSE}{\partial p(I_2)}\big|_{(0.7,0.5)} = (0.85 - 1.0) \cdot (1.0 - 0.7) = -0.055$*. Since* $\eta_2 = 1.0$*, we have* $p'(I_2) = 0.5 - (-0.055) = 0.555$ *in Line 8. Hence, in Line 10,* $\text{newVal} = (-0.1335)^2 + 0.7^2 \approx 0.508$*. As* $0.508 < 0.512$*, the condition of Line 11 turns true, such that we obtain* $\eta_2 = 2.0$*,* $p(I_1) = 0.7$*,* $p(I_2) = 0.555$ *and* $\text{best} = 0.508$*. Hence, in further iterations the increased* η_2 *speeds up movements along the partial derivative of* $p(I_2)$*.*

Tackling Mean Squared Error's Instability In Section 5.6, we chose the mean squared error objective, which however comes with the disadvantage that it does not satisfy the third desired property of Definition 5.2. We argue that Algorithm 7 counters to some extent the instability, which we illustrate by the following example.

Example 5.16. *Let us evaluate the gradient of Figures 5.4(c) and 5.4(d) in the point* $p(I_1) = p(I_2) = 0.5$*. Following Equation* (5.3)*, we obtain the gradient* $(0.375, -0.125)$ *for Figure 5.4(c). Analogously, Figure 5.4(d) has* $(0.875, -0.125)$*. Even though both figures show the same minimum, the gradients differ heavily in the partial derivative of* $p(I_1)$*.*

Inspecting the above example, we note that the gradient is indeed affected, but each partial derivative on its own points into the correct direction, i.e. increasing $p(I_2)$ and decreasing $p(I_1)$. Hence, weighting the partial derivatives can counter the effect. We achieve this by keeping one learning rate η_l per tuple and adapting them during runtime. In Section 5.9.2, we empirically show a superior convergence over a global learning rate. Previously, in the context of Markov logic networks the authors of [91] also reported speed ups in ill-conditioned instances by introducing separate learning rates per dimension.

Implementation Issues In this paragraph, we briefly describe two implementation issues, which were omitted in Algorithm 7 for presentation purposes. First, the absolute error bound in Line 4 is inconvenient, because the optima of inconsistent learning problem instances have an MSE value larger than 0.0. Therefore, we rely on both an absolute error bound ϵ_{abs}

and a relative error bound ϵ_{rel}. Next, Line 8 might yield a probability value that exceeds the interval $[0, 1]$, which we counter by the *logit* function. It defines a mapping from probability values in $[0, 1]$ to $\mathbb{R} \cup \{\pm\infty\}$.

Definition 5.6. *The logit function transforms a probability $p \in [0, 1]$ to a weight $w \in \mathbb{R} \cup \{\pm\infty\}$ as follows:*

$$w = \ln \frac{p}{1.0 - p} \qquad p = \frac{1}{1 + \exp(-w)}$$

Example 5.17. *If $p = 0.5$, then $w = 0.0$. Also $p = 1.0$ implies $w = +\infty$, whereas $p = 0.0$ yields $w = -\infty$.* $\diamond$

Therefore, whenever we add a partial derivative to a tuple probability we perform this addition on the weights in $\mathbb{R} \cup \{\pm\infty\}$, rather than on probability values in $[0, 1]$.

Algorithm Properties Algorithm 7 comes with three properties, which we share with alternative approaches we are aware of, including other gradient-based methods and expectation maximization [61, 63]. First, the algorithm is *non-deterministic*, which is caused by Lines 1 and 5. Second, gradient-based optimization methods, including Algorithm 7, can get stuck in *local optima*, which is nevertheless hard to avoid in non-convex problems. In this regard, the non-determinism is a potential advantage, since restarting the algorithm yields varying solutions, thus increasing the chance for finding a global optimum. Finally, the solutions returned by Algorithm 7 for the mean squared error objective are *not exact*, but rather very close to an optimum. However this can be controlled by the error bounds ϵ_{rel} and ϵ_{abs}, which trades runtime for distance to the next optimum. In Section 5.9.4, we provide an empirical investigation of this issue.

5.9 Experiments

Our evaluation focuses on the following four aspects. In Section 5.9.1 we compare the quality of our approach to learning techniques in statistical relational learning and to constraint-based reasoning techniques applied in information extraction settings. In Section 5.9.2, we compare the runtime behavior of our algorithm to statistical relational learning methods and to other gradient-based optimization techniques. In Section 5.9.3, we explore the scalability of our method to large datasets. Finally, in Section 5.9.4, we analyze our algorithm by comparing both objective functions of Section 5.6, varying the error rates ϵ_{rel} and ϵ_{abs}, and analyzing the ability of Algorithm 7 to find global optima. During all this section, if not stated otherwise, *PDB* refers to a single-threaded implementation of Algorithm 7 with the mean squared error objective, and a per-tuple learning rate. For checking convergence, we set $\epsilon_{abs} = 10^{-6}$ and $\epsilon_{rel} = 10^{-4}$. Additionally, we present the basic characteristics of all learning problem instances in Table A.1, which can be found in the appendix.

5.9.1 Quality

Statistical Relational Learning Setting

In this task, we compare the predictive performance a probabilistic database with tuple probability learning to established methods from statistical relational learning.

Dataset We employ the UW-CSE dataset (see Appendix A.2.3), which comprises a database describing the University of Washington's computer science department via the following relations: *AdvisedBy*, *CourseLevel*, *HasPosition*, *InPhase*, *Professor*, *ProjectMember*, *Publication*, *Student*, *TaughtBy*, *Ta* (teaching assistant), and *YearsInProgram*. Moreover, the dataset is split into five sub-departments, and we consider the relations of this dataset to be deterministic.

Task The goal is inspired by an experiment in [120], namely to predict the *AdvisedBy* relation from all input relations except *Student* and *Professor*. We train and test in a leave-one-out fashion by sub-department.

Deduction Rules We automatically create 49 deduction rules resembling all joins (including self-joins) between two relations (except *Student*, *Professor*, and *AdvisedBy*), having at least one argument of type person. Furthermore, we add one uncertain relation *Rules*, containing one tuple for each of the 49 rules and include the corresponding tuple in the join [38], for example:

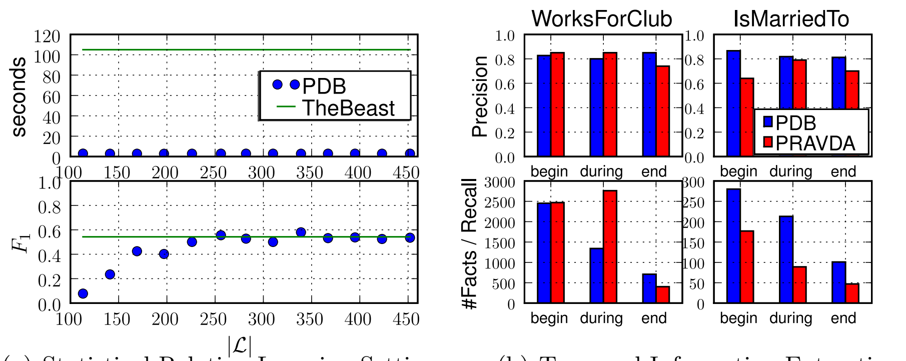
$$\mathit{AdvisedBy}(P_1, P_2) \leftarrow \left(\begin{array}{r} \mathit{Ta}(D, C, P_1, T) \wedge \\ \mathit{TaughtBy}(D, C, P_2, T) \wedge \end{array} \mathit{Rules}(1) \right)$$

The remaining rules are given in Appendix A.3.3. By this construction, all lineage instances of a single rule share the same uncertain tuple (e.g., $\mathit{Rules}(1)$). Furthermore, every grounded instance of *AdvisedBy* is expressed by the disjunction of tuples in *Rules* which deduce it. Then, we learn the probability values of the 49 tuples in *Rules*, hence classifying how well each rule predicts the *AdvisedBy* relation.

Labels Regarding labels, we employed the 113 instances of *AdvisedBy* as *positive labels*, i.e., all their probability labels are 1.0. In addition, there are about 16,000 person-person pairs not contained in *AdvisedBy*. Adhering the closed world assumption we randomly draw pairs from these as *negative labels* (with a probability label of 0.0).

Statistical Relational Learning Competitors We compete with *TheBeast* [121], the fastest Markov logic networks [120] implementation we are aware of. It is based on an in-memory database and performs inference via integer linear programming. We ran it on the same set of data, rules and probabilities to alter. Additionally, we ran the probabilistic Prolog engine *ProbLog* [63], but even on the reduced data size of one sub-department it did not terminate after one hour.

Results In Figure 5.5(a), we depict both the runtimes as well as the prediction quality in terms of the F_1 measure (the harmonic mean of precision and recall [95]) for the *AdvisedBy* relation. *TheBeast* is a straight line, since it allows only positive labels and negative labels are implicitly given by the closed world assumption. For *PDB*, we started with all positive labels and added increasing numbers of negative labels.



(a) Statistical Relation Learning Setting (b) Temporal Information Extraction

Figure 5.5: Quality Experiments

Analysis Regarding runtimes, *PDB* is consistently about 40 times faster than *TheBeast*. *PDB*'s runtime is not altered by adding labels, as it is mainly spent in grounding (see also Section 5.9.4). With respect to F_1, adding more negative labels to *PDB* yields improvements until we saturate at the same level as *TheBeast*.

Temporal Information Extraction

In the spirit of Example 5.1, we tackle the problem of extracting facts from free text by the use of textual patterns as introduced in Section 2.5. However, we consider facts with temporal annotations in the form of time intervals specifying the facts' validity. The temporal probabilistic database we are learning on follows the data model of Chapter 3.

Dataset The PRAVDA dataset [153] (see Appendix A.2.4) contains about 450,000 crawled web pages in the sports and celebrities domains, where about 12,500 textual patterns are employed to extract temporal facts.

Task Following [153], we consider two different temporal relations, namely $\mathit{WorksForClub}^T$ in the sports domain and $\mathit{IsMarriedTo}^T$ in the celebrities domain. The goal is to determine, for each textual pattern, whether it expresses a temporal *begin*, *during* or *end* event of one of the two relations, or none of them. For example, for $\mathit{WorksForClub}^T$, we could find that David Beckham joined Real Madrid in 2003 *(begin)*, scored goals for them in 2005 *(during)*, and left the club in 2007 *(end)*. Having classified the patterns, occurrences of entity pairs next to a textual pattern in the input documents thus yield the facts.

Probabilistic Database Setup We model temporal data in the probabilistic database as in Chapter 3. Text occurrences of a potential fact are stored in the deterministic relation $\mathit{Occurrence}^T(\mathit{Pid}, E_1, E_2, \mathit{Types}, T_b, T_e)$, where *Pid* is the pattern id, *Types* holds the types of the entities E_1, E_2 and T_b, T_e span the time interval. To encode the decision whether a pattern expresses a temporal *begin*, *during*, or *end* event, we instantiate three uncertain relations $\mathit{Begin}(\mathit{Pid})$, $\mathit{During}(\mathit{Pid})$, and $\mathit{End}(\mathit{Pid})$, which each hold one entry per pattern and whose probability values we learn. Text occurrences of potential facts are connected to the patterns by six rules (see Appendix A.3.3 for details) of the following kind

$$\begin{aligned} &\mathit{IsMarriedToBegin}^T(E_1, E_2, T_b, T_e) \\ &\quad \leftarrow \mathit{Begin}(\mathit{Pid}) \wedge \mathit{Occurrence}^T(\mathit{Pid}, E_1, E_1, pp, T_b, T_e) \end{aligned} \tag{5.4}$$

where *pp* stands for person-person type pair. To enforce that a textual pattern expresses at most one of *begin*, *during*, or *end*, we require them to be mutually exclusive via the deduction rules:

$$\begin{array}{lcl} \mathit{Constraint1}(\mathit{Pid}) & \leftarrow & \mathit{Begin}(\mathit{Pid}) \wedge \mathit{During}(\mathit{Pid}) \\ \mathit{Constraint2}(\mathit{Pid}) & \leftarrow & \mathit{Begin}(\mathit{Pid}) \wedge \mathit{End}(\mathit{Pid}) \\ \mathit{Constraint3}(\mathit{Pid}) & \leftarrow & \mathit{During}(\mathit{Pid}) \wedge \mathit{End}(\mathit{Pid}) \end{array}$$

Their resulting lineage formulas are labeled with 0.0. Moreover, we employ temporal precedence constraints by instantiating six rules of the form

$$\begin{aligned} & \qquad\qquad\qquad\quad \mathit{IsMarriedToBegin}^T(E_1, E_2, T_b, T_e) \wedge \\ &\mathit{Constraint4}(E_1, E_2) \leftarrow \mathit{IsMarriedToDuring}^T(E_1, E_2, T_b', T_e') \\ & \qquad\qquad\qquad\quad \wedge T_b' <^T T_e \end{aligned}$$

and label their lineage with 0.0. From the original work [153] we deploy 266 labels for textual patterns, i.e. which directly set the probability of a tuple in either *Begin*, *During* or *End*. Additionally, from [153] we adopt the 341 labels for facts, that is for lineage formulas resulting from one of the deduction rules like Equation (5.4). We have to stress that these labels feature probabilities and hence are non-Boolean.

Competitor The authors of [153] utilized a combination of Label Propagation and Integer Linear Programming to rate the textual patterns and to enforce temporal constraints.

Results In Figure 5.5(b), we report our the result of our system *(PDB)* along with the best result from [153] *(PRAVDA)*. To evaluate precision, we sampled 100 facts per relation and event type and annotated them manually. Recall is the absolute number of facts obtained.

Analysis For relations with a few, decisive textual patterns, *PDB* keeps up with precision, while slightly gaining in recall, probably due to the relaxation of constraints by the MSE objective. However, for *WorksForClub*'s *during* relation, there is a vast number of relevant patterns, which puts the undirected model of Label Propagation in favor, whereas our directed model suffers in terms of recall.

5.9.2 Runtime

Statistical Relational Learning Methods

We compare the runtimes of statistical relational learning implementations and our probabilistic database implementation with learning.

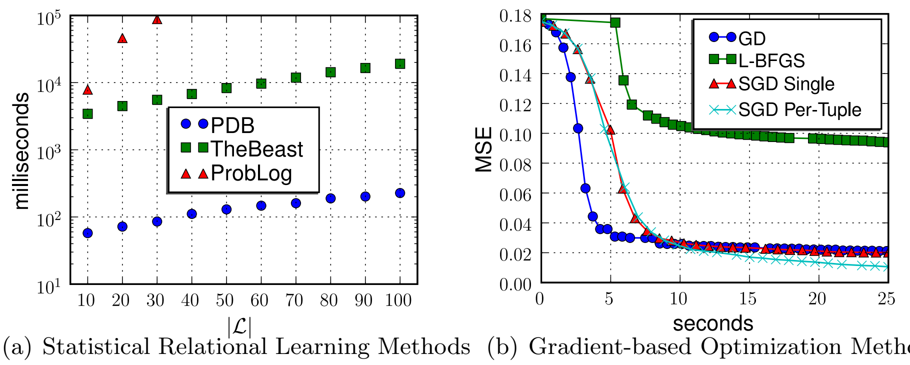
Setup To systematically verify scalability, we create synthetic datasets as follows. We fix $\mathcal{T} = \mathcal{T}_l$ to 100 tuples. Then, we instantiate a growing number of deduction rules of the form

$$\begin{aligned} &\mathit{Head}(c) \leftarrow R(i) \wedge \neg R(j) \wedge \neg R(k) \\ &\mathit{Head}(c) \leftarrow R(l) \wedge \neg R(m) \wedge \neg R(n) \end{aligned}$$

where all literals with relation R refer to a uniformly drawn tuple in $\mathcal{T}_l$, and negations exist with probability 0.5. The probability label of each $\mathit{Head}(c)$ is randomly set to either 0.0 or 1.0.

