On Probabilistic Fixpoint and Markov Chain Query Languages

Daniel Deutch
Tel Aviv University
danielde@post.tau.ac.il

Christoph Koch
Cornell University
koch@cs.cornell.edu

Tova Milo
Tel Aviv University
milo@cs.tau.ac.il

ABSTRACT

We study highly expressive query languages such as datalog, fixpoint, and while-languages on probabilistic databases. We generalize these languages such that computation steps (e.g. datalog rules) can fire probabilistically. We define two possible semantics for such query languages, namely inflationary semantics where the results of each computation step are added to the current database and non-inflationary queries that induce a random walk in-between database instances. We then study the complexity of exact and approximate query evaluation under these semantics.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: [Languages]
; H.2.1 [**Database Management**]: [Logical Design]
; G.3 [**Mathematics of Computing**]: [Probability and Statistics]

General Terms

Algorithms, Languages, Theory

1. INTRODUCTION

Probabilistic databases have recently started to attract considerable interest. A number of query languages that are analogs of relational algebra or SQL have been studied for probabilistic databases [6, 23, 8, 3, 4, 15, 16, 17, 12], but so far there has been virtually no work on more expressive languages such as fixpoint and while-languages beyond Fuhr's original proposal to use datalog on probabilistic databases in the context of information retrieval [11].

Highly expressive query languages with an iteration construct enable interesting new applications of probabilistic databases and query language research. Iterating languages with probabilistic changes to the database (state) can be used to declaratively specify (queries over) Markov Chains, random walks and stochastic processes [10]. This opens up entirely new application areas for data management research.

There are many applications of Markov Chains (MCs) and Markov Chain Monte Carlo (MCMC) [19, 14] in areas such as bioinformatics and statistical physics, (cf. e.g. [22, 5]), and their theory plays an increasingly important role in theoretical computer science and algorithmics. So far, few systems based on Markov Chains have been developed that are reusable, and none at all that are application domain-independent and that allow to program MCMC using easy to use declarative languages. Declarative datalog-like languages for defining Markov Chains (and walks over them) and systems that efficiently execute queries in such languages, would allow to improve research productivity in areas that use Markov Chains to solve computational tasks, just like relational database management systems have greatly improved productivity in the context of business software and information systems. Declarative query languages for Markov Chains would also allow to program MCMC applications on a higher level of abstraction, which may yield interesting insights into commonalities among such applications, their componentization, and combination into more sophisticated applications.

The main goals of this paper are to establish useful probabilistic query languages based on highly expressive languages in the database literature, and to study their expressiveness and complexity. Specifically, we study probabilistic extensions for fixpoint / while query languages, and a probabilistic extension for datalog, again with inflationary and non-inflationary semantics. We show how classical examples such as random walks over graphs, probabilistic reachability analysis and Bayesian Inference may be expressed in our query languages.

The core of our complexity results is summarized in Table 1. The languages in this table are ordered by increasing expressive power, that is, each language is expressively subsumed by the languages below it in Table 1. As we show, exact evaluation and approximation with quality guarantees relative to the sizes of the probability values to be computed ("relative approximation") are infeasible, but absolute approximation (i.e. approximation up to a constant factor) is efficiently feasible for inflationary queries. The depicted NP-hardness results relate to the corresponding decision problems, namely "does a given value p approximate the correct probability value up to a given ϵ", where "approximate" relates to either relative or absolute approximation, according to the corresponding problem.

We show that our hardness results apply even for very

Language	exact computation	relative approximation	absolute approximation
(linear) datalog without probabilistic rules	$\sharp P$-hard, in PSPACE	NP-hard (Thm. 4.1)	in PTIME
inflationary fixpoint with probabilistic rules	$\sharp P$-hard, in PSPACE	NP-hard	in PTIME (Thm. 4.3)
non-inflationary fixpoint with probabilistic rules	$\sharp P$-hard, in (2-)EXPTIME	NP-hard	NP-hard; PTIME in input size and mixing time (Thm. 5.1,5.6)

Table 1: Overview of complexity results (data complexity).

restricted variants of the query languages, and the algorithms apply to the full-fledged languages. We also discuss noninflationary probabilistic datalog (omitted from the table), whose expressive power is subsumed by that of non-inflationary fixpoint. Specifically, we show that our hardness results for non-inflationary fixpoint languages hold already for non-inflationary probabilistic datalog.

The paper is organized as follows. In Section 2 we recall the definitions of probabilistic databases as well as the notions of complexity and approximations used to measure the quality of query evaluation algorithms. In Section 3 we formally define our query languages, in inflationary and non-inflationary flavors; query evaluation for inflationary and non-inflationary queries is studied in Sections 4 and 5 respectively. We conclude with an overview of related work in Section 6.

2. PRELIMINARIES

This section provides common definitions that will be used in the remainder of the paper. First, we recall some common notions of computational complexity and approximation algorithms. These notions will be used to analyze our algorithms. Then, we give background on probabilistic databases. Last, we recall definitions and important properties of Markov Chains.

2.1 Complexity and Approximations

Complexity. At some parts of this paper we address the exact computation of probabilities, which is essentially a counting problem and its complexity analysis thus makes use of *counting complexity* [24]. Specifically, $\sharp P$ is the set of counting problems associated with decision problems in NP. More formally, a counting problem is in $\sharp P$ if there is a non-deterministic, polynomial-time Turing Machine whose number of accepting computations, for each instance I of the problem, is equal to the result count of I. A problem is $\sharp P$-hard if and only if every problem in $\sharp P$ can be reduced to it in polynomial time.

We further consider probabilistic algorithms, and recall in this context that BPP [18] (Bounded-error, Probabilistic, Polynomial time) is the class of all decision problems solvable by a probabilistic Turing machine in polynomial time, with an error probability of at most ϵ for all instances, for some $0 \leq \epsilon < 0.5$. Note that the choice of ϵ in this range may be arbitrary; any polynomial-time algorithm A with error probability bounded by ϵ may be transformed into one with error probability bounded by ϵ', by executing A multiple times and taking a "majority vote" of its answers. The Chernoff bound [7] guarantees that the number of required iterations is logarithmic in $\frac{1}{\epsilon}$.

When analyzing the complexity of query evaluation, one may consider *data complexity* and *combined complexity* [25]. Data complexity is the complexity of evaluating a query as a function of the *database* size, while combined complexity is a function of both the database and the query sizes. We focus here on data complexity, as, even for restricted versions of the query languages we study here, it has been shown that the combined complexity is $\sharp P$-hard. Thus, whenever we refer to the complexity of query evaluation, we mean data complexity.