Competitors Besides *TheBeast* [121], we compete with *ProbLog* [63], a probabilistic Prolog engine, whose grounding techniques and distribution semantics are closest to ours.

Results For each value of $|\mathcal{L}|$, we create five problem instances and depict their average runtime in Figure 5.6(a).



(a) Statistical Relational Learning Methods (b) Gradient-based Optimization Methods

Figure 5.6: Runtime Experiments

Analysis *PDB* converges on average about 600 times faster than *ProbLog* and about 70 times faster than *TheBeast*. We believe that *ProbLog* is slowed down by its implementation which mainly connects a number of existing software packages, hence prohibiting optimizations. *TheBeast* repeatedly solves integer linear programs where our updating steps of Algorithm 7 are faster.

Gradient-based Optimization Methods

We compare the stochastic gradient descent (SGD) of Algorithm 7 with per-tuple learning rate to other gradient-based optimization methods.

Setup We employ the YAGO2 dataset (see Appendix A.2.1). The task is to learn the probability values of tuples $\mathcal{T}_l$ in the *LivesIn* relation. Moreover, we label the following rule's

$$\mathit{ToLabel}(L) \leftarrow \mathit{LivesIn}(P, L)$$

lineage formulas with synthetic target probabilities uniformly drawn from $[0, 1]$. Since the projection of the deduction rule on the first argument yields disjoint lineage formulas ϕ with respect to their tuples $T(\phi)$, the resulting learning problem instance is consistent. Hence, its global optima have a mean squared error (MSE) of 0.0.

Competitors Algorithm 7 with per tuple learning rate *(SGD Per-Tuple)* competes with a single learning rate *(SGD Single)*, with gradient descent *(GD)*, and with *L-BFGS* [103], which approximates the Hessian with its second derivatives. All methods are initialized with the same learning rate.

Results We plot the mean squared error (MSE) against the runtime of the different methods in Figure 5.6(b).

Analysis *GD* takes less time per iteration. Hence its curve drops faster in the beginning, but then stagnates. The two *SGD* variants behave similarly at first. Later on, the per-tuple learning rate yields constant improvements, whereas the single learning rate does not. This is caused by the fact that the per-tuple learning rate can distinguish between tuples where the surface of the objective function is smooth and where it is rigged. In contrast, the global learning rate has to rely on the same step size for all tuples. *L-BFGS*, finally, improves slowly in comparison.

5.9.3 Scalability

In this experiment, we study the scalability of our implementation.

Dataset As before, we run on YAGO2 (see Appendix A.2.1). For tuples in $\mathcal{T} \backslash \mathcal{T}_l$, we uniformly draw synthetic probability values from $[0, 1]$.

Tuples and Labels In order to create labels, we run queries on YAGO2 and assign a synthetic target probability to their answers' lineage formulas.

In $P1$, we set $\mathcal{T}_l := \mathit{ActedIn} \cup \mathit{WasBornIn}$, which comprise $217,846$ tuples. Then, we employ the following deduction rules:

$$\begin{aligned}
\mathit{Movie}(M) &\leftarrow \mathit{ActedIn}(P, M)\\
\mathit{Creator}(P) &\leftarrow \mathit{ActedIn}(P_2, M) \wedge \mathit{Created}(P, M)\\
\mathit{Location}(L) &\leftarrow \mathit{WasBornIn}(P, L)\\
\mathit{Person}(P) &\leftarrow \mathit{WasBornIn}(P, L)\\
\mathit{Person2}(P) &\leftarrow \mathit{WasBornIn}(P, L) \wedge \mathit{IsLocatedIn}(L, L_2)
\end{aligned}$$

These rules result in $228,050$ lineage formulas. As the relations *ActedIn* and *WasBornIn* are employed several times, each tuple in $\mathcal{T}_l$ occurs in many labels.

In the second problem instance $P2$, we assign again $\mathcal{T}_l := \mathit{ActedIn} \cup \mathit{WasBornIn}$, but rely on different deduction rules inducing $79,600$ labeled lineage formulas:

$$\begin{aligned}
\mathit{Movie}(M) &\leftarrow \mathit{ActedIn}(P, M)\\
\mathit{Actor}(P) &\leftarrow \mathit{ActedIn}(P, M) \wedge \mathit{Created}(P_2, M)\\
\mathit{Location}(L) &\leftarrow \mathit{WasBornIn}(P, L)\\
\mathit{Person}(P) &\leftarrow \mathit{WasBornIn}(P, L) \wedge \mathit{LivesIn}(P_2, L)
\end{aligned}$$

We note that tuples of *Created* and *LivesIn* are not subject to learning, but have fixed probability values. Here, the challenge are the large lineage formulas caused by the deduction rule in the last line whose join involves many tuples.

Finally, in $P3$ we set $\mathcal{T}_l := \mathit{IsLocatedIn\_Transitive}$ having about $1.7 \cdot 10^6$ tuples. Furthermore, we deploy a single deduction rule which generates

459, 597 labeled lineage formulas:

$$\mathit{Location}(L) \leftarrow \mathit{IsLocatedIn\_Transitive}(L, L_2)$$

This is the largest learning problem instance. Nevertheless, we note that the generated lineage formulas are disjoint with respect to the tuples occurring therein. So the resulting learning problem instance is consistent.

Results Figure 5.7 contains the results of the three large learning problem instances $P1$ to $P3$, where we report the time spent on *Grounding* the lineage formulas, and in *Algorithm 7*, which had multi-threading enabled.

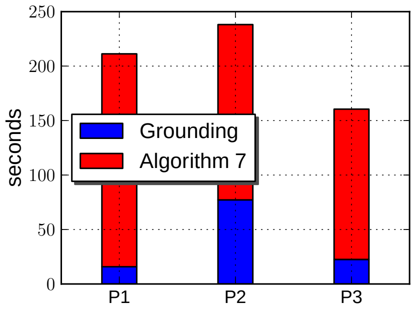


Figure 5.7: Scalability Experiment

Analysis Even for these large problem instances our algorithms succeed very fast, proving the scalability of our approach. In detail, the *Grounding* time is determined by the number of labels $|\mathcal{L}|$ and the size, i.e. $T(\phi)$, of the lineage formulas being instantiated. In $P2$ the lineage formulas involve on average about 60 tuples (see Table A.1), which explains the long grounding time. *Algorithm 7* is faster on consistent instances, i.e. $P3$, even though $|\mathcal{T}_l|$ is very large in this instance. In $P1$ the labeled lineage formulas heavily overlap, i.e. each tuple is contained in many labels, which causes the longer runtime of *Algorithm 7*.

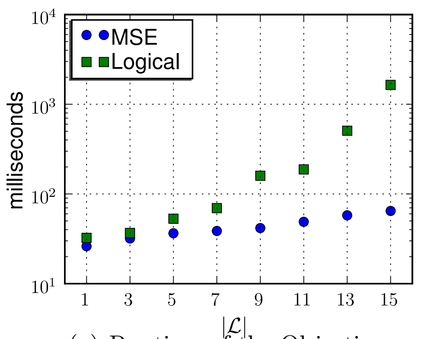
5.9.4 Algorithm Analysis

In the last series of experiments we investigate (1) the runtime behavior of the two objectives of Section 5.6, (2) the impact of varying the error rates ϵ_{abs} and ϵ_{rel} on both runtime and quality, and (3) the ability of Algorithm 7 to find global optima.

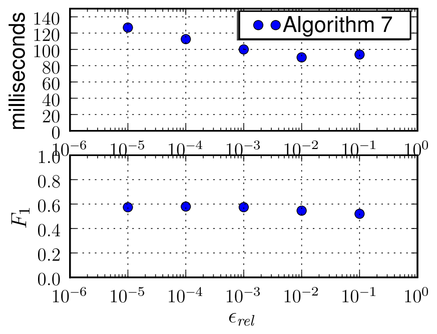
Objectives

We run Algorithm 7 once with the *Logical* objective (see Definition 5.3) and once with the mean squared error *(MSE)* objective (see Definition 5.4). The synthetic data is created analogously to the experiment of Figure 5.6(a).

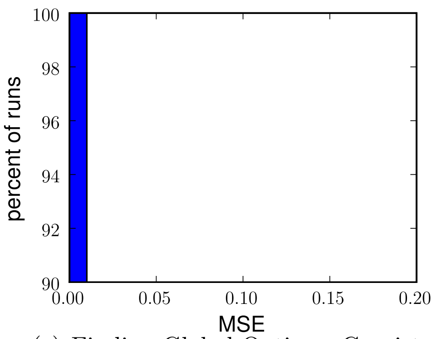
Results and Analysis Already on tiny instances of up to 15 labels as in Figure 5.8(a), the *Logical* objective slows down significantly in comparison to *MSE*, due to the expensive probability computations of Equation (5.2).



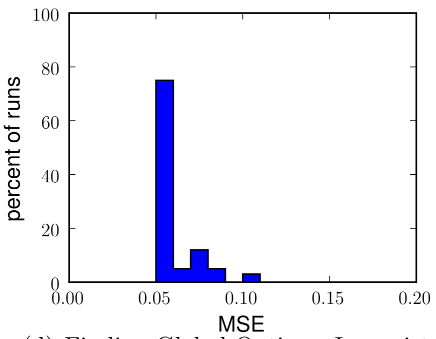
(a) Runtime of the Objectives



(b) Varying the error rates ϵ_{abs} and ϵ_{rel}



(c) Finding Global Optima: Consistent



(d) Finding Global Optima: Inconsistent

Figure 5.8: Algorithm Analysis Experiments

Varying Error Rates

We reuse the setup of Figure 5.5(a) with twice as many negative labels as positive ones, that is, $|\mathcal{L}| = 339$. Then, we vary ϵ_{rel} from 10^{-1} to 10^{-5}, where we set $\epsilon_{abs} := \frac{\epsilon_{rel}}{100}$. In comparison, all other experiments had fixed $\epsilon_{rel} = 10^{-4}$ and $\epsilon_{abs} = 10^{-6}$. In Figure 5.8(b) we display both runtime and quality in terms of the F_1 measure for each setting of the error rates.

Discussion Figure 5.8(b) shows that decreasing the error-rates increases the runtime, as expected, where however the impact is small. Especially, if we compare the overall runtime presented in Figure 5.5(a), which is about three seconds, we realize that Algorithm 7 consumes only a tiny fraction of the total runtime. With respect to quality, the F_1 measure remains mostly stable, with an exception for $\epsilon_{rel} > 10^{-3}$.

Finding Global Optima

As a last experiment, we rerun Algorithm 7 repeatedly on the problem of Figure 5.5(a) and compare the resulting mean squared error (MSE). In particular, we focus on two instances, where the first is *consistent*, since we rely on the $|\mathcal{L}| = 113$ positive labels only, and the second is *inconsistent* featuring $|\mathcal{L}| = 339$ both positive and negative labels. In Figures 5.8(c) and 5.8(d), we depict histograms of the resulting mean squared error (MSE) of 100 runs on the consistent and inconsistent instance, respectively.

Discussion In the consistent case Algorithm 7 always terminates in solutions, which are extremely close to the global optimum of 0.0 (see Figure 5.8(c)). The inconsistent instance, however, has a more rigged optimization landscape with global optima of values larger than 0.0. In this case, Algorithm 7 converges very close to a (probably) global optimum in 78% of the runs.

5.10 Summary and Outlook

Contribution We presented the first work on learning tuple probabilities in tuple-independent probabilistic databases. As input we consider lineage formulas labeled by target probabilities. These lineage formulas can result from a number of processes including query answers and constraints on the probabilistic database. We analyzed the theoretical properties of this learning problem by characterizing its complexity and its solution space. Furthermore, we cast the learning problem into an optimization problem which allows gradient-based solutions, and we presented an algorithm that solves it for database-like instance sizes. Finally, we investigated the relationship to other problems including conditioning probabilistic databases and deriving probabilistic databases from incomplete databases.

Future Directions For future work, we see numerous promising directions. Studying tractable subclasses of the learning problem or dropping the tuple-independence assumption would improve our theoretical understanding. Also, other formulations of the optimization problem, such as expectation maximization, might yield quality gains. Other valuable targets lie in the creation of a large, publicly available probabilistic database benchmark and the application of the learning problem to a broader range of related problems.

Chapter 6

Implementation

The algorithms and definitions of Chapter 3 and Chapter 5 were implemented in a system called *TPDBlearn* which is publicly available for download[1]. The present chapter hence discusses design decisions and implementation issues of the software. Within the variety of facets, we focus on three key building-blocks, namely the encoding of a temporal probabilistic database in a relational database (Section 6.1), the efficient modeling of propositional lineage in memory (Section 6.2), and how to implement the large scale learning algorithm (Section 6.4). Even though it is not contained in the *TPDBlearn* software, we cover the implementation of first-order lineage in Section 6.3. Throughout this chapter, we assume familiarity with database design [65] and object-oriented programming [53, 156].

6.1 Database Layout

The main advantage of deploying a database in the back-end is to leverage its capacities on conjunctive queries, which we leverage for both bodies of deduction rules and queries.

Tables All extensional relations are stored in the relational database back-end, for which we employed PostgreSQL[2]. We created one table for each extensional relation, where we depict two generic ones, called *Temporal-Relation1* and *TemporalRelation2* in the diagram of Figure 6.1. The first N columns represent the non-temporal arguments followed by the time-interval, which is encoded as begin and end time-point. Finally, the last column is a lineage id, which points to a separate table holding all lineage ids and the respective probability value of this tuple. The advantage of this setup is that intensional relations come with exactly the same table layout as extensional relations. The only difference is that their lineage id refers

[1] `http://people.mpi-inf.mpg.de/~mdylla/tpdblearn`

[2] `http://www.postgresql.org/`

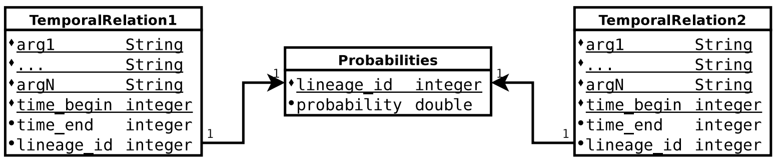


Figure 6.1: Database Layout

to a datastructure in memory, rather than in database. Extensional tables always reside in the database, whereas intensional tables are written on the fly while grounding and are deleted once the query terminates.