Approximations. We use two notions of approximation, common in the literature, namely *absolute* and *relative* approximations [26]. An approximation algorithm A to a counting problem C takes as input an instance I of C and a parameter ϵ. Denote by $C(I)$ the correct answer for I, and by $A(I)$ the output of A when executed over I. We say that A is an *absolute* approximation if $|A(I) - C(I)| \leq \epsilon$ for every input instance I, ϵ. A is a *relative* approximation if $C(I) * (1 - \epsilon) \leq A(I) \leq C(I) * (1 + \epsilon)$ for every I.

A *randomized* absolute approximation algorithm A receives as input an additional parameter δ, and satisfies $Pr(|A(I) - C(I)| \leq \epsilon) > \delta$ for every input instance I, where Pr stands for probability. Analogously, a randomized *relative* approximation algorithm A satisfies $Pr(C(I) * (1 - \epsilon) \leq A(I) \leq C(I) * (1 + \epsilon)) > \delta$.

The corresponding decision problems ask whether a given value p is an absolute (relative) approximation of the correct answer $C(I)$, up to a given ϵ; when we provide in the sequel NP-hardness (data complexity) results for these problems, we mean NP-hardness of the corresponding decision problem (w.r.t. the database size).

2.2 Probabilistic databases

We use the definition of [16] for probabilistic databases, as follows. The schema used is a relational database schema; a probabilistic database over a schema with relation names $R_1, ..., R_k$ is a finite set of possible worlds

$$\{(R_1^1, \ldots, R_k^1, p^{[1]}), \ldots, (R_1^n, \ldots, R_k^n, p^{[n]})\}$$

with positive rational weights $p^{[i]}$ s.t. $\sum_i p^{[i]} = 1$.

We give here two examples for previously introduced models that allow the generation of probabilistic databases, namely *probabilistic c-tables* [13] and relational algebra enriched with a *repair-key* [16] construct.

Probabilistic c-tables. Probabilistic c-tables with finite variable domains are defined as follows.

Definition 2.1 ([13]). A c-table is a relation in which each tuple is associated with a condition. A condition is a boolean combination of (in)equalities involving variables and constants of some *finite* domain V. In a *probabilistic* c-table (pc-table), these variables are random variables. A probabilistic c-table is a pair of a c-table and the representation of a joint probability distribution of the random variables occurring in the c-table. The set of *possible worlds* of a probabilistic c-table is the set of possible valuations $\theta = (X_1 = x_1, \ldots, X_k = x_k)$ of its random variables. The *probability* of a world is the probability of the valuation and the *database* of the world is the set of tuples whose conditions are true for θ.

The probabilistic c-tables defined in this way are succinct means of representing any finite probabilistic database. One may fix without loss of generality that the random variables of a probabilistic c-table are independent. Thus, the joint distribution can be given by individual distributions for each of the random variable, assigning for each random variable X and each possible value x of X, a probability $\Pr[X = x]$. The joint distribution can be obtained as the product of the distributions of the individual random variables.

Repair-key. Let $\vec{A}, P$ be column names from the schema of a relation R, where $\vec{A}$ is a vector of columns and P is a single column, containing only numerical values which are all greater than zero.

The operation repair-key$_{\vec{A}@P}(R)$ [16], samples one maximal *repair* of key $\vec{A}$. That is, for each distinct value $\vec{a}$ of $\vec{A}$ appearing in tuples of R, denote the set of tuples in which $\vec{a}$ is the key value by $T_{\vec{a}}$. For each such $\vec{a}$, we sample exactly one tuple $\vec{t}$ from $T_{\vec{a}}$ with the probability distribution given by (normalized) column P [1]. That is, the probability of $\vec{t}$ to be chosen is $\frac{\vec{t}.P}{\sum_{\vec{t'} \in T_{\vec{a}}} \vec{t'}.P}$. The application of a repair-key construct generates a set of possible worlds (samples) with a single tuple for each key value; the probability of a possible world is the product of probabilities of the chosen tuples within their groups $T_{\vec{a}}$, that is, the groups are assumed independent.

EXAMPLE 2.2. Consider the simple relation R depicted in Table 2. Each tuple in this relation corresponds to a (possibly incorrect) fact regarding a basketball player and the team for which he currently plays. The third column represents the level of belief in this fact, i.e., that this player plays for this and no other team, to be considered in relation to the levels of belief in alternatives. If this database is constructed by accumulating the opinions of various users, this belief value is possibly derived from the popular support in the fact (i.e. the number of users that think that this fact is correct). The Player column constitutes a primary key here, but note that, still, there are multiple tuples sharing the same value for this attribute. Thus, the key should be repaired, meaning a single tuple (and thus, team) for each player should be chosen.

By applying repair-key$_{Player@Belief}(R)$, this choice is performed probabilistically, with probabilities determined by the relative belief in each option. That is, the probability of the tuple (Bryant, LA Lakers) to be chosen is $\frac{17}{17+3}$, and the probability of the tuple (Bryant, NY Knicks) to be chosen is $\frac{3}{17+3}$; similarly, for the tuples having "Iverson" as a player, one of them is chosen according to their relative belief values

Player	Team	Belief
Bryant	LA Lakers	17
Bryant	NY Knicks	3
Iverson	Philadelphia 76ers	8
Iverson	Memphis Grizzlies	7

Table 2: Basketball Players Table

($\frac{8}{8+7}$ and $\frac{7}{8+7}$). Each possible world database consists of a single fact for each key value, and its probability is the multiplication of the probability values over all tuples chosen to appear in the Database. □

We further exemplify the use of repair-key in Section 3.

The standard notion of relational algebra can be enriched with the repair-key construct. Each application of the repair-key generates a set of possible worlds, and further relational algebra operations are applied in each possible world independently. The result of evaluating such an enriched query Q over a relation R is a probabilistic database, and we denote it by $Q(R)$. $Q(R)$ is thus a set of relations, each corresponding to a possible world and associated with a probability value. Note that the repair-key operator is the natural probabilistic generalization of the witness operator – see e.g. [1]. Its expressive power was discussed in [16]. Any finite probabilistic database can be constructed from certain databases (that is, databases consisting of a single possible world with probability 1) using queries of this language. Specifically, with the repair-key construct, pc-tables as defined above may be simply viewed as "macros": the probabilistic choices generating possible worlds may be simulated by a polynomial number of repair-key construct applications. We further consider the connection between c-tables and the repair-key construct in the sequel.

For ease of presentation, we will use the following abbreviations: we optionally omit the column parameter P and write just repair-key$_{\vec{A}}(R)$. In that case, a tuple is chosen from its group $T_{\vec{a}}$ uniformly at random; that is,

$$\text{repair-key}_{\vec{A}}(R) := \text{repair-key}_{\vec{A}@P}(R \times \rho_P(\{1\})).$$

Here $\times$ is the standard relational product, and ρ is renaming. We also use

$$\text{repair-key}_{@P}(R) := \text{repair-key}_{\emptyset @P}(R)$$

for the choice of a single tuple from R, regardless of key values, with probability set for each tuple according to its relative value of P with respect to the sum of P values over all tuples of R. repair-key(R) chooses a single tuple out of R, uniformly.

2.3 Markov Chains

We next provide, in short, some definitions for Markov Chains and their properties. The reader is referred to [10] for details.

Markov Chains. A Markov Chain (MC) is a finite state machine (FSM) with transitions annotated by probabilities such that the sum of the probability annotations of the outgoing transitions of a state is 1. The probability of each transition in an MC does not depend on previous states of the process. A *random walk* over an MC starts at some arbitrary state of the MC and at each point randomly chooses,

[1]We assume here that there is a functional dependency in R : $schema(R) - P \to P$. Alternatively, assume a semantics of repair-key that first replaces any group of tuples $(\vec{b}, p_1), \ldots, (\vec{b}, p_n)$ that violates this functional dependency by a single tuple $(\vec{b}, \sum_i p_i)$.

according to the transition probabilities, a transition originating at its current state and continues from its target state.

A Markov Chain is *irreducible* if a random walk starting at any state i will eventually reach any state j with probability greater than 0. The *period* of a state s in a Markov Chain is a number k such that any return to a state i must occur in multiples of k steps. Namely, $k = gcd\{n : Pr(X_n = i|X_0 = i) > 0\}$, where X_i is the state after i steps of the walk. If $k = 1$ for all states, then the MC is said to be *aperiodic*. A state i is said to be *positively recurrent*, if, starting at i, the probability of eventually getting back to i in a finite number of steps is 1. An MC is positively recurrent if all of its states are positively recurrent. An MC is *ergodic* if it is aperiodic and positively recurrent.

Stationary Distributions. The *transition matrix* P is defined, for a given MC M, by Pij being the probability of transition from state i to state j. The *stationary distribution* π over an MC M is then a distribution over the state space of M, such that $\pi = \pi P$. Such distribution uniquely exists for M if and only if M is *irreducible* and *positively recurrent*.

Markov Chain Monte Carlo. Markov Chain Monte Carlo (MCMC) algorithms [14, 19] sample from a probability distribution π by constructing an ergodic MC M whose stationary distribution corresponds to π, performing a random walk over M, and using the distribution of nodes observed at a random point of the walk as an estimate for the underlying stationary distribution. To sample correctly, the Markov chain must be first *mixed*, meaning that the probability of the walk being at a state i is independent of the initial state of the walk, and is approximately π_i (the stationary probability of state i). When M is *ergodic*, this is guaranteed to occur if the walk is long enough.

The *mixing time* of an ergodic MC is thus defined as the number of steps it takes to "forget" the initial state, namely, for some given ϵ, it is the smallest $t(\epsilon)$ such that $|Pr(S_{t(\epsilon)} = i) - \pi_i| < \epsilon$ for each state i, where $S_{t(\epsilon)}$ is the state of the random walk after $t(\epsilon)$ steps. For an ergodic MC there exists such $t(\epsilon)$ for every value of ϵ.

3. LANGUAGE DEFINITIONS

In this section, we formally define our query languages and provide some examples of their use.

3.1 Noninflationary language

We start by considering standard relational databases, where first-order interpretations are generalized by a sampling extension based on the repair-key operation. Then we define non-inflationary queries based on this interpretation. We extend the discussion to probabilistic c-tables below.

Definition 3.1. A *probabilistic first-order interpretation* over relational schema $R_1, \ldots, R_k$ is a tuple $Q = (Q_1, \ldots, Q_k)$ of queries in relational algebra extended by the repair-key operation such that the result schema of each query Q_i is the schema of R_i. Given a relational database $\mathcal{A}$, such an interpretation defines a probabilistic database $Q(\mathcal{A})$ whose worlds are all $(\mathcal{A}_1, ..., \mathcal{A}_k, P)$ s.t. each $\mathcal{A}_i$ $i = 1, ..., k$ is a possible result (world) of $Q_i(\mathcal{A})$ and P is the multiplication of the probability values associated with each $\mathcal{A}_i$ in $Q_i(\mathcal{A})$.

Next we introduce our probabilistic analog of (noninflationary) while-queries (cf. e.g. [1]).

Definition 3.2. A *noninflationary query* or *forever-query* over a given database schema is a pair (Q, e), where Q is a probabilistic first-order interpretation of the schema, called the *transition kernel*, and e is a low-complexity Boolean relational database query, called the *query event*. We will assume that query events are of the form $\vec{t} \in R$, checking whether a given tuple $\vec{t}$ is in a relation of the current database state.

Conceptually, such a query is evaluated against an input relational database – the initial state – by the following program:

```
State := the input database;
forever {
    State := $Q$(State);
}
```

The query language semantics is as follows. Note that $Q(state)$ is in general a probabilistic database, i.e. is a set of possible worlds; the loop body is applied in every possible world, possibly generating new possible worlds etc. Intuitively, the application of the forever-loop corresponds to random walk over the database states, starting from the input database state, with probabilities of walking from state s to state s' dictated by the probability of s' among the possible worlds of $Q(s)$. The query result is the probability that the query event is true at an arbitrary point in time in the infinite random walk, i.e., the probability that after arbitrarily many steps, the current state satisfies $\vec{t} \in R$.

More formally, a *world sequence* in the forever-loop is a sequence of database states $seq = [s_1, ..., s_k]$ such that s_1 is the initial database state and s_i is a possible world of $Q(s_{i-1})$, associated with its probability p_i. We denote the length of seq by $len(seq)$. The probability of seq, denoted $Pr(seq)$, is $\prod_{i=1,...,k} p_i$. The probability of being in a given database state s, at an arbitrary point in time, in an infinite random walk over database instances, is then defined as

$$Pr(s) = \lim_{k \to \infty} \sum_{\{seq|len(seq)=k\}} Pr(seq) \cdot \frac{|\{i|s_i = s, 1 \leq i \leq k\}|}{k}.$$

Following [10], the limit exists and is finite. Last, the query result is the sum of probabilities $Pr(s)$ of all database states s satisfying $\vec{t} \in R$.