Indices On all extensional relations we create indices in all permutations (as far as possible) of the non-temporal arguments always followed by both the begin time-point and the lineage id. Additionally, we instantiate one index over the lineage id only. In contrast, intensional tables come with no index at all, since we mostly read them only once.

Querying We query the database by rewriting the deduction rules and queries to SQL [65], which we illustrate by the two deduction rules of Example 3.7. The first rule defines marriages to start with the wedding and end with the divorce:

$$\mathit{Marriage}^T(P_1, P_2, T_{b,1}, T_{e,2}) \leftarrow \left(\begin{array}{c} \mathit{Wedding}^T(P_1, P_2, T_{b,1}, T_{e,1}) \wedge \\ \mathit{Divorce}^T(P_1, P_2, T_{b,2}, T_{e,2}) \wedge \\ T_{e,1} <^T T_{b,2} \end{array} \right)$$

In SQL we write it as follows:

```
SELECT
        t0.arg0, t0.arg1,
        t0.time_begin, t1.time_end,
        t0.lineage_id, t1.lineage_id
FROM
        Wedding AS t0
JOIN
        Divorce AS t1
ON      t0.arg0 = t1.arg0 AND t0.arg1 = t1.arg1
WHERE
        t0.time_end < t1.time_begin;
```

The first two columns of the result of the query above contain the constants instantiating P_1 and P_2, followed by the limits for the time-interval, and finally two lineage ids. These lineage ids indicate the lineage of the tuples, which the current result originated from. Also, across different result rows, a change in the lineage ids signalizes a new grounding, i.e. of Definition 2.4. It is worthwhile to mention that the results of the above query are not deduplicated, hence Algorithm 4 has to be applied afterwards. The second deduction rule expresses that non-divorced couples remain married:

$$\mathit{Marriage}^T(P_1, P_2, T_{b,1}, t_{max}) \leftarrow \begin{pmatrix} \mathit{Wedding}^T(P_1, P_2, T_{b,1}, T_{e,1}) \wedge \\ \neg \mathit{Divorce}(P_1, P_2) \end{pmatrix}$$

In SQL the deduction rule is encoded as follows:

```
SELECT
        t0.arg0, t0.arg1,
        t0.time_begin,
        t0.lineage_id, t1.lineage_id
FROM
        Wedding AS t0
LEFT OUTER JOIN
        Divorce AS t1
ON      t0.arg0 = t1.arg0 AND t0.arg1 = t1.arg1;
```

The major difference to the previous query is the usage of an outer join, which can entail that the second lineage id is null. From a logical perspective this replaces the *Divorce* literal by *false*.

6.2 Propositional Lineage

Since probability computations are the bottleneck in probabilistic databases, an efficient lineage implementation can make the difference between a query being executed in reasonable time or not.

Design Decisions As a first choice, we keep lineage at all times *in memory*, such that processing is sped up in comparison to disk-based approaches. Second, for all lineage types we keep *pools of objects*, such that constructing a lineage formula is done by merely setting attribute values.

Lineage Classes In detail, the classes are depicted in the unified modeling language (UML) diagram[3] of Figure 6.2. Lineage is represented as an abstract class called *AbstractLineage* whose attributes contain an array *subLineage* for the subformulas and a map *tupleIdsToCount* which points from tuple ids to the number of times, this tuple occurs in a subformula. Hence, Shannon expansions (see Section 2.2.5) are characterized by tuple ids whose value in the map is greater than one. Also, when initializing a lineage object, the new map results from counting how often tuple ids occur in the key sets of the maps of the subformulas. The different types of lineage, logical and, or, not, and tuple identifiers are subclasses of *AbstractLineage*. Here, only *LineageTuple* sticks out, which stores the tuple id along with the probability of the tuple. With respect to probability computations, we implemented

[3]http://www.omg.org/spec/UML/

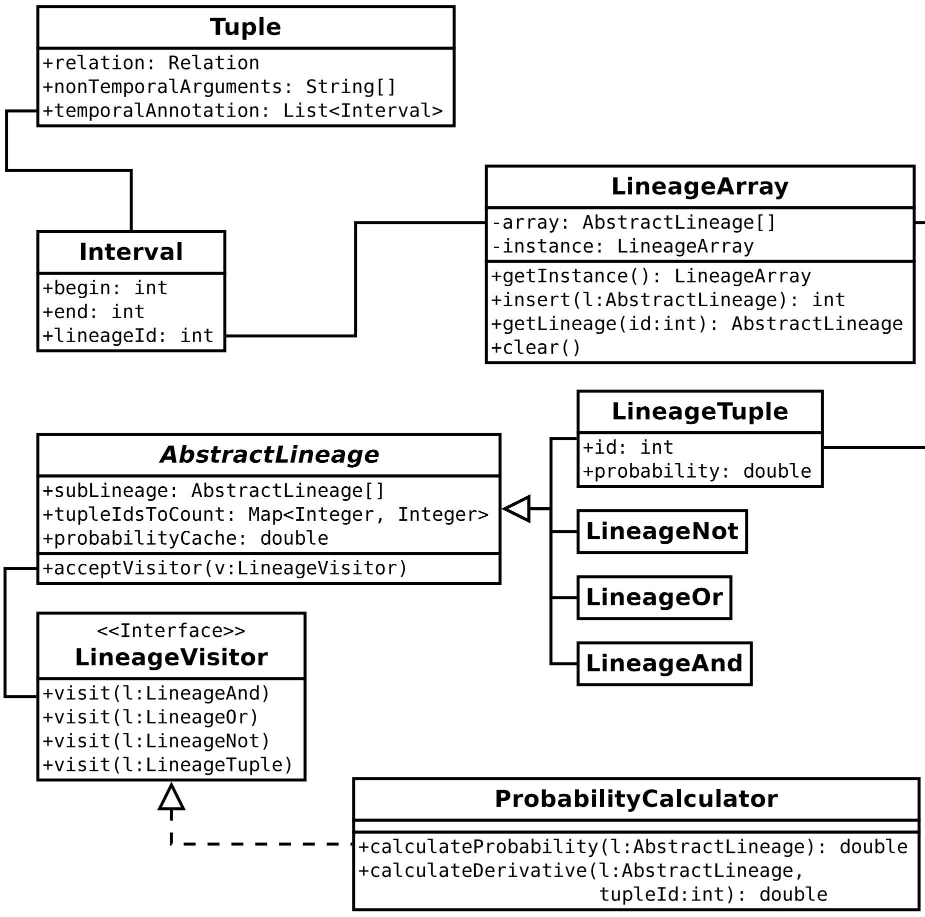

Tuple
+relation: Relation
+nonTemporalArguments: String[]
+temporalAnnotation: List<Interval>
Interval
+begin: int
+end: int
+lineageId: int
LineageArray
-array: AbstractLineage[]
-instance: LineageArray
+getInstance(): LineageArray
+insert(l:AbstractLineage): int
+getLineage(id:int): AbstractLineage
+clear()
LineageTuple
+id: int
+probability: double
AbstractLineage
+subLineage: AbstractLineage[]
+tupleIdsToCount: Map<Integer, Integer>
+probabilityCache: double
+acceptVisitor(v:LineageVisitor)
LineageNot
LineageOr
LineageAnd
<<Interface>>
LineageVisitor
+visit(l:LineageAnd)
+visit(l:LineageOr)
+visit(l:LineageNot)
+visit(l:LineageTuple)
ProbabilityCalculator
+calculateProbability(l:AbstractLineage): double
+calculateDerivative(l:AbstractLineage, tupleId:int): double

Figure 6.2: UML Diagram for Propositional Lineage

them by the visitor design pattern [53] on the lineage classes as captured by the interface *LineageVisitor* and its implementation *ProbabilityCalculator*.

In Memory Lineage The management of in-memory lineage is performed in the *LineageArray* class (see Figure 6.2), which for convenient access is implemented by the singleton design pattern [53]. The class stores one huge array referencing all active lineage formulas. The lower ids of the array coincide with the tuple ids of the database (see Figure 6.1). Hence, whenever we touch a tuple in database, we lazily load, i.e. in batches, the tuple probability from the *probabilities* table of Figure 6.1 and instantiate the corresponding *LineageTuple* objects. The higher ids of the array refer to lineage formulas which are created during grounding. When a query terminates, we simply return all lineage objects in the higher id range to their respective object pools.

Tuples and Time-Intervals Tuples, that is the *Tuple* class of Figure 6.2, are only instantiated to represent query results. Intermediate results reside just in the database. Besides the relation and the non-temporal arguments, the *Tuple* class holds a list of time-intervals, which in turn refer to a lineage id in the *LineageArray* each. This allows every interval to take a different probability.

Shape of Lineage Since we store only pointers to subformulas in the *AbstractLineage* class, the objects can represent lineage the form of a directed acyclic graph [25]. In Figure 6.3 we display the propositional lineage formulas of two answers which share subformulas. In this form caching

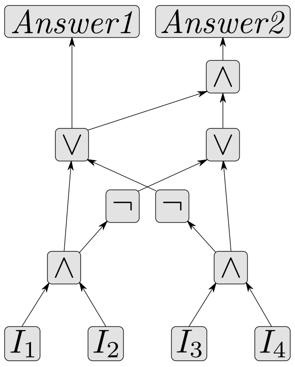


Figure 6.3: Propositional Lineage Example

of probabilities is very efficient, because lineage formulas are reused even among different answers. We do not compile the propositional formula to this form, but it can result from grounding.

6.3 First-Order Lineage

We next discuss the structure of an implementation of first-order lineage (see Section 4.4) which supports creation via SLD steps (see Definition 4.4) with scheduling. Attempting a correct implementation is a little bit more challenging than for propositional lineage of the previous section. We display the UML diagram of our first-order lineage implementation in Figure 6.4.

Connections to Algorithms To start with, the sets A_{top} and A_{cand} of Algorithm 6 hold instances of *LineageAnswer* which is a subclass of *Lineage-And*, since queries are conjunctions of literals. In addition, the scheduler of Algorithm 5 chooses instances of *LineageLiteral* to expand.

Variables As distinguished in Definition 4.4, variables can be of three types, namely *QueryVariable*, *ExistentialVariable*, and *UniversalVariable*,

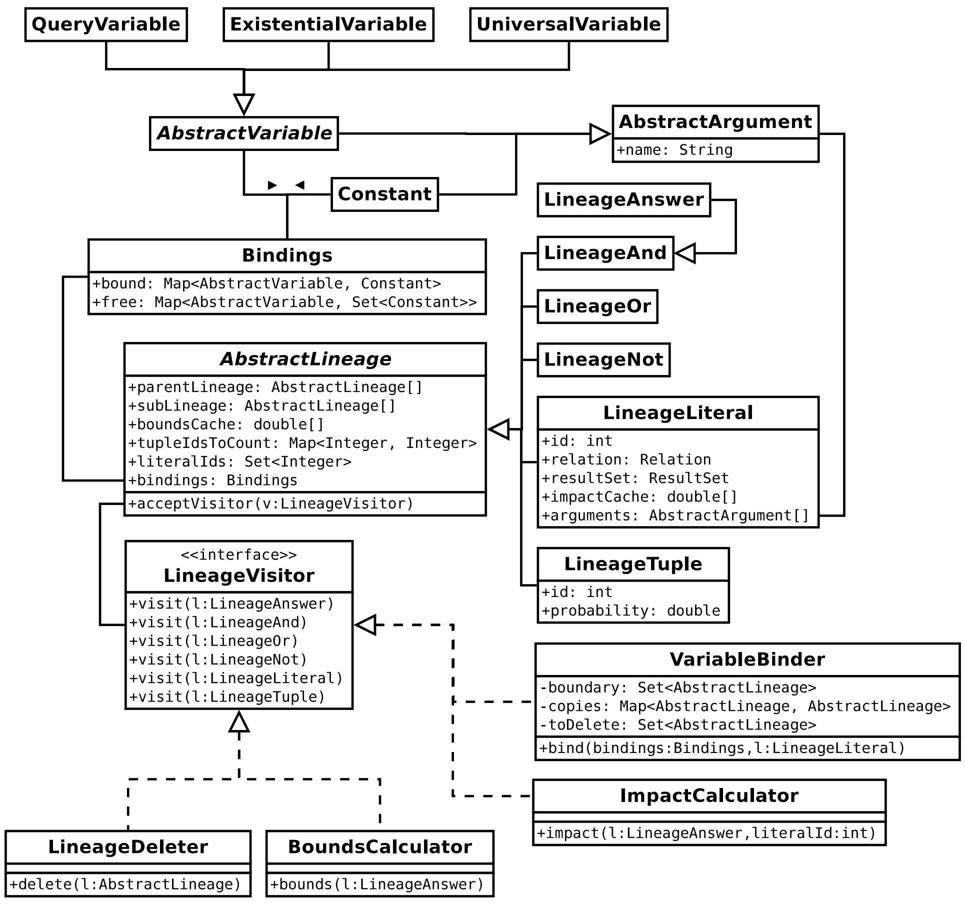


Figure 6.4: UML Diagram for First-Order Lineage

where the latter two are quantified. The class *Bindings* holds variables, which are either assigned to a constant, that is bound, or free. If they are free, then the set of constants (the value of the map) implicitly represents negative bindings being constants the variable may not be assigned to. This is necessary for implementing sorted input lists (see Section 4.6.1). Furthermore, we note that every lineage instance carries bindings, as they are an attribute of *AbstractLineage*. By this we capture the scope of each variable, which simply occurs in all lineage objects under the quantifier or query. Quantifiers are implicitly contained at the upper most lineage object which holds a variable.

Creating First-Order Lineage Instantiating the query and deduction rules is straight forward. The tedious part, is binding variables to constants, which can entail new answers, that is copying the lineage formula, or expanding the lineage formula via Equation (4.1). For this, we employ the visitor *VariableBinder*. It starts at the literal we are considering and moves along the scope of the variable (up and down via the *parentLineage* and *subLineage* attributes of *AbstractLineage*) until all of the scope of the variable is covered. Within this process, we rely on the attributes of *VariableBinder* to keep track of lineage objects that have already been copied (*copies*), lineage objects that must not be copied (*boundary*), and lineage objects which should be deleted (*toDelete*). Deletion is performed after binding the variables, and has to ensure that the resulting lineage formula is valid.

If you enjoy implementing complicated datastructures, we note that for lazy copying of first-order lineage, i.e. if new query answers are produced, all of the attributes of *AbstractLineage* except for *bindings* have to exist and to be managed separately for each query answer.