We note that Q, along with the initial database, define a Markov Chain M over database states. When M is ergodic, the query result is also the probability that $\vec{t} \in R$ according to the stationary distribution of M.

Example 3.3 (Random walk in a graph). We show how to express a random walk as a forever-query. Consider a scenario in which the input database consists of two input relations: a unary relation $C(I)$, consisting of a single tuple that stands for the start node of the walk, and a ternary relation $E(I, J, P)$ reflecting graph edges that are annotated with probabilities. That is, $(i, j, p) \in E$ iff there is an edge with probability weight p from node i to node j in the graph. p is the probability of performing a transition to node j, given that we are at a node i. We can then define a random walk by the interpretation Q:

$$\begin{array}{rcl} C & := & \rho_I \pi_J\big(\text{repair-key}_{I@P}(C \bowtie E)\big) \\ E & := & E \quad \% \text{ unchanged.} \end{array}$$

Recall that $\bowtie$ stands for the natural join, π for projection, and ρ for renaming. In each iteration, if the current tuple in C is i, then the query selects (by using repair-key) the next node j out of those for which $(i, j) \in E$; j is then the new tuple of C, while E remains constant in-between iterations. If the query event is $v \in C$ and E defines an ergodic Markov chain M, we are asking for the probability of v according to the stationary distribution of M.

To see a variant of this, let V denote the set of nodes in the graph (or, if E is viewed as a Markov chain, the states). V is obtained by a query over E, namely the union of projecting E on I, and projecting it on J (we do not consider isolated nodes). If we modify the first equation in the interpretation Q to be:

$$\begin{array}{l} C := \text{repair-key}_{@P} \\ \qquad \big(\rho_I(\pi_J(\text{repair-key}_{I@P}(C \bowtie E))) \times \rho_P(\{1-\alpha\}) \\ \qquad\qquad \cup\, \pi_I(\text{repair-key}_{I@P}(V)) \times \rho_P(\{\alpha\})\big) \end{array}$$

then we compute the *PageRank* at node v where α is the usual dampening factor, i.e., the probability that we abandon our current walk and jump uniformly to an arbitrary node in the graph.

Forever-queries over Probabilistic Databases. Above, query evaluation was defined for the case where the input database is a standard relational database. In some cases it is useful to consider *probabilistic* databases as input, e.g., given as a collection of probabilistic c-tables. As mentioned above, we can view such a pc-table as a macro for the corresponding algebraic expression that uses the repair-key construct. (To observe that this is possible see e.g. [16].) In particular, under the non-inflationary semantics, when such macros appear in the transition kernel, the probabilistic choices of tuples in the pc-table are made *in each iteration*. We will see examples for this in Section 5.

3.2 Inflationary queries

Inflationary queries are defined as the counterpart of the *while*$^+$ queries [1]. Intuitively, with inflationary queries, assignment of the new database state is cumulative rather than destructive: the query results are added to the current relations of the tuple, instead of replacing them as in the non-inflationary case. Formally, inflationary queries are defined as the following fragment of the noninflationary ones:

DEFINITION 3.4. A forever-query (Q, e) is called an *inflationary query* if for any relational database $\mathcal{A}$ and any possible world $\mathcal{B}$ of $Q(\mathcal{A})$, $\mathcal{B} \supseteq \mathcal{A}$.

We can define inflationary queries through transition kernels that define the new state as the union of the old state with the result of a query on the old state.

EXAMPLE 3.5 (REACHABILITY IN A GRAPH). We consider the same input database as in Example 3.3, where $E(I, J, P)$ is a ternary relation standing for the edge relation and C is an unary relation, initially consisting of a single tuple a, standing for the start node of the walk. Now, we express the query asking for the probability that node v is eventually reached in a random walk starting from the given start node of the walk. These nodes will be stored in C. We further use an additional auxiliary relation C_{old}, initially empty. In the inflationary language, Q is defined by the following probabilistic first-order interpretation:

$$\begin{array}{rcl} C_{old} & := & C \\ C & := & C \cup \rho_I \pi_J(\text{repair-key}_{I@P}(C - C_{old}) \bowtie E) \\ E & := & E \quad \% \text{ unchanged.} \end{array}$$

The query event is $v \in C$.

Note that the value of the C_{old} relation used in the second line is not the C_{old} value written in the first line, but its previous version. This is due to the semantics of query evaluation: at each point we fire all rules "in parallel", namely all righthand side expressions are evaluated against the old database state; only then the actual update takes place. □

There is a subtlety here that does not arise in non-probabilistic inflationary fixpoint queries: we have to be careful about the re-use of the same tuple multiple times, to generate new tuples. We illustrate this next.

EXAMPLE 3.6. Re-consider Example 3.3, and assume now that $E = \{(a, b, 0.5), (a, c, 0.5)\}$. Now $\Pr[b \in C] = 0.5$. However, if we replace the update rule for C by

$$C := C \cup \rho_I(\pi_J(\text{repair-key}_{I@P}(C \bowtie E))),$$

then we forever try to add, for each node in C, one node reachable from it via a single edge. Suppose $C = \{a, c\}$. Then we can either add b or c (which is already in C, so we add nothing) as successor of a. We may forever choose c as successor of a; but, the probability of this world goes to 0 as we continue iterating. In all other worlds, eventually $b \in R$. Thus $\Pr[b \in C] = 1$. □

The example demonstrates a general property: if we do not restrict the re-use of tuples for the generation of new tuples, then all tuples that appear in the result of the query without repair-key, appear in the query result with probability 1. The example also demonstrates that there are computation paths in which we do not reach a fixpoint in a finite number of steps (but the probability of each such path approaches 0).

Evaluating inflationary queries over Probabilistic Databases. We have explained above (see bottom of Section 3.1) that pc-tables are in fact only macros and may be simulated via rules containing a repair-key. Consequently, when such pc-tables (macros) appear in an inflationary query the semantics is different than in the case of non-inflationary queries: the probabilistic choices of tuples from the pc-tables now take place *only once*, at the beginning of query evaluation, rather than being repeatedly made as for the non-inflationary semantics. This is because, with inflationary semantics, rules fire only when there is a new valuations to the rules right-hand side; since pc-tables are implemented as repair-key application over ground facts, no such new valuation occurs during query evaluation.

3.3 Probabilistic Datalog with probabilistic rules

Probabilistic datalog syntactically extends datalog [1] by a repair-key construct. We use the following notation for the repair-key construct. In a rule head, the key columns are underlined, and the head is optionally postfixed by $@P$, where P is the Datalog variable binding to the repair-key weighting column (if omitted, the weighting is uniform, like for the repair-key algebra operator).

Example 3.7. Consider a relation I with schema $R(A, B, C, D, E)$. The rule

$$H(\underline{X}, \underline{Y}, Z)@P \leftarrow R(X, Y, Z, P, W)$$

corresponds to $\pi_{ABC}(\text{repair-key}_{AB@D}(\pi_{ABCD}R))$ in relational algebra extended by repair-key. □

Recall that each (standard) datalog rule can be compiled to an expression in relational algebra that computes the rule head [1]. For non-inflationary queries, given a probabilistic datalog query, defined by a probabilistic datalog program Q and a query event e, we may use the same translation mechanisms (with the addition of the @ operation, translated into the repair-key construct), to translate (Q, e) into an equivalent non-inflationary query (as defined in Def. 3.2). Under inflationary semantics, it is easy to observe that if at every step all possible valuations for the rule body are used for application of repair-key, then every probabilistic datalog program is equivalent to a conventional datalog program without repair-key, as all tuples that may appear in the output appear with probability 1 (see Example 3.6). We thus define the inflationary semantics of probabilistic datalog queries as follows.

```
Repeat forever
{
  In parallel, for each rule r:
                         $R(\underline{\vec{X}}, \vec{Y})@P \leftarrow B(\vec{X}, \vec{Y}, \vec{Z})$ do
  {
    newVals[r] := valuations of the body of r on
                  the old database state − oldVals[r];
    oldVals[r] := oldVals[r] ∪ newVals[r];
    R := R $\bigcup$ repair_key$_{\vec{X}@P}(\pi_{\vec{X},\vec{Y},P}$(newVals[r]));
  }
}
```

We recall that in datalog, the term IDB relation stands for a derived relation, and an EDB relation is an input relation. The above computation is applied over an initial database, where the EDB relations contain the data, the IDB relations are empty, and for each rule r (where B stands for the body of R), there are two auxiliary relations newVals[r] and oldVals[r] that are initially empty. If the body of a rule is empty, the single valuation of the body is the empty valuation and the rule thus fires only once, in the first iteration.

A probabilistic Datalog query must reach a fixpoint in each possible computation path since there are only polynomially many possible tuples over the active domain of the initial state. A rule in which all head variables are underlined is essentially non-probabilistic: it deterministically adds *all* the tuples that a classical Datalog rule would add.

Also note the similarly to the above explanations for the inflationary semantics, when a probabilistic Datalog query is evaluated over a pc-table, the pc-table may first be simulated by Datalog rules; these rules are fired only once since the bodies of these rules correspond to ground facts and thus have no new valuations throughout the iteration (if non-inflationary semantics is used, then the random choices for the pc-table are made in each iteration).

It is not hard to show the following proposition.

Proposition 3.8. *For every probabilistic datalog program, there is an equivalent inflationary query.*

We next express the reachability query (Example 3.5) in probabilistic datalog.

Example 3.9 (Reachability in a graph revisited). The query that computes Reachability of graph nodes from a start node v (see Example 3.5) can be expressed in probabilistic datalog simply as

$$\begin{aligned} C(v) &\leftarrow \\ C2(\underline{X}, Y) &\leftarrow C(X), E(X, Y). \\ C(\underline{Y}) &\leftarrow C2(X, Y). \end{aligned}$$

Consider the evaluation of the above query over an initial database $E = \{(v, w, 0.5), (v, u, 0.5)\}$. It first adds the tuple v to C, as the (empty) valuation for the body of the first rule was not used yet. Then, there exist two new valuations, $\{X = v, Y = w\}$ and $\{X = v, Y = u\}$ one of which is chosen (with probability of 0.5 for each choice). Let W be the world (bearing a probability of 0.5) where $X = v, Y = w$ (respectively, $X = v, Y = u$) was chosen. Now $X = v, Y = u$ (respectively, $X = v, Y = w$) is no longer a new valuation (because it was a valid valuation in the previous iteration as well), and then the third rule must fire, leading to the addition of the tuple u (w) to C in this world. Note that the use of $C2$ is essential to enforce the application of repair-key over each of the nodes appearing currently in C, modeling the probabilistic choice of the next nodes. If we would only write $C(\underline{y}) \leftarrow C(x), E(x, y)$, no probabilistic choice of valuations of the body were made. Instead, all valuations would be used as in classical datalog. □

We next consider a more complicated example, namely computation of marginal distributions for a given Bayesian Network.

Example 3.10 (Bayesian Network). In this example, we construct the joint distribution over a number of Boolean random variables as defined by a Bayesian Network. We use probabilistic Datalog with repair-key. For simplicity, we assume that there is an upper bound K on the in-degree of each node in the Bayesian network, i.e., the number of parent variables of each random variable. The input database consists of two input relations that specify the Bayesian network, a relation with schema $S_k(N_0, \ldots, N_k)$ which specifies that the random variable named N_0 has exactly the parents $N_1, \ldots, N_k$ and a relation with schema $T_k(N_0, V_0, V_1, \ldots, V_k, P)$ that specifies the tuples of the conditional probability tables. That is, if the parent variables of X are called $Y_1, \ldots, Y_k$ (i.e., $(X, Y_1, \ldots, Y_k) \in S_k$) and if tuple $(X, x, y_1, \ldots, y_k, p)$ is in T_k, where X is the name of a random variable, $x, y_1, \ldots, y_k \in \{0, 1\}$ are the values of random variables $X, Y_1, \ldots, Y_k$, and $p \in [0, 1]$, then $\Pr[X = x \mid Y_1 = y_1, \ldots, Y_k = y_k] = p$.

There is a single IDB predicate $V(N, V)$ which specifies a complete valuation of each random variable N in each possible world.

The probabilistic datalog program consists of $K+1$ rules (for $0 \leq k \leq K$):

$$\begin{array}{rcl} V(\underline{N_0}, V_0)@P & \leftarrow & T_k(N_0, V_0, V_1, \ldots, V_k, P), \\ & & S_k(N_0, N_1, \ldots, N_k), \\ & & V(N_1, V_1), \ldots, V(N_k, V_k). \end{array}$$

Now, marginal probabilities such as $\Pr[X = x \wedge Y = y]$ can be computed as the probability of event q if rule

$$q \leftarrow V(X, x), V(Y, y).$$

is added to the datalog program. Here X, x, Y, y are constants, not datalog variables.

In the remainder of the paper, we will consider both inflationary and non-inflationary datalog. We also study datalog restricted in one or both of the following ways: datalog without repair-key applied over probabilistic c-tables, and linear datalog, which is datalog in which each rule body contains at most one IDB atom.