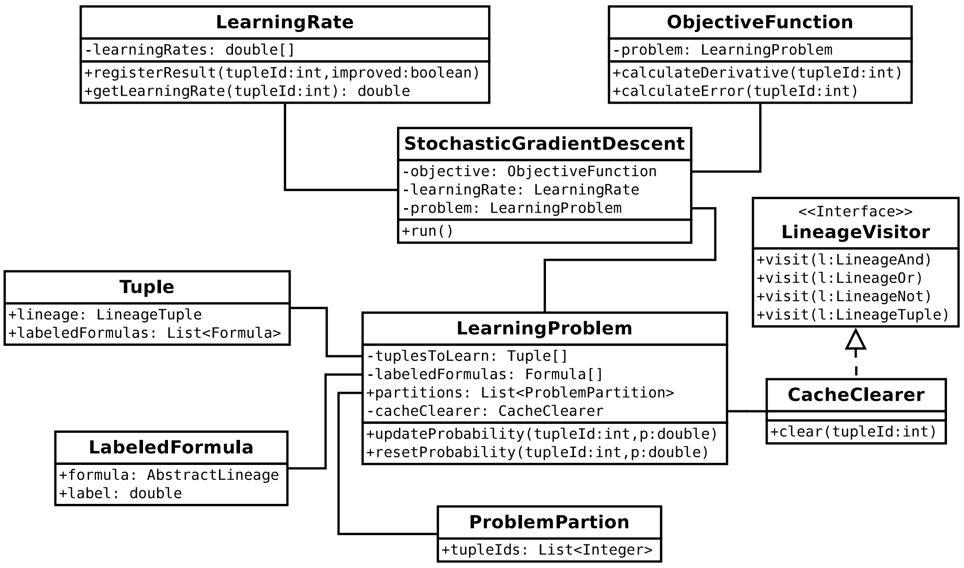
Sorted Input Lists To enable tuple by tuple reading from a relation, the next tuple to read is saved in every *LineageLiteral* instance in the attribute *resultSet*. Furthermore, as mentioned before, the set in *Bindings* attribute *free* represent constants a variable may not be assigned to. In combination these two features allow the implementation of Definition 4.9.

6.4 Learning Tuple Probabilities

In this section we present implementation details on Algorithm 7. The bottleneck are probability computations of the labeled lineage formulas, which are invoked repeatedly. To mitigate this effort, we employ the following set of strategies:

- We preprocess each labeled lineage formula by Algorithm 3.
- Moreover, to check whether the new probability value of I_l yielded an improvement in the objective (Line 11), we compute only probabilities of lineage formulas ϕ where I_l occurs in, that is $I_l \in \mathit{Tup}(\phi)$.
- Each lineage node has a probability cache (see Figure 6.2). When we update the probability of a tuple occurring in a lineage formula, we only recompute the path from the tuple to the root of the formula. All other probabilities within the formula are reused.
- We cache the probability of each labeled formula. When we decide to discard the new probability value of a tuple, i.e. in Line 16 of Algorithm 7, we reuse the old cached probability of all lineage formulas this tuple occurs in.
- Finally, if two tuples $I_l, I'_l \in \mathcal{T}_l$ are disjoint with respect to the labels' lineage formulas they occur in, that is $\{\phi_i \mid (\phi_i, l_i) \in \mathcal{L}, I_l \in \mathit{Tup}(\phi_i), I'_l \in \mathit{Tup}(\phi_i)\} = \emptyset$, then their probability values can be updated in parallel.

With respect to the implementation we depict its UML diagram in Figure 6.5. Algorithm 7 is modeled by several classes each corresponding to a



LearningRate
-learningRates: double[]
+registerResult(tupleId:int,improved:boolean)
+getLearningRate(tupleId:int): double
ObjectiveFunction
-problem: LearningProblem
+calculateDerivative(tupleId:int)
+calculateError(tupleId:int)
StochasticGradientDescent
-objective: ObjectiveFunction
-learningRate: LearningRate
-problem: LearningProblem
+run()
<<Interface>>
LineageVisitor
+visit(l:LineageAnd)
+visit(l:LineageOr)
+visit(l:LineageNot)
+visit(l:LineageTuple)
Tuple
+lineage: LineageTuple
+labeledFormulas: List<Formula>
LearningProblem
-tuplesToLearn: Tuple[]
-labeledFormulas: Formula[]
+partitions: List<ProblemPartition>
-cacheClearer: CacheClearer
+updateProbability(tupleId:int,p:double)
+resetProbability(tupleId:int,p:double)
CacheClearer
+clear(tupleId:int)
LabeledFormula
+formula: AbstractLineage
+label: double
ProblemPartion
+tupleIds: List<Integer>

Figure 6.5: UML Diagram for Learning Tuple Probabilities

component, i.e. the learning rate, the objective function, and the stochastic gradient descent implementation. The learning problem is captured in its own class where all tuples of $\mathcal{T}_l$ are objects of type *TupleToLearn* and all elements of $\mathcal{L}$ are represented by *LabeledFormula* objects. The learning problem instance consists of several partitions which can be computed in parallel.

Chapter 7

Conclusion

In modern settings, uncertain data is ubiquitous, since it can result from physical measurements, automated information extraction from unstructured data, incomplete data which features missing data entries, and inconsistent data, just to mention a few. On this kind of data, probabilistic databases have the potential to change the way we manage uncertain data as traditional relational databases did for deterministic data.

Contributions In this thesis, we advanced the state-of-the-art of probabilistic databases in several ways. First, in Chapter 3 we devised a closed and complete temporal probabilistic data model which allows us to cope with data being variable over time as well as uncertain. Then, in Chapter 4 we introduced an approach to query probabilistic databases for the top-k most probable answers. Our method was the first to support partially grounded query answers which were represented by first-order lineage. Next, in Chapter 5 we addressed the learning of tuple probabilities from a database perspective. Learning is a key building block for future probabilistic database engines since it enables creating, updating and cleaning of probabilistic databases. Finally, in Chapter 6 we presented implementation and design details on how to turn the above contributions into software.

Future Directions Regarding possible extensions and improvements of this work there are numerous opportunities. We present here the three most important directions.

- First of all, the independence assumption among database tuples is strong and can be replaced by more sophisticated correlations, such as *independent-disjoint* probabilistic databases [30] or *pc-tables* [60]. In this setting, however, further research is needed to extend our theoretical results.

- Common to all probabilistic database systems is the inherent $\#\mathcal{P}$-hardness of probability computations, which can result in non-interactive query runtimes. To limit the impact of this, a valuable research path is to consider more notions of *relaxed answer semantics*, such as focusing on rankings, or to develop more *approximation techniques* of answer probabilities which trade runtime for precision.

- As a final point, the introduction of first-order lineage calls for *lifted probability computations*, which would then compute probabilities of sets of query answers. This is a new direction in probabilistic databases in which we are about to see the first work appearing [12].

Appendix A

Supplementary Material

A.1 System Setup

All experiments are performed on an 8-core Intel Xeon 2.4 GHz with 48 GB of RAM. The presented algorithms are implemented in Java, and rely on a PostgreSQL 8.4[1] database as storage back-end. For more details on the implementation we refer the interested reader to Chapter 6. We always report average runtimes over warm disk caches by running each query or problem instance 4 times in a row, and report the average runtime of the latter 3 runs.

A.2 Datasets

In this section we elaborate on the datasets employed in experiments throughout the thesis.

A.2.1 YAGO

Yet Another Great Ontology (YAGO)[2] is a research project which creates a high-quality knowledge base. Its data is mined automatically from the infoboxes and categories of Wikipedia. The semantical classes originate from Wordnet[3]. YAGO comes in two major releases which are discussed next.

YAGO1

The original release of YAGO [138] comes with 82 manually defined relations, such as *BornIn*, *LivesIn*, *During*, or *SubClassOf* which in total hold about $85 \cdot 10^6$ tuples. Each tuple is annotated with a confidence, which is derived

[1] `http://www.postgresql.org/`
[2] `http://www.mpi-inf.mpg.de/yago-naga/yago/`
[3] `http://wordnet.princeton.edu/`

from manual quality assessments. Nonetheless, these values are the same for every entry in a relation. As this is a rather unrealistic setup for probabilistic databases we draw synthetic a probability from the uniform distribution for each tuple.

YAGO2

The successor version, YAGO2 [66], additionally contains location data. Also, it offers specific support for temporal annotations, and entity names are represented in many languages. This leads to an increase in size. YAGO2 comprises about 110 relations holding about $224.4 \cdot 10^6$ tuples of which $1.6 \cdot 10^6$ have temporal annotations. Again, confidences are given, but they are constant for all tuples in a relation. So, we employ synthetic probabilities in the experiments.

A.2.2 IMDB

The international movie database (IMDB) webpage[4] features data on nearly every existing movie. We downloaded their dataset and extracted the six relations *Directed*, *Acted*, *Edited*, *Produced*, *Written*, and *HasCategory* summing up to $26 \cdot 10^6$ tuples. As the data is deterministic, we synthetically sampled tuple probabilities from the uniform distribution.

A.2.3 UW-CSE

The UW-CSE dataset[5], comprises a database describing the computer science department of University of Washington via the following relations: *AdvisedBy*, *CourseLevel*, *HasPosition*, *InPhase*, *Professor*, *ProjectMember*, *Publication*, *Student*, *TaughtBy*, *Ta* (teaching assistant), and *YearsInProgram*. In total, there are 2,161 tuples distributed to these relations. Thus, the dataset is rather small. Moreover, it is split into five sub-departments, namely artificial intelligence, graphics, language, systems, and theory.

A.2.4 PRAVDA

This dataset[6] contains temporal facts in the sports and celebrities domain which are automatically extracted from text. The considered relation are *WorksForClub* and *Marriage* where *begin*, *during* and *end* observations are distinguished (see [153]). For example, joining a soccer club marks a *begin* event. In total the dataset comprises about 25,000 fact candidates which are extracted using about 12,500 textual patterns. The distribution of pattern-fact co-occurrence is heavily skewed, i.e. few patterns create most facts.

[4]`http://www.imdb.com`

[5]`http://alchemy.cs.washington.edu/data/uw-cse/`

[6]`http://www.mpi-inf.mpg.de/yago-naga/pravda/`

A.3 Deduction Rules and Constraints

A.3.1 Temporal Probabilistic Data Model Experiments

We here present all deduction rules, all constraints and additional results of the experiments described in Section 3.7.

Temporal Information Extraction

This section list all deduction rules and constraints of the temporal information extraction experiment of Section 5.9.1. The experiment is conducted on the PRAVDA dataset (see Appendix A.2.4).

Temporal Deduction rules Since each fact might have been found in several documents, we aggregate the different extractions as distinguished by their *Id*s to duplicate-free tuples:

$$\begin{aligned}
\mathit{AttendedSchool}^T(P, S, T_b, T_e) &\leftarrow \mathit{AttendedSchool}^T(P, S, Id, T_b, T_e)\\
\mathit{Born}^T(P, T_b, T_e) &\leftarrow \mathit{Born}^T(P, Id, T_b, T_e)\\
\mathit{Died}^T(P, T_b, T_e) &\leftarrow \mathit{Died}^T(P, Id, T_b, T_e)\\
\mathit{Divorce}^T(P_1, P_2, T_b, T_e) &\leftarrow \mathit{Divorce}^T(P_1, P_2, Id, T_b, T_e)\\
\mathit{Founded}^T(P, O, T_b, T_e) &\leftarrow \mathit{Founding}^T(P, O, Id, T_b, T_e)\\
\mathit{GraduatedFrom}^T(P, S, T_b, T_e) &\leftarrow \mathit{GraduatedFrom}^T(P, S, Id, T_b, T_e)\\
\mathit{IsDating}^T(P_1, P_2, T_b, T_e) &\leftarrow \mathit{IsDating}^T(P_1, P_2, Id, T_b, T_e)\\
\mathit{MovedTo}^T(P, L, T_b, T_e) &\leftarrow \mathit{MovedTo}^T(P, L, Id, T_b, T_e)\\
\mathit{Wedding}^T(P_1, P_2, T_b, T_e) &\leftarrow \mathit{Wedding}^T(P_1, P_2, Id, T_b, T_e)
\end{aligned}$$

Then, we deduce the $\mathit{Marriage}^T$ relation as in Example 3.7:

$$\begin{aligned}
\mathit{Marriage}^T(P_1, P_2, T_b, T'_e) &\leftarrow \left(\begin{array}{c}\mathit{Wedding}^T(P_1, P_2, T_b, T_e)\wedge\\ \mathit{Divorce}^T(P_1, P_2, T'_b, T'_e)\wedge\end{array} T_e <^T T'_b\right)\\
\mathit{Marriage}^T(P_1, P_2, T_b, t_{max}) &\leftarrow \left(\begin{array}{c}\mathit{Wedding}^T(P_1, P_2, T_b, T_e)\wedge\\ \neg\mathit{Divorce}(P_1, P_2)\end{array}\right)\\
\mathit{Divorce}(P_1, P_2) &\leftarrow \mathit{Divorce}^T(P_1, P_2, T_b, T_e)
\end{aligned}$$

After grounding all the above deduction rules we query for the relations Born^T, Died^T, $\mathit{Founded}^T$, $\mathit{GraduatedFrom}^T$, $\mathit{IsDating}^T$, $\mathit{MovedTo}^T$, and $\mathit{Marriage}^T$.

Constraints We manually designed the constraints by measuring their effect on the precision-recall values in the training set. Our constraints can be divided into three groups, namely irreflexive relations, and precedence and disjointness among relations.

We achieved improved results by constraining $\mathit{Divorce}^T$ to be *irreflexive*:

$$\neg(\mathit{Divorce}^T(P_1, P_2, T_b, T_e) \wedge P_1 = P_2)$$

The prime target for *temporal precedence* constraints is the birth date, which should occur before any other event in the life of a person:

$$\begin{gathered}
\neg(\mathit{Born}^T(P_1,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_2,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{Born}^T(P_1,T_b,T_e) \wedge \mathit{AttendedSchool}^T(P_1,S,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{Born}^T(P_1,T_b,T_e) \wedge \mathit{Founded}^T(P_1,O,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{Born}^T(P_1,T_b,T_e) \wedge \mathit{GraduatedFrom}^T(P_1,S,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{Born}^T(P_1,T_b,T_e) \wedge \mathit{IsDating}^T(P_1,P_2,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{Born}^T(P_2,T_b,T_e) \wedge \mathit{IsDating}^T(P_1,P_2,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{Born}^T(P_1,T_b,T_e) \wedge \mathit{MovedTo}^T(P_1,L,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{Born}^T(P_1,T_b,T_e) \wedge \mathit{Wedding}^T(P_1,P_2,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{Born}^T(P_2,T_b,T_e) \wedge \mathit{Wedding}^T(P_1,P_2,T'_b,T'_e) \wedge T'_b <^T T_e)
\end{gathered}$$

Also, the date of death should occur after any other fact relating to a person:

$$\begin{gathered}
\neg(\mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Died}^T(P_2,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{Founded}^T(P_1,O,T_b,T_e) \wedge \mathit{Died}^T(P_1,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{GraduatedFrom}^T(P_1,S,T_b,T_e) \wedge \mathit{Died}^T(P_1,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{IsDating}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Died}^T(P_1,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{IsDating}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Died}^T(P_2,T'_b,T'_e) \wedge T'_b <^T T_e)\\
\neg(\mathit{MovedTo}^T(P_1,L,T_b,T_e) \wedge \mathit{Died}^T(P_1,T'_b,T'_e) \wedge T'_b <^T T_e)
\end{gathered}$$

Finally, we require the *IsDating*^{T} relation to take place before the couple is married:

$$\neg(\mathit{IsDating}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_2,T'_b,T'_e) \wedge T'_b <^T T_e)$$

We enforce marriages of a person P_1 to two different persons P_2 and P_3 to be *temporally disjoint* by writing:

$$\begin{gathered}
\neg\begin{pmatrix} \mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_3,T'_b,T'_e) \\ \wedge P_2 \neq P_3 \wedge T_b <^T T'_b \wedge T'_b <^T T_e \end{pmatrix}\\
\neg\begin{pmatrix} \mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_3,T'_b,T'_e) \\ \wedge P_2 \neq P_3 \wedge T_b <^T T'_e \wedge T'_e <^T T_e \end{pmatrix}\\
\neg\begin{pmatrix} \mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_3,T'_b,T'_e) \\ \wedge P_2 \neq P_3 \wedge T'_b <^T T_b \wedge T_b <^T T'_e \end{pmatrix}\\
\neg\begin{pmatrix} \mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_3,T'_b,T'_e) \\ \wedge P_2 \neq P_3 \wedge T'_b <^T T_e \wedge T_e <^T T'_e \end{pmatrix}
\end{gathered}$$

Now, we give the same constraints, but with exchanged order of arguments:

$$\begin{gathered}
\neg\begin{pmatrix} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T'_b,T'_e) \\ \wedge P_2 \neq P_3 \wedge T_b <^T T'_b \wedge T'_b <^T T_e \end{pmatrix}\\
\neg\begin{pmatrix} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T'_b,T'_e) \\ \wedge P_2 \neq P_3 \wedge T_b <^T T'_e \wedge T'_e <^T T_e \end{pmatrix}\\
\neg\begin{pmatrix} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T'_b,T'_e) \\ \wedge P_2 \neq P_3 \wedge T'_b <^T T_b \wedge T_b <^T T'_e \end{pmatrix}\\
\neg\begin{pmatrix} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T'_b,T'_e) \\ \wedge P_2 \neq P_3 \wedge T'_b <^T T_e \wedge T_e <^T T'_e \end{pmatrix}
\end{gathered}$$

We continue by restricting that married persons cannot date other persons during their marriage:

$$\begin{array}{l}
\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{IsDating}^T(P_1,P_3,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b <^T T_b' \wedge T_b' <^T T_e \end{array}\right) \\
\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{IsDating}^T(P_1,P_3,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b <^T T_e' \wedge T_e' <^T T_e \end{array}\right) \\
\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{IsDating}^T(P_1,P_3,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b' <^T T_b \wedge T_b <^T T_e' \end{array}\right) \\
\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{IsDating}^T(P_1,P_3,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b' <^T T_e \wedge T_e <^T T_e' \end{array}\right) \\
\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{IsDating}^T(P_3,P_1,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b <^T T_b' \wedge T_b' <^T T_e \end{array}\right) \\
\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{IsDating}^T(P_3,P_1,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b <^T T_e' \wedge T_e' <^T T_e \end{array}\right) \\
\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{IsDating}^T(P_3,P_1,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b' <^T T_b \wedge T_b <^T T_e' \end{array}\right) \\
\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{IsDating}^T(P_3,P_1,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b' <^T T_e \wedge T_e <^T T_e' \end{array}\right)
\end{array}$$

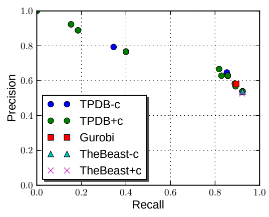
This concludes the constraints of the temporal information extraction experiment.

Precision-Recall Plots In Figure A.1 we display the missing six detailed precision-recall plots, where the first two are available in Figure 3.4(a). Each plot is created by varying θ_p of Equation (3.7) such that the different precision-recall pairs are induced.

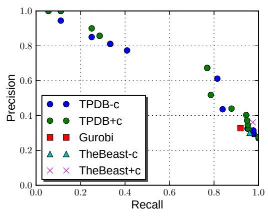
Many Constraints

Next, we give all deduction rules and constraints from query $Q5$ of Section 3.7.3. The deduction rules capture temporal knowledge about persons:

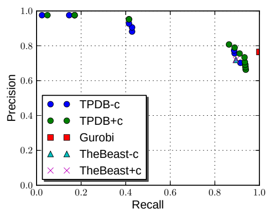
$$\begin{array}{rl}
\mathit{Born}^T(P,T_b,T_e) \leftarrow & \mathit{WasBornOnDate}^T(Id,P,T_b,T_e) \\
\mathit{Died}^T(P,T_b,T_e) \leftarrow & \mathit{DiedOnDate}^T(Id,P,T_b,T_e) \\
\mathit{Divorce}^T(P,Y,T_b,T_e) \leftarrow & \left(\begin{array}{c} \mathit{IsMarriedTo}(Id,P,Y) \\ \wedge \mathit{OccursUntil}^T(Id_2,Id,T_b,T_e) \end{array}\right) \\
\mathit{HasChild}^T(P_1,P_2,T_b,T_e) \leftarrow & \left(\begin{array}{c} \mathit{HasChild}(Id,P_1,P_2) \\ \wedge \mathit{WasBornOnDate}^T(Id_2,P_2,T_b,T_e) \end{array}\right) \\
\mathit{Wedding}^T(P_1,P_2,T_b,T_e) \leftarrow & \left(\begin{array}{c} \mathit{IsMarriedTo}(Id,P_1,P_2) \\ \wedge \mathit{OccursSince}^T(Id_2,Id,T_b,T_e) \end{array}\right) \\
\mathit{Marriage}^T(P_1,P_2,T_b,T_e') \leftarrow & \left(\begin{array}{c} \mathit{Wedding}^T(P_1,P_2,T_b,T_e) \\ \wedge \mathit{Divorce}^T(P_1,P_2,T_b',T_e') \end{array}\right)
\end{array}$$



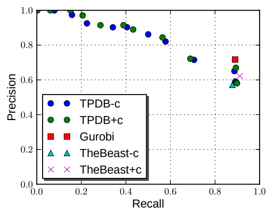
(a) GraduatedFrom



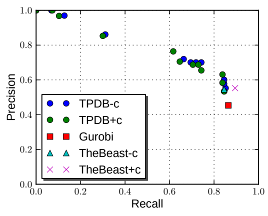
(b) Died



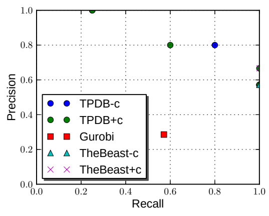
(c) Born



(d) MovedTo



(e) AttendedSchool



(f) Founded

Figure A.1: Precision and Recall (varying θ_p)

We query for the relations *Born*, *Died*, *HasChild*, and *Marriage*, which are restricted by precedence and disjointness constraints. The precedence constraints look as follows:

$$\begin{array}{c}
\neg(\mathit{Born}^T(P_1,T_b,T_e) \wedge \mathit{HasChild}^T(P_1,P_2,T_b',T_e') \wedge T_b' <^T T_e)\\
\neg(\mathit{Born}^T(P_1,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_2,T_b',T_e') \wedge T_b' <^T T_e)\\
\neg(\mathit{Born}^T(P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_2,T_b',T_e') \wedge T_b' <^T T_e)\\
\neg(\mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Died}^T(P_1,T_b',T_e') \wedge T_b' <^T T_e)\\
\neg(\mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Died}^T(P_2,T_b',T_e') \wedge T_b' <^T T_e)
\end{array}$$

Furthermore, as disjointness constraints we utilize:

$$\begin{array}{c}
\neg\left(\begin{array}{c}\mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_3,T_b',T_e')\\ \wedge P_2 \neq P_3 \wedge T_b <^T T_b' \wedge T_b' <^T T_e\end{array}\right)\\
\neg\left(\begin{array}{c}\mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_3,T_b',T_e')\\ \wedge P_2 \neq P_3 \wedge T_b <^T T_e' \wedge T_e' <^T T_e\end{array}\right)\\
\neg\left(\begin{array}{c}\mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_3,T_b',T_e')\\ \wedge P_2 \neq P_3 \wedge T_b' <^T T_b \wedge T_b <^T T_e'\end{array}\right)\\
\neg\left(\begin{array}{c}\mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_1,P_3,T_b',T_e')\\ \wedge P_2 \neq P_3 \wedge T_b' <^T T_e \wedge T_e <^T T_e'\end{array}\right)\\
\neg\left(\begin{array}{c}\mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T_b',T_e')\\ \wedge P_2 \neq P_3 \wedge T_b <^T T_b' \wedge T_b' <^T T_e\end{array}\right)\\
\neg\left(\begin{array}{c}\mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T_b',T_e')\\ \wedge P_2 \neq P_3 \wedge T_b <^T T_e' \wedge T_e' <^T T_e\end{array}\right)\\
\neg\left(\begin{array}{c}\mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T_b',T_e')\\ \wedge P_2 \neq P_3 \wedge T_b' <^T T_b \wedge T_b <^T T_e'\end{array}\right)
\end{array}$$

$$\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_1,P_2,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b' <^T T_e \wedge T_e <^T T_e' \end{array}\right)$$
$$\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b <^T T_b' \wedge T_b' <^T T_e \end{array}\right)$$
$$\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b <^T T_e' \wedge T_e' <^T T_e \end{array}\right)$$
$$\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b' <^T T_b \wedge T_b <^T T_e' \end{array}\right)$$
$$\neg\left(\begin{array}{c} \mathit{Marriage}^T(P_2,P_1,T_b,T_e) \wedge \mathit{Marriage}^T(P_3,P_1,T_b',T_e') \\ \wedge P_2 \neq P_3 \wedge T_b' <^T T_e \wedge T_e <^T T_e' \end{array}\right)$$

A.3.2 Top-k Query Processing Experiments

Scheduling

Here, we list the deduction rules and queries of the experiments of Figure 4.6(c). For $Q9$, having one level of deduction rules, the deduction rules are:

$$\begin{array}{rl} \mathit{Query}(X,Y) \leftarrow & \mathit{DiedIn}(X,Y) \wedge \mathit{BornIn}(X,Y) \\ \mathit{Query}(X,Y) \leftarrow & \exists Z\ \mathit{LivesIn}(X,Y) \wedge \mathit{IsMarriedTo}(Z,Y) \\ \mathit{Query}(X,Y) \leftarrow & \mathit{IsCitizenOf}(X,Y) \\ \mathit{Query}(X,Y) \leftarrow & \mathit{PoliticianOf}(X,Y) \end{array}$$

which are queried by $\mathit{Query}(X, \mathit{Constant})$ using 100 constants. The next query pattern $Q10$ features two levels of deduction rules

$$\begin{array}{rl} \mathit{Query}(X,Y) \leftarrow & \exists Z\ \mathit{IsCitizenOf}(X,Y) \wedge \mathit{DiedOnDate}(X,Z) \\ \mathit{Query}(X,Y) \leftarrow & \mathit{Subquery}(X,Y) \\ \mathit{Query}(X,Y) \leftarrow & \mathit{BornIn}(X,Y) \\ \mathit{Subquery}(X,Y) \leftarrow & \mathit{DiedIn}(X,Y) \\ \mathit{Subquery}(X,Y) \leftarrow & \mathit{LivesIn}(X,Y) \\ \mathit{Query}(X,Y) \leftarrow & \exists Z\ \mathit{PoliticianOf}(X,Y) \wedge \mathit{HasChild}(X,Z) \end{array}$$

and is initiated by $\mathit{Query}(X, \mathit{Constant})$. Finally, $Q11$ encodes three levels of deduction rules

$$\begin{array}{rl} \mathit{Query}(X,Y) \leftarrow & \exists Z\ \mathit{LivesIn}(X,Z) \wedge \mathit{DiedOnDate}(X,Y) \\ \mathit{Query}(X,Y) \leftarrow & \mathit{Subquery1}(X,Y) \\ \mathit{Subquery1}(X,Y) \leftarrow & \exists Z \mathit{BornOnDate}(X,Y) \wedge \mathit{BornIn}(X,Z) \\ \mathit{Subquery1}(X,Y) \leftarrow & \mathit{Subquery2}(X,Y) \\ \mathit{Subquery2}(X,Y) \leftarrow & \mathit{CreatedOnDate}(X,Y) \\ \mathit{Subquery2}(X,Y) \leftarrow & \mathit{WrittenInYear}(X,Y) \end{array}$$

which are invoked by $\mathit{Query}(X, \mathit{Constant})$. We note that the temporal arguments are treated as strings here, i.e. the 100 constants are all strings representing a date.

A.3.3 Learning Tuple Probabilities Experiments

In this section we present the full set of deduction rules inducing the labeled lineage formulas as well as the definition of $\mathcal{T}$ and $\mathcal{T}_l$ for each of the experiments of Section 5.9, where the information was presented only partially. We start by providing statistics on all learning problem instances of the experiments as depicted in Table A.1, where we calculate *Avg.* $Tup(\phi)$ as $\frac{1}{|\mathcal{L}|}\sum_{(\phi_i,l_i)\in\mathcal{L}} |Tup(\phi_i)|$.