4. COMPLEXITY OF THE INFLATIONARY LANGUAGES

We next discuss the complexity of query evaluation with respect to the inflationary semantics. It is easy to show (following e.g. [20]) that exact query evaluation (i.e., asking whether the probability of the given query event is exactly p) is $\sharp P$-hard. We thus turn to approximations, and first consider *relative approximation.* Unfortunately we can show that such approximation is infeasible as well, as the following theorem holds.

THEOREM 4.1. *Unless $P = NP$ ($BPP = NP$), there exists no PTIME deterministic (randomized) relative approximation scheme for the query evaluation under the inflationary semantics. This holds even if (1) the problem is restricted to evaluation of Linear Datalog Programs and (2) either repair-key is applied only on base relations, or (2') the queries are without repair key and applied over probabilistic c-tables.*

PROOF. We prove the theorem for the case where conditions (1) and (2') above hold; the proof for the case where (1) and (2) hold follows, as discussed below.

We use a reduction from 3-SAT. Given a 3-CNF formula F consisting of clauses $\{c_1, ..., c_m\}$ over a set $\{v_1, ..., v_n\}$ of variables, we construct a database D and a Datalog program Q as follows.

Database. The database D consists of the following EDB relations: $C(C, L)$, $O(C_1, C_2)$, $A(L)$, and $R(C)$, and its tuples are initialized as follows. $A(L)$ is a probabilistic c-table, where tuples stand for all variables in V and their negation; each tuple in A is associated with a predicate such that $A(v_i)$ is associated with $x_i = 0$, $A(\neg v_i)$ is associated with $x_i = 1$ for each variable $v_i \in V$, with $Pr(x_i = 0) = Pr(x_i = 1) = 0.5$ for each i, and all x_i variables are independent. The relation O contains a tuple $O(c_i, c_{i+1})$ for each $i = 1, ..., m-1$, and the relation C contains a tuple $C(c_i, l_j)$ for each clause c_i and a literal l_j that appears in c_i.

Program. We define the following program Q:

$$\begin{array}{rcl} R(c_0) & \leftarrow & \\ R(c) & \leftarrow & R(c'), O(c', c), C(c, l), A(l) \\ Done(a) & \leftarrow & R(c_n) \end{array}$$

We next show that existence of PTIME relative approximation algorithms (deterministic and randomized) for evaluating $(Q, a \in Done)$ over D implies a PTIME algorithm for deciding whether F has a satisfying assignment. We start with the following lemma:

LEMMA 4.2. *Denote by p the query result of $(Q, a \in Done)$ over D. If F is satisfiable, then $p \geq \frac{1}{2^n}$ where n is the number of variables in F. Otherwise, $p = 0$.*

PROOF. Consider a probabilistic choice for the tuples of A. Such choice must correspond to a consistent assignment, as for each variable v_i exactly one out of $A(v_i)$ and $A(\neg v_i)$ holds.

Assume now that F has no satisfying assignment. Thus, for each probabilistic choice for the tuples of A, there exists some $c_i \in C$ such that for all values of l, $C(c_i, l), A(l)$ does not hold. Thus the rule for $R(c_i)$ will not fire, and consequently $R(c_{i+1}), ..., R(c_n)$ will not appear in the database as well, and neither will $Done(a)$. This holds for all probabilistic choices, thus $p = 0$.

Conversely, assume that F has at least one satisfying assignment, then there exists a choice for the tuples of A corresponding to this assignment, and this choice bears the probability of $\frac{1}{2^n}$. For such choice, in the i-th step of evaluating Q, $R(c_i)$ holds; to observe this, note that at the first step $R(c_0)$ holds, and assume that at the $(i-1)$-th step $R(c_{i-1})$ holds. $O(c_{i-1}, c_i)$ holds by the DB initialization, as well as $C(c_i, l), A(l)$ for some literal l that satisfies c_i (note that the assignment is satisfying, thus such l must exist), thus in at the i'th step $R(c_i)$ holds. Consequently, at the n'th step $R(c_n)$ holds, thus $Done(a)$ holds in the next step. This happens for each choice of tuples corresponding to a satisfying assignment; each such satisfying assignment is chosen with probability $\frac{1}{2^n}$, thus we obtain $p \geq \frac{1}{2^n}$. $\square$

We next use Lemma 4.2 to prove Theorem 4.1. We note that the proof is rather straightforward following Chernoff's lower bound, but we repeat it for completeness. First, assume the existence of a deterministic query evaluation algorithm with relative error ϵ, then we simply apply it and conclude that F is satisfiable if and only if the algorithm's output was non-zero. For the second part of the Theorem, assume that we have a randomized query evaluation algorithm A; then we run it N times independently, and decide that F is unsatisfiable if the majority of the executions return 0. We next show that the construction yields a BPP algorithm for 3-SAT. We say that an invocation of A "succeeds" if it returns 0 and the formula F is indeed not satisfiable, or if it returns a number other than 0 and F is indeed satisfiable. There exists a δ such that each invocation of A succeeds with probability of at least $1 - \delta$, and fails with probability of at most δ. Define X_i to be a binary random variable that represents success or failure of the i'th invocation of A, and let $X = X_1 + ... + X_N$. We next bound the probability that $X < \frac{N}{2}$. Since the invocations are independent, X bears a binomial distribution, $E(X) = N * (1 - \delta)$, and by Chernoff's Lower bound $P(X < k * E(x)) < e^{-E(x) * \frac{(1-k)^2}{2}}$.

Set $k = \frac{N}{2*E(x)}$ to obtain $P(X < \frac{N}{2}) < e^{-E(x)*\frac{(1-\frac{N}{2*E(x)})^2}{2}}$. $1 - \frac{N}{2*E(x)} = 1 - \frac{1}{2*(1-\delta)}$, and denote this fraction by β. β is positive as $1 - \delta > 0.5$. The bound now translates into $e^{-E(x)*\frac{\beta^2}{2}} = e^{-N*(1-\delta)\frac{\beta^2}{2}}$. To achieve any error bound Γ, we simply set N such that $-N*(1-\delta)*\beta^2 < 2*ln\Gamma$, i.e. $N > \frac{ln\frac{1}{\Gamma^2}}{(1-\delta)*\beta^2}$.

We have proven Theorem 4.1 for the case where conditions (1) and (2') specified in the Theorem hold. We next show that the Theorem also holds for the case where (1) and (2) holds: in this case, we change our construction by replacing the c-table A by a similar table $A(V, L, P)$, whose tuples are $A(i, v_i, 0.5)$, $A(i, \neg v_i, 0.5)$ for each i; using repair-key we then choose a single such tuple for each i, and proceed as above.