Statistical Relational Learning Setting

The database contains all relations from the UW-CSE dataset (see Appendix A.2.3) as well as the *Rules* relation, which reads as:

$$\mathcal{T} := \begin{array}{c} \mathit{CourseLevel} \cup \mathit{HasPosition} \cup \mathit{InPhase} \cup \mathit{Professor} \cup \mathit{ProjectMember} \\ \cup \mathit{Publication} \cup \mathit{Student} \cup \mathit{TaughtBy} \cup \mathit{Ta} \cup \mathit{YearsInProgram} \cup \mathit{Rules} \end{array}$$

Then, we assign $\mathcal{T}_l := \mathit{Rules}$, which is also the only uncertain relation. The 49 automatically created rules in its full version are:

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{Ta}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{Ta}(D,C,P_2,\mathit{Te})\wedge \end{array} \mathit{Rules}(0) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{Ta}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{TaughtBy}(D,C,P_2,\mathit{Te})\wedge \end{array} \mathit{Rules}(1) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{Ta}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{Publication}(D,\mathit{Ti},P_2)\wedge \end{array} \mathit{Rules}(2) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{Ta}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{YearsInProgram}(D,P_2,Y)\wedge \end{array} \mathit{Rules}(3) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{Ta}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{HasPosition}(D,P_2,\mathit{Po})\wedge \end{array} \mathit{Rules}(4) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{Ta}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{InPhase}(D,P_2,\mathit{Ph})\wedge \end{array} \mathit{Rules}(5) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{Ta}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{ProjectMember}(D,\mathit{Pr},P_2)\wedge \end{array} \mathit{Rules}(6) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{TaughtBy}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{Ta}(D,C,P_2,\mathit{Te})\wedge \end{array} \mathit{Rules}(7) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{TaughtBy}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{TaughtBy}(D,C,P_2,\mathit{Te})\wedge \end{array} \mathit{Rules}(8) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{TaughtBy}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{Publication}(D,\mathit{Ti},P_2)\wedge \end{array} \mathit{Rules}(9) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{TaughtBy}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{YearsInProgram}(D,P_2,Y)\wedge \end{array} \mathit{Rules}(10) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{TaughtBy}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{HasPosition}(D,P_2,\mathit{Po})\wedge \end{array} \mathit{Rules}(11) \right)$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{TaughtBy}(D,C,P_1,\mathit{Te})\wedge \\ \mathit{InPhase}(D,P_2,\mathit{Ph})\wedge \end{array} \mathit{Rules}(12) \right)$$

Section	**Figure**	**Source**	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{T}_l\rvert$	$\lvert\mathcal{L}\rvert$	Avg. $Tup(\phi)$	Boolean	Inconsistent
5.9.1	5.5(a)	UW-CSE	$2,161$	49	113 to 452	5.8 to 8.3	yes	yes
5.9.1	5.5(b)	PRAVDA	$75,091$	$37,383$	$89,874$	2.3	no	yes
5.9.2	5.6(a)	synthetic	100	100	10 to 100	5.8	yes	some
5.9.2	5.6(b)	YAGO2	$224,440,854$	$19,985$	$5,562$	3.6	no	no
5.9.3	5.7 $P1$			$217,846$	$228,050$	2.7	no	yes
5.9.3	5.7 $P2$	YAGO2	$224,440,854$	$217,846$	$79,600$	60.6	no	yes
5.9.3	5.7 $P3$			$1,721,156$	$459,597$	3.7	no	no
5.9.4	5.8(a)	synthetic	100	100	1 to 15	5.8	yes	no
5.9.4	5.8(b)				339	6.0	yes	yes
5.9.4	5.8(c)	UW-CSE	$2,161$	49	113	8.5	yes	no
5.9.4	5.8(d)				339	6.0	yes	yes

Table A.1: Learning Problem Instance Statistics

$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} TaughtBy(D,C,P_1,Te)\wedge & \\ ProjectMember(D,Pr,P_2)\wedge & Rules(13) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} Publication(D,Ti,P_1)\wedge & \\ Ta(D,C,P_2,Te)\wedge & Rules(14) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} Publication(D,Ti,P_1)\wedge & \\ TaughtBy(D,C,P_2,Te)\wedge & Rules(15) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} Publication(D,Ti,P_1)\wedge & \\ Publication(D,Ti,P_2)\wedge & Rules(16) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} Publication(D,Ti,P_1)\wedge & \\ YearsInProgram(D,P_2,Y)\wedge & Rules(17) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} Publication(D,Ti,P_1)\wedge & \\ HasPosition(D,P_2,Po)\wedge & Rules(18) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} Publication(D,Ti,P_1)\wedge & \\ InPhase(D,P_2,Ph)\wedge & Rules(19) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} Publication(D,Ti,P_1)\wedge & \\ ProjectMember(D,Pr,P_2)\wedge & Rules(20) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} YearsInProgram(D,P_1,Y)\wedge & \\ Ta(D,C,P_2,Te)\wedge & Rules(21) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} YearsInProgram(D,P_1,Y)\wedge & \\ TaughtBy(D,C,P_2,Te)\wedge & Rules(22) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} YearsInProgram(D,P_1,Y)\wedge & \\ Publication(D,Ti,P_2)\wedge & Rules(23) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} YearsInProgram(D,P_1,Y)\wedge & \\ YearsInProgram(D,P_2,Y)\wedge & Rules(24) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} YearsInProgram(D,P_1,Y)\wedge & \\ HasPosition(D,P_2,Po)\wedge & Rules(25) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} YearsInProgram(D,P_1,Y)\wedge & \\ InPhase(D,P_2,Ph)\wedge & Rules(26) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} YearsInProgram(D,P_1,Y)\wedge & \\ ProjectMember(D,Pr,P_2)\wedge & Rules(27) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} HasPosition(D,P_1,Po)\wedge & \\ Ta(D,C,P_2,Te)\wedge & Rules(28) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} HasPosition(D,P_1,Po)\wedge & \\ TaughtBy(D,C,P_2,Te)\wedge & Rules(29) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} HasPosition(D,P_1,Po)\wedge & \\ Publication(D,Ti,P_2)\wedge & Rules(30) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} HasPosition(D,P_1,Po)\wedge & \\ YearsInProgram(D,P_2,Y)\wedge & Rules(31) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} HasPosition(D,P_1,Po)\wedge & \\ HasPosition(D,P_2,Po)\wedge & Rules(32) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} HasPosition(D,P_1,Po)\wedge & \\ InPhase(D,P_2,Ph)\wedge & Rules(33) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} HasPosition(D,P_1,Po)\wedge & \\ ProjectMember(D,Pr,P_2)\wedge & Rules(34) \end{pmatrix}$$
$$AdvisedBy(P_1,P_2) \leftarrow \begin{pmatrix} InPhase(D,P_1,Ph)\wedge & \\ Ta(D,C,P_2,Te)\wedge & Rules(35) \end{pmatrix}$$

$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{InPhase}(D,P_1,\mathit{Ph})\wedge \\ \mathit{TaughtBy}(D,C,P_2,\mathit{Te})\wedge \end{array} \mathit{Rules}(36) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{InPhase}(D,P_1,\mathit{Ph})\wedge \\ \mathit{Publication}(D,\mathit{Ti},P_2)\wedge \end{array} \mathit{Rules}(37) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{InPhase}(D,P_1,\mathit{Ph})\wedge \\ \mathit{YearsInProgram}(D,P_2,Y)\wedge \end{array} \mathit{Rules}(38) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{InPhase}(D,P_1,\mathit{Ph})\wedge \\ \mathit{HasPosition}(D,P_2,\mathit{Po})\wedge \end{array} \mathit{Rules}(39) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{InPhase}(D,P_1,\mathit{Ph})\wedge \\ \mathit{InPhase}(D,P_2,\mathit{Ph})\wedge \end{array} \mathit{Rules}(40) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{InPhase}(D,P_1,\mathit{Ph})\wedge \\ \mathit{ProjectMember}(D,\mathit{Pr},P_2)\wedge \end{array} \mathit{Rules}(41) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{ProjectMember}(D,\mathit{Pr},P_1)\wedge \\ \mathit{Ta}(D,C,P_2,\mathit{Te})\wedge \end{array} \mathit{Rules}(42) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{ProjectMember}(D,\mathit{Pr},P_1)\wedge \\ \mathit{TaughtBy}(D,C,P_2,\mathit{Te})\wedge \end{array} \mathit{Rules}(43) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{ProjectMember}(D,\mathit{Pr},P_1)\wedge \\ \mathit{Publication}(D,\mathit{Ti},P_2)\wedge \end{array} \mathit{Rules}(44) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{ProjectMember}(D,\mathit{Pr},P_1)\wedge \\ \mathit{YearsInProgram}(D,P_2,Y)\wedge \end{array} \mathit{Rules}(45) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{ProjectMember}(D,\mathit{Pr},P_1)\wedge \\ \mathit{HasPosition}(D,P_2,\mathit{Po})\wedge \end{array} \mathit{Rules}(46) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{ProjectMember}(D,\mathit{Pr},P_1)\wedge \\ \mathit{InPhase}(D,P_2,\mathit{Ph})\wedge \end{array} \mathit{Rules}(47) \right)$$
$$\mathit{AdvisedBy}(P_1,P_2) \leftarrow \left(\begin{array}{l} \mathit{ProjectMember}(D,\mathit{Pr},P_1)\wedge \\ \mathit{ProjectMember}(D,\mathit{Pr},P_2)\wedge \end{array} \mathit{Rules}(48) \right)$$

Regarding the variables, D is a department name, P_1 and P_2 are persons, C is a course, Po is a position, Te is a term, Ph is a phase, Pr is a project, Ti is a title, and Y is a year. Finally, the positive labels with label probability 1.0 are all instances of *AdvisedBy*. In contrast, negative labels (label probability 0.0) are uniformly drawn from the person-person pairs not present in the *AdvisedBy* relation.

Temporal Information Extraction

The database is contained of $\mathcal{T} := \mathit{Occurrence} \cup \mathit{Begin} \cup \mathit{During} \cup \mathit{End}$, where the tuples to learn are $\mathcal{T}_l := \mathit{Begin} \cup \mathit{During} \cup \mathit{End}$. Moreover, *Occurrence* is a deterministic relation, whereas the other three relations are set to be uncertain. We employ three types of rules as described in Section 5.9.1. First, for reconciling facts we have the following deduction rules

$$\begin{array}{l} \mathit{IsMarriedToBegin}^T(E_1,E_2,T_b,T_e) \\ \quad \leftarrow \mathit{Begin}^T(\mathit{Pid}) \wedge \mathit{Occurrence}^T(\mathit{Pid},E_1,E_1,pp,T_b,T_e) \\ \mathit{IsMarriedToDuring}^T(E_1,E_2,T_b,T_e) \\ \quad \leftarrow \mathit{During}^T(\mathit{Pid}) \wedge \mathit{Occurrence}^T(\mathit{Pid},E_1,E_1,pp,T_b,T_e) \end{array}$$

$$\begin{aligned}
&\mathit{IsMarriedToEnd}^T(E_1, E_2, T_b, T_e)\\
&\quad\leftarrow \mathit{End}^T(\mathit{Pid}) \wedge \mathit{Occurrence}^T(\mathit{Pid}, E_1, E_1, \mathit{pp}, T_b, T_e)\\
&\mathit{WorksForClubBegin}^T(E_1, E_2, T_b, T_e)\\
&\quad\leftarrow \mathit{Begin}^T(\mathit{Pid}) \wedge \mathit{Occurrence}^T(\mathit{Pid}, E_1, E_1, \mathit{pc}, T_b, T_e)\\
&\mathit{WorksForClubDuring}^T(E_1, E_2, T_b, T_e)\\
&\quad\leftarrow \mathit{During}^T(\mathit{Pid}) \wedge \mathit{Occurrence}^T(\mathit{Pid}, E_1, E_1, \mathit{pc}, T_b, T_e)\\
&\mathit{WorksForClubEnd}^T(E_1, E_2, T_b, T_e)\\
&\quad\leftarrow \mathit{End}^T(\mathit{Pid}) \wedge \mathit{Occurrence}^T(\mathit{Pid}, E_1, E_1, \mathit{pc}, T_b, T_e)
\end{aligned}$$

where E_1 and E_2 are entities, T_b and T_e encode the time-interval, *Pid* is the pattern id, and *pp* and *pc* are constants standing for the type pairs person-person and person-club, respectively. The next deduction rules enforce mutual exclusion among the *Begin*, *During*, and *End* entry of each pattern id

$$\begin{array}{rcl}
\mathit{Constraint1}(\mathit{Pid}) & \leftarrow & \mathit{Begin}(\mathit{Pid}) \wedge \mathit{During}(\mathit{Pid})\\
\mathit{Constraint2}(\mathit{Pid}) & \leftarrow & \mathit{Begin}(\mathit{Pid}) \wedge \mathit{End}(\mathit{Pid})\\
\mathit{Constraint3}(\mathit{Pid}) & \leftarrow & \mathit{During}(\mathit{Pid}) \wedge \mathit{End}(\mathit{Pid})
\end{array}$$

which we achieve by labeling their resulting lineage with probability 0.0. Finally, we encode temporal precedence constraints by the deduction rules

$$\begin{aligned}
\mathit{Constraint4}(E_1, E_2) &\leftarrow \begin{pmatrix} \mathit{IsMarriedToBegin}^T(E_1, E_2, T_b, T_e) \wedge \\ \mathit{IsMarriedToDuring}^T(E_1, E_2, T'_b, T'_e) \wedge \\ T'_b < T_e \end{pmatrix}\\
\mathit{Constraint5}(E_1, E_2) &\leftarrow \begin{pmatrix} \mathit{IsMarriedToBegin}^T(E_1, E_2, T_b, T_e) \wedge \\ \mathit{IsMarriedToEnd}^T(E_1, E_2, T'_b, T'_e) \wedge \\ T'_b < T_e \end{pmatrix}\\
\mathit{Constraint6}(E_1, E_2) &\leftarrow \begin{pmatrix} \mathit{IsMarriedToDuring}^T(E_1, E_2, T_b, T_e) \wedge \\ \mathit{IsMarriedToEnd}^T(E_1, E_2, T'_b, T'_e) \wedge \\ T'_b < T_e \end{pmatrix}\\
\mathit{Constraint7}(E_1, E_2) &\leftarrow \begin{pmatrix} \mathit{WorksForClubBegin}^T(E_1, E_2, T_b, T_e) \wedge \\ \mathit{WorksForClubDuring}^T(E_1, E_2, T'_b, T'_e) \wedge \\ T'_b < T_e \end{pmatrix}\\
\mathit{Constraint8}(E_1, E_2) &\leftarrow \begin{pmatrix} \mathit{WorksForClubBegin}^T(E_1, E_2, T_b, T_e) \wedge \\ \mathit{WorksForClubEnd}^T(E_1, E_2, T'_b, T'_e) \wedge \\ T'_b < T_e \end{pmatrix}\\
\mathit{Constraint9}(E_1, E_2) &\leftarrow \begin{pmatrix} \mathit{WorksForClubDuring}^T(E_1, E_2, T_b, T_e) \wedge \\ \mathit{WorksForClubEnd}^T(E_1, E_2, T'_b, T'_e) \wedge \\ T'_b < T_e \end{pmatrix}
\end{aligned}$$

whose resulting lineage we label by probability 0.0 as well. Additionally, we employ the 266 labels for textual patterns and the 341 labels for facts from the original work.