However, we may show that randomized absolute approximation may be achieved in PTIME.

Theorem 4.3. *Randomized absolute approximation is in PTIME (data complexity) for the inflationary fixpoint language with repair-key, even when applied on probabilistic c-tables.*

Proof Sketch. We employ a sampling algorithm that for every given ϵ, δ, approximates the query result up to ϵ with probability of at least δ, as follows. Each sample is done by randomly choosing a single value for each independent variable in the c-table according to its probability; then, we consecutively apply the query rules, making probabilistic choices for each application of the repair-key construct that appear within these rules (i.e. we sample a possible repair, according to its probability dictated by the current database status and the repair-key operation), until reaching a fixpoint. At the end of each sample, we check for satisfaction and declare the probability p to be the average number of satisfactions over all samples.

Each such sample can be done in PTIME (data complexity), as the applications sequence entails the same number of steps as evaluation of non-probabilistic Datalog (with a linear time overhead incurred by the random choices of values for the independent variables).

To bound the number of samples required for an ϵ, δ-approximation we use the Chernoff bound. Denoting by x the random boolean variable whose value indicates query satisfaction or dissatisfaction, the output p of our algorithm is the average of m samples of x. Denote by $\hat{p}$ the correct probability of x (the query probability), the (additive) Chernoff bound gives $Pr(\mid p - \hat{p} \mid \geq \epsilon) \leq 2 * e^{-2*\epsilon^2*m}$ where m is the number of samples. So choosing m s.t. $-2*\epsilon^2*m \leq \frac{ln(\delta)}{2}$, i.e. $m \geq \frac{ln(\frac{1}{\delta})}{4*\epsilon^2}$ samples is sufficient. The overall complexity is thus polynomial in the database size and in ϵ, and logarithmic in $\frac{1}{\delta}$. □

To complete the picture, we may exactly evaluate inflationary fixpoint queries in PSPACE. (Recall that exact evaluation of even simple select-project-join queries over Probabilistic Databases [20], and consequently of queries with repair-key, even without recursion, is $\sharp P - hard$ [16].)

Proposition 4.4. *Exact evaluation of inflationary fixpoint queries with the repair-key construct (even over probabilistic c-tables) is in PSPACE.*

Proof Sketch. Consider the following algorithm for computing the probability that a given tuple is in the fixpoint of a probabilistic datalog query with the repair-key construct. We iterate through possible worlds of the input probabilistic database (specified by a probabilistic c-table; to iterate through the possible worlds we only need to iterate through the valuations of the independent random variables, which is easy to do in PSPACE).

For each possible world we iteratively evaluate the query. In each iteration, the program may choose one out of (exponentially) many increments to the database state. However, due to the independence of choice across key valuations and rules, we can iterate through these alternatives using only a polynomial amount of memory. Thus we can make a full traversal through the tree of possible computations down to all the fixpoints. This tree has exponentially many nodes, but its depth is bounded by the number of tuples a fixpoint can consist of at most, which is polynomial in the active domain of the input database. We only have to store at most a path in this computation tree from the root to a fixpoint, i.e., polynomially many databases.

The result probability is initially zero. While we traverse this tree, whenever we reach a leaf (a fixpoint), we test the query event and, if it is true on that fixpoint, add its probability weight to the result probability. □

5. COMPLEXITY OF THE NON-INFLATIONARY LANGUAGE

We next consider the evaluation of non-inflationary queries. We have shown in the previous section that, for inflationary queries, exact computation is $\sharp P$-hard with respect to data complexity and that unless $P = NP$, a PTIME relative approximation is also impossible. From this, it follows that the same is true for noninflationary queries, of which inflationary queries are a special case. As for absolute approximation, we have shown above that such approximation is feasible for the inflationary semantics; unfortunately, we may show that this is not the case for non-inflationary queries, as the following theorem holds.

Theorem 5.1. *Unless $P = NP$ ($BPP = NP$), there is no PTIME deterministic (randomized) absolute approximation for non-inflationary queries. This holds even if (1) the problem is restricted to evaluation of Datalog programs and (2) either repair-key is applied only on base relations, or (2') the queries are without repair key and applied over probabilistic c-tables.*

Proof Sketch. Similarly to the proof of Theorem 4.1, we use here a reduction from 3-SAT, but this time use the expressive power of non-inflationary queries to show that even an absolute approximation is impossible. Again, we prove the theorem for the case where conditions (1) and (2') above hold; the proof for the case where (1) and (2) hold follows in exactly the same manner as explained for Theorem 4.1. Similarly, we show the proof for the deterministic case; the proof for the randomized case again follows that of Theorem 4.1.

Recall first that under non-inflationary semantics, tuples from pc-tables are repeatedly sampled, in each iteration.

Given a 3-CNF formula F consisting of clauses $\{c_1, ..., c_m\}$ over a set $\{v_1, ..., v_n\}$ of variables, we construct a database and a Datalog program as follows.

Database. D consists of the following relations: $C(C, L)$, $O(C_1, C_2)$, $A(L)$, and $R(C, X)$, and initialize its tuples as follows. As in the proof of Theorem 4.1, $A(L)$ is a probabilistic c-table where tuples stand for all variables in V and their negation; each tuple in A is associated with a predicate such that $A(v_i)$ is associated with $x_i = 0$, $A(\neg v_i)$ is associated with $x_i = 1$ for each variable $v_i \in V$, with $Pr(x_i = 0) = Pr(x_i = 1) = 0.5$ for each i, and all x_i variables are independent.

$O(C_1, C_2)$ bears a tuple $O(c_i, c_{i+1})$ for each $i = 1, ..., m-1$; and $C(C, L)$ bears a tuple $C(c_i, l_j)$ for each clause c_i and a literal l_j that appears in c_i.

Program. Consider the following program Q:

$$
\begin{array}{rcl}
R(c_0, l) & \leftarrow & A(l) \\
R(c_k, l) & \leftarrow & R(c_{k-1}, l), R(c_{k-1}, l'), O(c_{k-1}, c_k), C(c_k, l') \\
Done(a) & \leftarrow & R(c_n, .) \\
Done(x) & \leftarrow & Done(x)
\end{array}
$$

We employ an approximation algorithm for estimating the probability of $(Q, a \in Done)$ over D up to an absolute error of 0.5 , and output that F is satisfiable if and only if the result is greater than 0.5. The correctness of the construction follows from the next lemma:

Lemma 5.2. *Denote by p the result of evaluating $(Q, a \in Done)$ over D. It holds that $p = 1$ if F is satisfiable, and $p = 0$ otherwise.*

Proof Sketch. For each probabilistic choice of tuples for A, the set of literals $\{l \mid R(c_0, l)\}$ corresponds to a randomly chosen assignment α, to the variables $\{v_1, ..., v_n\}$ (note that for each variable x, a single literal out of x and $\neg x$ is chosen randomly). In what follows we relate to a set of literals α (without contradictions, i.e., α does not include both a variable and its negation) as an assignment, and say that such a set α' is consistent with α if the set of literals corresponding to α' is a subset of those corresponding to α.