Appendix B

Table of Symbols

Symbol	Description	Introduced in
a	constant	Section 2.1.1
$\bar{a}$	vector of constants	Section 2.1.1
$\mathcal{U}$	universe of constants	Section 2.1.1
$\mathcal{U}^T$	time universe, sequence of time-points	Section 3.4.1
X	variable	Section 2.1.1
$\bar{X}$	vector of variables and constants	Section 2.1.1
$Var(\bar{X})$	set of variables in $\bar{X}$	Section 2.1.1
t_b	constant, time-point, start of an interval	Section 3.4.1
t_e	constant, time-point, end of an interval	Section 3.4.1
t_{min}	constant, time-point, begin of the time-universe	Section 3.4.1
t_{max}	constant, time-point, end of the time-universe	Section 3.4.1
T_b	temporal variable, begin of an interval	Section 3.4.1
T_e	temporal variable, end of an interval	Section 3.4.1
$=^T$	temporal equality predicate, compares time-points	Section 3.4.4
$<^T$	temporal precedence predicate, compares time-points	Section 3.4.4
R	relation	Section 2.1.1
R^T	temporal relation	Section 3.4.2
$R(\bar{a})$	ground literal, tuple	Section 2.1.1
$R(\bar{X})$	first-order literal	Section 2.1.1
$Q(\bar{a})$	query answer	Section 2.1.4
$Q(\bar{X})$	query	Section 2.1.4
σ	substitution of variables to constants or variables	Section 2.1.3
$\mathcal{R}$	set of tuples, relation instance	Section 2.1.1
$\mathcal{T}$	set of tuples, usually of all the database	Section 2.1.1
$\mathcal{W}$	set of tuples, possible world	Section 2.2.1
$\mathcal{T}_l$	set of tuples with probabilities to be learned	Section 5.5
I_n	tuple identifier	Section 2.1.1

Symbol	Description	Introduced in
ϕ, ψ	propositional lineage formula	Section 2.2.4
Φ, Ψ	first-order lineage formula	Section 4.4
ϕ_{low}	propositional formula for lower bound of Φ	Section 4.4.3
ϕ_{up}	propositional formula for upper bound of Φ	Section 4.4.3
ϕ_c	propositional formula encoding constraints	Section 2.2.6
λ	function returning the lineage of its argument	Section 2.2.4
p	probability of a tuple	Section 2.2.3
$P(\phi)$	probability of ϕ	Section 2.2.5
$Tup(\phi)$	tuple identifiers occurring in ϕ	Section 2.2.5
$\mathcal{M}$	models, satisfying assignments	Section 2.2.5
$\mathcal{D}$	set of deduction rules	Section 2.1.2
$\mathcal{D}_c$	set of deduction rules which ground constraints	Section 2.2.6
$\mathcal{C}_p$	set of literals for grounding positive constraints	Section 2.2.6
$\mathcal{C}_n$	set of literals for grounding negative constraints	Section 2.2.6
$\mathcal{L}$	set of labels during learning	Section 5.5

Bibliography

[1] S. Abiteboul, L. Herr, and J. V. den Bussche. Temporal Connectives versus Explicit Timestamps in Temporal Query Languages. In J. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases, Proceedings of the International Workshop on Temporal Databases, Zürich, Switzerland, 17-18 September 1995*, Workshops in Computing, pages 43–57, Berlin, Heidelberg, New York, 1995. Springer.

[2] S. Abiteboul, R. Hull, and V. Vianu, editors. *Foundations of Databases.* Addison-Wesley, Boston, MA, USA, 1st edition, 1995.

[3] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *SIGMOD Record*, 16(3):34–48, Dec. 1987.

[4] S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic XML models. *VLDB Journal*, 18(5):1041–1064, 2009.

[5] M. O. Akinde, O. G. Jensen, and M. H. Böhlen. Minimizing Detail Data in Data Warehouses. In H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso, editors, *Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia, Spain, March 23-27, 1998, Proceedings*, volume 1377 of *Lecture Notes in Computer Science*, pages 293–307, Berlin, Heidelberg, New York, 1998. Springer.

[6] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, Nov. 1983.

[7] P. Alvaro, W. R. Marczak, N. Conway, J. M. Hellerstein, D. Maier, and R. Sears. Dedalus: datalog in time and space. In *Proceedings of the First international conference on Datalog Reloaded*, Datalog, pages 262–281, Berlin, Heidelberg, New York, 2011. Springer.

[8] A. Amarilli and P. Senellart. On the Connections Between Relational and XML Probabilistic Data Models. In *Proceedings of the 29th British*

National Conference on Big Data, BNCOD, pages 121–134, Berlin, Heidelberg, New York, 2013. Springer-Verlag.

[9] L. Anselma, P. Terenziani, and R. T. Snodgrass. Valid-Time Indeterminacy in Temporal Relational Databases: Semantics and Representations. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2880–2894, Dec. 2013.

[10] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and Simple Relational Processing of Uncertain Data. In *Proceedings of the 24th International Conference on Data Engineering*, ICDE, pages 983–992, Washington, DC, USA, 2008. IEEE Computer Society.

[11] K. Apt. *Principles of Constraint Programming.* Cambridge University Press, Cambridge, UK, 2003.

[12] P. Beame, J. Li, S. Roy, and D. Suciu. Model Counting of Query Expressions: Limitations of Propositional Methods. In *Proceedings of the 17th International Conference on Database Theory*, ICDT, New York, NY, USA, 2014. ACM. To appear.

[13] R. Bekkerman, M. Bilenko, and J. Langford, editors. *Scaling up machine learning: Parallel and distributed approaches.* Cambridge University Press, Cambridge, UK, 1st edition, 2011.

[14] O. Benjelloun, A. Das Sarma, A. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB Journal*, 17(2):243–264, Mar. 2008.

[15] M. H. Böhlen, R. T. Snodgrass, and M. D. Soo. Coalescing in Temporal Databases. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB, pages 180–191, San Rafael, California, USA, 1996. Morgan Kaufmann Publishers.

[16] B. Bollig and I. Wegener. Improving the Variable Ordering of OBDDs Is NP-Complete. *IEEE Transactions on Computers*, 45(9):993–1002, Sept. 1996.

[17] L. Bottou and O. Bousquet. The Tradeoffs of Large Scale Learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*, pages 161–168. Curran Associates, 2008.

[18] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *Proceedings of the International Conference on Management of data*, SIGMOD, pages 891–893, New York, NY, USA, 2005. ACM.

[19] S. Brin. Extracting Patterns and Relations from the World Wide Web. In *Selected Papers from the International Workshop on The World Wide Web and Databases*, WebDB, pages 172–183, Berlin, Heidelberg, New York, 1999. Springer.

[20] P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods*. Springer, Berlin, Heidelberg, New York, 2nd edition, 1991.

[21] S. Ceri, G. Gottlob, and L. Tanca. What You Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, Mar. 1989.

[22] J. Chen and L. Feng. Efficient pruning algorithm for top-K ranking on dataset with value uncertainty. In *Proceedings of the 22nd ACM International Conference on Conference on Information and Knowledge Management*, CIKM, pages 2231–2236, New York, NY, USA, 2013. ACM.

[23] J. Chomicki. Polynomial Time Query Processing in Temporal Deductive Databases. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS, pages 379–391, New York, NY, USA, 1990. ACM.

[24] J. Chomicki and T. Imieliński. Temporal Deductive Databases and Infinite Objects. In *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS, pages 61–73, New York, NY, USA, 1988. ACM.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 3rd edition, 2009.

[26] Y. Cui, J. Widom, and J. L. Wiener. Tracing the Lineage of View Data in a Warehousing Environment. *ACM Transactions on Database Systems*, 25(2):179–227, June 2000.

[27] N. Dalvi, C. Ré, and D. Suciu. Probabilistic Databases: Diamonds in the Dirt. *Communications of the ACM*, 52(7):86–94, July 2009.

[28] N. Dalvi, K. Schnaitter, and D. Suciu. Computing Query Probability with Incidence Algebras. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS, pages 203–214, New York, NY, USA, 2010. ACM.

[29] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB Journal*, 16(4):523–544, Oct. 2007.

[30] N. Dalvi and D. Suciu. Management of Probabilistic Data: Foundations and Challenges. In *Proceedings of the Twenty-sixth ACM*

SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS, pages 1–12, New York, NY, USA, 2007. ACM.

[31] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS, pages 293–302, New York, NY, USA, 2007. ACM.

[32] N. Dalvi and D. Suciu. The Dichotomy of Probabilistic Inference for Unions of Conjunctive Queries. *Journal of the ACM*, 59(6):30:1–30:87, Jan. 2013.

[33] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI, pages 2468–2473, San Rafael, California, USA, 2007. Morgan Kaufmann Publishers.

[34] R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, May 1991.

[35] A. Dekhtyar, R. Ross, and V. S. Subrahmanian. Probabilistic temporal databases, I: algebra. *ACM Transactions on Database Systems*, 26(1):41–95, Mar. 2001.

[36] A. Dickenstein and I. Z. Emiris. *Solving Polynomial Equations: Foundations, Algorithms, and Applications.* Springer, Berlin, Heidelberg, New York, 1st edition, 2010.

[37] A. Dignös, M. H. Böhlen, and J. Gamper. Temporal alignment. In *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 433–444, New York, NY, USA, 2012. ACM.

[38] M. Dylla, I. Miliaraki, and M. Theobald. Top-k Query Processing in Probabilistic Databases with Non-Materialized Views. Research Report MPI-I-2012-5-002, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, June 2012.

[39] M. Dylla, I. Miliaraki, and M. Theobald. A Temporal-Probabilistic Database Model for Information Extraction. *Proceedings of the VLDB Endowment*, 6(14):1810–1821, 2013.

[40] M. Dylla, M. Sozio, and M. Theobald. Resolving Temporal Conflicts in Inconsistent RDF Knowledge Bases. In T. Härder, W. Lehner, B. Mitschang, H. Schöning, and H. Schwarz, editors, *Datenbanksysteme für Business, Technologie und Web (BTW)*, GI Lecture Notes in Informatics (LNI), pages 474–493. Gesellschaft für Informatik, 2011.

[41] M. Dylla, M. Theobald, and I. Miliaraki. Top-k query processing in probabilistic databases with non-materialized views. In *Proceedings of the 29th International Conference on Data Engineering*, ICDE, pages 122–133, Washington, DC, USA, 2013. IEEE Computer Society.

[42] C. E. Dyreson and R. T. Snodgrass. Supporting Valid-time Indeterminacy. *ACM Transactions on Database Systems*, 23(1):1–57, Mar. 1998.

[43] T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle. Querying Uncertain Spatio-Temporal Data. In A. Kementsietsidis and M. A. V. Salles, editors, *Proceedings of the 28th International Conference on Data Engineering*, ICDE, pages 354–365, Washington, DC, USA, 2012. IEEE Computer Society.

[44] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS, pages 102–113, New York, NY, USA, 2001. ACM.

[45] W. Feller. *An introduction to probability theory and its applications*. Wiley, Hoboken, NJ, USA, 3rd edition, 1968.

[46] R. Fink, L. Han, and D. Olteanu. Aggregation in Probabilistic Databases via Knowledge Compilation. *Proceedings of the VLDB Endowment*, 5(5):490–501, Jan. 2012.

[47] R. Fink, J. Huang, and D. Olteanu. Anytime approximation in probabilistic databases. *VLDB Journal*, 22(6):823–848, 2013.

[48] R. Fink and D. Olteanu. On the optimal approximation of queries using tractable propositional languages. In *Proceedings of the 14th International Conference on Database Theory*, ICDT, pages 174–185, New York, NY, USA, 2011. ACM.

[49] R. Fink, D. Olteanu, and S. Rath. Providing Support for Full Relational Algebra in Probabilistic Databases. In *Proceedings of the 27th International Conference on Data Engineering*, ICDE, pages 315–326, Washington, DC, USA, 2011. IEEE Computer Society.

[50] J. R. Finkel, T. Grenager, and C. Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL, pages 363–370, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[51] M. Fisher, D. Gabbay, and L. Vila. *Handbook of Temporal Reasoning in Artificial Intelligence.* Foundations of Artificial Intelligence. Elsevier, Essex, UK, 1st edition, 2005.

[52] N. Fuhr and T. Rölleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Transactions on Information Systems*, 15(1):32–66, Jan. 1997.

[53] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley, Boston, MA, USA, 1994.

[54] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, NY, USA, 1990.

[55] T. Ge, S. B. Zdonik, and S. Madden. Top-k queries on uncertain data: on score distribution and typical answers. In U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul, editors, *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 375–388, New York, NY, USA, 2009. ACM.

[56] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning.* MIT Press, Cambridge, MA, USA, 1st edition, 2007.

[57] G. Gottlob and C. Papadimitriou. On the Complexity of Single-rule Datalog Queries. *Information and Computation*, 183(1):104–122, May 2003.

[58] E. Grädel, Y. Gurevich, and C. Hirsch. The Complexity of Query Reliability. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS, pages 227–234, New York, NY, USA, 1998. ACM.

[59] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance Semirings. In *Proceedings of the Twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS, pages 31–40, New York, NY, USA, 2007. ACM.

[60] T. J. Green and V. Tannen. Models for Incomplete and Probabilistic Information. *IEEE Data Engineering Bulletin*, 29(1):17–24, 2006.

[61] M. R. Gupta and Y. Chen. Theory and Use of the EM Algorithm. *Foundations and Trends in Signal Processing*, 4(3):223–296, Mar. 2011.

[62] B. Gutmann, A. Kimmig, K. Kersting, and L. Raedt. Parameter Learning in Probabilistic Databases: A Least Squares Approach. In

Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I, PKDD, pages 473–488, Berlin, Heidelberg, New York, 2008. Springer.

[63] B. Gutmann, I. Thon, and L. De Raedt. Learning the Parameters of Probabilistic Logic Programs from Interpretations. In *Proceedings of the 2011 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I*, PKDD, pages 581–596, Berlin, Heidelberg, New York, 2011. Springer.

[64] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, Berlin, Heidelberg, New York, 2nd edition, 2009.

[65] J. W. Hector Garcia-Molina, Jeffrey D. Ullman. *Database systems: the complete book*. Prentice-Hall, Upper Saddle River, NJ, USA, 1st edition, 2002.

[66] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. *Artificial Intelligence*, 194:28–61, Jan. 2013.

[67] M. Hua, J. Pei, and X. Lin. Ranking queries on uncertain data. *VLDB Journal*, 20(1):129–153, 2011.

[68] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach. In *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 673–686, New York, NY, USA, 2008. ACM.

[69] I. F. Ilyas, G. Beskales, and M. A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Computing Surveys*, 40(4):11:1–11:58, Oct. 2008.

[70] I. F. Ilyas and M. A. Soliman. *Probabilistic Ranking Techniques in Relational Databases*. Synthesis Lectures on Data Management. Morgan & Claypool, San Rafael, California, USA, 2011.

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

[72] J. Jestes, G. Cormode, F. Li, and K. Yi. Semantics of Ranking Queries for Probabilistic Data. *IEEE Transactions on Knowledge and Data Engineering*, 23(12):1903–1917, 2011.

[73] A. Jha and D. Suciu. Probabilistic databases with MarkoViews. *Proceedings of the VLDB Endowment*, 5(11):1160–1171, July 2012.

[74] A. Jha and D. Suciu. Knowledge Compilation Meets Database Theory: Compiling Queries to Decision Diagrams. *Theory of Computing Systems*, 52(3):403–440, Apr. 2013.

[75] F. Kabanza, J.-M. Stevenne, and P. Wolper. Handling Infinite Temporal Data. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS, pages 392–403, New York, NY, USA, 1990. ACM.

[76] B. Kanagal and A. Deshpande. Lineage processing over correlated probabilistic databases. In *Proceedings of the International Conference on Management of data*, SIGMOD, pages 675–686, New York, NY, USA, 2010. ACM.

[77] B. Kanagal, J. Li, and A. Deshpande. Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In *Proceedings of the International Conference on Management of data*, SIGMOD, pages 841–852, New York, NY, USA, 2011. ACM.

[78] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS, pages 299–313, New York, NY, USA, 1990. ACM.

[79] O. Kennedy and C. Koch. PIP: A database system for great and small expectations. In *Proceedings of the 26th International Conference on Data Engineering*, ICDE, pages 157–168, Washington, DC, USA, 2010. IEEE Computer Society.

[80] M. Keulen and A. Keijzer. Qualitative Effects of Knowledge Rules and User Feedback in Probabilistic Data Integration. *VLDB Journal*, 18(5):1191–1217, Oct. 2009.

[81] S. Khanna, S. Roy, and V. Tannen. Queries with Difference on Probabilistic Databases. *Proceedings of the VLDB Endowment*, 4(11):1051–1062, 2011.

[82] B. Kimelfeld and P. Senellart. Probabilistic XML: Models and Complexity. In Z. Ma and L. Yan, editors, *Advances in Probabilistic Databases for Uncertain Information Management*, volume 304 of *Studies in Fuzziness and Soft Computing*, pages 39–66. Springer, Berlin, Heidelberg, New York, 2013.

[83] C. Koch and D. Olteanu. Conditioning probabilistic databases. *Proceedings of the VLDB Endowment*, 1(1):313–325, Aug. 2008.

[84] K. Kulkarni and J.-E. Michels. Temporal Features in SQL:2011. *SIGMOD Record*, 41(3):34–43, Oct. 2012.

[85] S. K. Lahiri and M. Musuvathi. An Efficient Decision Procedure for UTVPI Constraints. In *Proceedings of the 5th International Conference on Frontiers of Combining Systems*, FroCoS, pages 168–183, Berlin, Heidelberg, New York, 2005. Springer.

[86] J. Li and A. Deshpande. Ranking Continuous Probabilistic Datasets. *Proceedings of the VLDB Endowment*, 3(1):638–649, 2010.

[87] J. Li, C. Liu, R. Zhou, and W. Wang. Top-k Keyword Search over Probabilistic XML Data. In *Proceedings of the 27th International Conference on Data Engineering*, ICDE, pages 673–684, Washington, DC, USA, 2011. IEEE Computer Society.

[88] J. Li, B. Saha, and A. Deshpande. A Unified Approach to Ranking in Probabilistic Databases. *VLDB Journal*, 20(2):249–275, Apr. 2011.

[89] X. Lian and L. Chen. Probabilistic Inverse Ranking Queries in Uncertain Databases. *VLDB Journal*, 20(1):107–127, Feb. 2011.

[90] J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, Heidelberg, New York, 2nd edition, 1987.

[91] D. Lowd and P. Domingos. Efficient Weight Learning for Markov Logic Networks. In *Knowledge Discovery in Databases: PKDD 2007, 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD, pages 200–211, Berlin, Heidelberg, New York, 2007. Springer.

[92] C. Lutz, F. Wolter, and M. Zakharyashev. Temporal Description Logics: A Survey. In *Temporal Representation and Reasoning, 2008. TIME '08. 15th International Symposium on*, pages 3–14, Washington, DC, USA, 2008. IEEE Computer Society.

[93] M. Magnani and D. Montesi. A Survey on Uncertainty Management in Data Integration. *Journal of Data and Information Quality*, 2(1):5:1–5:33, July 2010.

[94] M. Mampaey, J. Vreeken, and N. Tatti. Summarizing Data Succinctly with the Most Informative Itemsets. *ACM Transactions on Knowledge Discovery from Data*, 6(4):16:1–16:42, Dec. 2012.

[95] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[96] V. Markl, P. J. Haas, M. Kutsch, N. Megiddo, U. Srivastava, and T. M. Tran. Consistent selectivity estimation via maximum entropy. *VLDB Journal*, 16(1):55–76, 2007.

[97] C. Mayfield, J. Neville, and S. Prabhakar. ERACER: A Database Approach for Statistical Inference and Data Cleaning. In *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 75–86, New York, NY, USA, 2010. ACM.

[98] T. Meiser, M. Dylla, and M. Theobald. Interactive reasoning in uncertain RDF knowledge bases. In C. Macdonald, I. Ounis, and I. Ruthven, editors, *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM, pages 2557–2560, New York, NY, USA, 2011. ACM.

[99] K. P. Murphy. *Machine Learning: A Probabilistic Perspective.* MIT Press, Cambridge, MA, USA, 2012.

[100] R. Murthy, R. Ikeda, and J. Widom. Making Aggregation Work in Uncertain and Probabilistic Databases. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1261–1273, 2011.

[101] N. Nakashole, G. Weikum, and F. Suchanek. PATTY: A Taxonomy of Relational Patterns with Semantic Types. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL, pages 1135–1145, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[102] F. Niu, C. Ré, A. Doan, and J. Shavlik. Tuffy: Scaling Up Statistical Inference in Markov Logic Networks Using an RDBMS. *Proceedings of the VLDB Endowment*, 4(6):373–384, Mar. 2011.

[103] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, Berlin, Heidelberg, New York, 2nd edition, 2006.

[104] P. Ohrstrom. *Temporal Logic: From Ancient Ideas to Artificial Intelligence.* Springer, Berlin, Heidelberg, New York, 2009.

[105] D. Olteanu and J. Huang. Using OBDDs for Efficient Query Evaluation on Probabilistic Databases. In S. Greco and T. Lukasiewicz, editors, *Scalable Uncertainty Management, Second International Conference, SUM 2008*, volume 5291 of *Lecture Notes in Computer Science*, pages 326–340, Berlin, Heidelberg, New York, 2008. Springer.

[106] D. Olteanu, J. Huang, and C. Koch. SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases. In *Proceedings of the 25th International Conference on Data Engineering*, ICDE, pages 640–651, Washington, DC, USA, 2009. IEEE Computer Society.

[107] D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *Proceedings of the 26th International Conference on Data Engineering*, ICDE, pages 145–156, Washington, DC, USA, 2010. IEEE Computer Society.

[108] D. Olteanu and H. Wen. Ranking Query Answers in Probabilistic Databases: Complexity and Efficient Algorithms. In *Proceedings of the 28th International Conference on Data Engineering*, ICDE, pages 282–293, Washington, DC, USA, 2012. IEEE Computer Society.

[109] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Upper Saddle River, NJ, USA, 1st edition, 1982.

[110] L. Peng, Y. Diao, and A. Liu. Optimizing Probabilistic Query Processing on Continuous Uncertain Data. *Proceedings of the VLDB Endowment*, 4(11):1169–1180, 2011.

[111] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer, Berlin, Heidelberg, New York, 3rd edition, 2008.

[112] D. Poole. First-order Probabilistic Inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI, pages 985–991, San Rafael, California, USA, 2003. Morgan Kaufmann Publishers.

[113] H. Poon and P. Domingos. Sum-Product Networks: A New Deep Architecture. In F. G. Cozman and A. Pfeffer, editors, *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI, pages 337–346. AUAI Press, 2011.

[114] Y. Qi, R. Jain, S. Singh, and S. Prabhakar. Threshold Query Optimization for Uncertain Data. In *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 315–326, New York, NY, USA, 2010. ACM.

[115] C. Ré, N. N. Dalvi, and D. Suciu. Efficient Top-k Query Evaluation on Probabilistic Data. In *Proceedings of the 23rd International Conference on Data Engineering*, ICDE, pages 886–895, Washington, DC, USA, 2007. IEEE Computer Society.

[116] C. Ré and D. Suciu. Approximate Lineage for Probabilistic Databases. *Proceedings of the VLDB Endowment*, 1(1):797–808, Aug. 2008.

[117] C. Ré and D. Suciu. The trichotomy of HAVING queries on a probabilistic database. *VLDB Journal*, 18(5):1091–1116, 2009.

[118] C. Ré and D. Suciu. Understanding Cardinality Estimation Using Entropy Maximization. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS, pages 53–64, New York, NY, USA, 2010. ACM.

[119] T. Rekatsinas, A. Deshpande, and L. Getoor. Local Structure and Determinism in Probabilistic Databases. In *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 373–384, New York, NY, USA, 2012. ACM.

[120] M. Richardson and P. Domingos. Markov Logic Networks. *Machine Learning*, 62(1-2):107–136, Feb. 2006.

[121] S. Riedel. Improving the Accuracy and Efficiency of MAP Inference for Markov Logic. In D. A. McAllester and P. Myllymäki, editors, *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, UAI, pages 468–475. AUAI Press, 2008.

[122] T. Rölleke and N. Fuhr. Probabilistic Reasoning for Large Scale Databases. In *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW)*, Informatik Aktuell, pages 118–132, Berlin, Heidelberg, New York, 1997. Springer.

[123] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, Upper Saddle River, NJ, USA, 2nd edition, 2003.

[124] Y. Sagiv and M. Yannakakis. Equivalences Among Relational Expressions with the Union and Difference Operators. *Journal of the ACM*, 27(4):633–655, Oct. 1980.

[125] A. D. Sarma, M. Theobald, and J. Widom. Exploiting Lineage for Confidence Computation in Uncertain and Probabilistic Databases. In *Proceedings of the 24th International Conference on Data Engineering*, ICDE, pages 1023–1032, Washington, DC, USA, 2008. IEEE Computer Society.

[126] A. D. Sarma, M. Theobald, and J. Widom. LIVE: a lineage-supported versioned DBMS. In *Proceedings of the 22nd international conference on Scientific and statistical database management*, volume 6187 of *Lecture Notes in Computer Science*, pages 416–433, Berlin, Heidelberg, New York, 2010. Springer.

[127] S. Sathe, H. Jeung, and K. Aberer. Creating Probabilistic Databases from Imprecise Time-series Data. In *Proceedings of the 27th International Conference on Data Engineering*, ICDE, pages 327–338, Washington, DC, USA, 2011. IEEE Computer Society.

[128] P. Sen, A. Deshpande, and L. Getoor. PrDB: managing and exploiting rich correlations in probabilistic databases. *VLDB Journal*, 18(5):1065–1090, Oct. 2009.

[129] P. Sen, A. Deshpande, and L. Getoor. Read-once Functions and Query Evaluation in Probabilistic Databases. *Proceedings of the VLDB Endowment*, 3(1-2):1068–1079, Sept. 2010.

[130] A. N. Shiryaev. *Probability.* Springer, Berlin, Heidelberg, New York, 2nd edition, 1995.

[131] P. Singla and P. Domingos. Discriminative Training of Markov Logic Networks. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, AAAI'05, pages 868–873. AAAI Press, 2005.

[132] P. Singla and P. Domingos. Lifted First-order Belief Propagation. In D. Fox and C. P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, AAAI, pages 1094–1099. AAAI Press, 2008.

[133] R. M. Smullyan. *First-order logic.* Springer, Berlin, Heidelberg, New York, 1st edition, 1968.

[134] R. T. Snodgrass, editor. *The TSQL2 Temporal Query Language.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1995.

[135] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k Query Processing in Uncertain Databases. In *Proceedings of the 23rd International Conference on Data Engineering*, ICDE, pages 896–905, Washington, DC, USA, 2007. IEEE Computer Society.

[136] M. A. Soliman, I. F. Ilyas, D. Martinenghi, and M. Tagliasacchi. Ranking with Uncertain Scoring Functions: Semantics and Sensitivity Measures. In *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 805–816, New York, NY, USA, 2011. ACM.

[137] J. Stoyanovich, S. Davidson, T. Milo, and V. Tannen. Deriving Probabilistic Databases with Inference Ensembles. In *Proceedings of the 27th International Conference on Data Engineering*, ICDE, pages 303–314, Washington, DC, USA, 2011. IEEE Computer Society.

[138] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW, pages 697–706, New York, NY, USA, 2007. ACM.

[139] D. Suciu, D. Olteanu, R. Christopher, and C. Koch. *Probabilistic Databases.* Morgan & Claypool, San Rafael, California, USA, 1st edition, 2011.

[140] P. P. Talukdar, D. Wijaya, and T. Mitchell. Coupled Temporal Scoping of Relational Facts. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM, pages 73–82, New York, NY, USA, 2012. ACM.

[141] B. Taskar, P. Abbeel, and D. Koller. Discriminative Probabilistic Models for Relational Data. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, UAI, pages 485–492, San Rafael, California, USA, 2002. Morgan Kaufmann Publishers.

[142] P. Terenziani. Coping with Events in Temporal Relational Databases. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1181–1185, 2013.

[143] M. Theobald, L. D. Raedt, M. Dylla, A. Kimmig, and I. Miliaraki. 10 Years of Probabilistic Querying - What Next? In B. Catania, G. Guerrini, and J. Pokorný, editors, *Advances in Databases and Information Systems - 17th East European Conference, ADBIS 2013, Genoa, Italy, September 1-4, 2013. Proceedings*, volume 8133 of *Lecture Notes in Computer Science*, pages 1–13. Springer, Berlin, Heidelberg, New York, 2013.

[144] D. Toman and J. Chomicki. Datalog with Integer Periodicity Constraints. *Journal of Logic Programming*, 35(3):263–290, 1998.

[145] A. Tuzhilin and J. Clifford. A Temporal Relational Algebra As Basis for Temporal Relational Completeness. In *Proceedings of the 16th International Conference on Very Large Data Bases*, VLDB, pages 13–23, San Rafael, California, USA, 1990. Morgan Kaufmann Publishers.

[146] G. A. V. Sperschneider. *Logic: A Foundation for Computer Science*. Addison-Wesley, Boston, MA, USA, 1st edition, 1991.

[147] L. G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[148] J. van Benthem. *The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2nd edition, 1991.

[149] G. Van Den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt. Lifted Probabilistic Inference by First-order Knowledge Compilation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, IJCAI, pages 2178–2185. AAAI Press, 2011.

[150] M. van Keulen, A. de Keijzer, and W. Alink. A Probabilistic XML Approach to Data Integration. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE, pages 459–470, Washington, DC, USA, 2005. IEEE Computer Society.

[151] C. Wang, L.-Y. Yuan, J.-H. You, O. R. Zaïane, and J. Pei. On Pruning for Top-K Ranking in Uncertain Databases. *Proceedings of the VLDB Endowment*, 4(10):598–609, 2011.

[152] Y. Wang, M. Dylla, Z. Ren, M. Spaniol, and G. Weikum. PRAVDA-live: Interactive Knowledge Harvesting. In X. wen Chen, G. Lebanon, H. Wang, and M. J. Zaki, editors, *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM, pages 2674–2676, New York, NY, USA, 2012. ACM.

[153] Y. Wang, M. Dylla, M. Spaniol, and G. Weikum. Coupling Label Propagation and Constraints for Temporal Fact Extraction. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ACL, pages 233–237, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[154] Y. Wang, M. Yahya, and M. Theobald. Time-aware Reasoning in Uncertain Knowledge Bases. In A. de Keijzer and M. van Keulen, editors, *Proceedings of the Fourth International VLDB workshop on Management of Uncertain Data (MUD)*, CTIT Workshop Proceedings Series, pages 51–65. Centre for Telematics and Information Technology (CTIT), University of Twente, The Netherlands, 2010.

[155] G. Weikum and M. Theobald. From Information to Knowledge: Harvesting Entities and Relationships from Web Sources. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS, pages 65–76, New York, NY, USA, 2010. ACM.

[156] M. Weisfeld. *The Object-Oriented Thought Process.* Addison-Wesley, Boston, MA, USA, 3rd edition, 2008.