We then use the following auxiliary proposition (proved by induction on i).

Proposition 5.3. *In the i-th iteration of evaluating Q over D, $\{l \mid R(c_i, l)\}$ corresponds to an assignment α' consistent with α and satisfying $c_1, ..., c_i$, if such exists; if none exists, $R(c_i, l)$ does not hold for any l.*

It follows from Proposition 5.3 that after the n-th iteration, $\{l \mid R(c_n, l)\}$ is non-empty iff there is a satisfying assignment consistent with the initial α, for $c_1, ..., c_n$, i.e. for F. Then, in the $(n+2)$-th iteration, $Done(a)$ holds; the last rule of the program guarantees that from this point on, $Done(a)$ remains in the database forever.

As the set of possible instantiations of the repair-key corresponds to all possible consistent assignments, it follows that if a satisfying assignment α exists, it will be generated by repair-key at some step k of the evaluation; after additional $n + 2$ iterations, as explained above, $a \in Done$ holds. Thus in all iterations from $k+n+2$ and on, $a \in Done$ holds, hence $Pr(a \in Done) = 1$ if F is satisfiable; conversely, if there is no satisfying assignment then $Pr(a \in Done) = 0$.

This completes the proof of Lemma 5.2. □

Thus, the existence of a PTIME (BPP) absolute approximation algorithm, with $\epsilon < 0.5$, provides a PTIME algorithm for 3-SAT (F is satisfiable iff the algorithm returned $p > 0.5$), and implies $P = NP$ ($BPP = NP$). □

However, we may show that under some restrictions on the transition kernel of the query, an EXPTIME exact evaluation of non-inflationary queries is possible.

Proposition 5.4. *Let $q = (Q, e)$ and let D be the input database, such that the Markov Chain of database instances, defined by Q, D, is irreducible and positively recurrent. Exact evaluation of q over D is in EXPTIME.*

Proof Sketch. There are polynomially many possible tuples over the active domain of the input database and thus only exponentially many states in the Markov chain defined by Q. Thus we can compute the stochastic matrix defining the transition relation of this Markov chain in EXPTIME by evaluating Q on each of the states. For each state, we compute a representation of the result of Q on that state as a probabilistic c-table. This is feasible in polynomial time per state. Then we extract the reachable states and their probability weights from this representation, yielding the matrix of the Markov chain. Next we run Gaussian elimination on this matrix to compute the principal eigenvector. This is feasible in cubic time in the size of the matrix. Finally, we sum up the weights in the eigenvector of those database states on which the query event is true. This is the query result. It is easy to see that this result can be computed in exponential time overall. □

We may further generalize our query evaluation algorithm, and obtain

Theorem 5.5. *The exact evaluation problem for non-inflationary queries is in 2-EXPTIME.*

Proof. We denote by M the Markov Chain induced by the transition kernel and the input database, and consider two cases. If M consists of a single strongly connected component, then it is irreducible and positively recurrent [10], and we may apply the same construction as in Proposition 5.4 to compute the probability of query satisfaction. Otherwise, we can compute the DAG of its strongly connected components. With probability 1, a random walk will eventually (after a finite number of steps) get to a component that is one of the leaves of this DAG and stay there forever. We can thus compute probabilities of getting to each of these leaves (by considering all paths that lead to each such leaf). This computation may be exponential in the size of the DAG, which in turn may be exponential in the database size, leading to the 2-EXPTIME complexity. Then we can perform Gaussian elimination on each of the connected components that reside in the leaves, and compute the probability of being at each individual node. This probability is factorized by the probability of getting to the connected component. □

It is open if a lower complexity (e.g. EXPTIME) is possible in the general case.

5.1 Improving Performance

We next present two simple ideas that do not improve the worst case complexity of non-inflationary query evaluation in general but may improve its performance in real-life cases. First, we consider the partitioning of query evaluation into smaller instances that may be solved independently using the techniques suggested above. Second, we consider a different technique for query evaluation that is based on sampling.

Partitioning. In many real-life cases of large programs, many derived tuples are "independent", in the sense that the derivation of one tuple does not change the probability of deriving another. We thus employ a pre-processing stage, as follows: we start with the original database and set a unique identifier to each tuple, which is a singleton set. Then we evaluate all rules in an inflationary manner (as in "regular" datalog on non-probabilistic databases), keeping provenance for each added tuple. That is, whenever a tuple is added to the database due to an application over tuples with identifiers $I_1, ..., I_k$, the new tuple is set the identifier $I_1 \bigcup I_2 .... \bigcup I_k$. At the end of the process, the sets of all identifiers that appear in the database and are not subsumed by any other identifier are the partition classes.

We now partition the database into tuple sets according to the classes obtained above, and consider the Markov Chain induced by each, as in Theorem 5.5 above. We independently find the probability of each state within it, and sum up the probabilities of being at states where the query does *not* hold. Last, the overall probability p that the query does not hold is the multiplication of such probabilities over all classes, and the query satisfaction probability is $1 - p$.

Sampling. We have shown above that approximating query results for non-inflationary queries is $\sharp P$-hard in general. However, in some cases an approximation may be obtained by *sampling* the possible worlds and computing their probability as the observed percentage of query satisfaction. The difficulty lies in obtaining *independent* samples. To that end, recall from Section 2 the definition of *mixing time* of a Markov Chain. Given a non-inflationary query $q = (Q, e)$ and a database instance D, such that Q and D induce an ergodic Markov Chain M, denote by $T(q, D)$ the mixing time of M. We can then prove the following proposition.

THEOREM 5.6. *A randomized absolute approximation algorithm for evaluation of a non-inflationary q over D, such that Q and D induce an ergodic Markov Chain, may be done in Polynomial Time in the size of D and in $T(q, D)$ (and exponential in the size of q).*

Proof Sketch. The algorithm generates independent samples as follows. It applies the transition kernel Q step by step, each time computing intermediate probabilities for query results up until convergence. The number of steps required for convergence is $T(q, D)$. When convergence is achieved, we omit all computed probabilities and start the actual sampling; we re-start to obtain further samples. Once we have independent samples in hand, the proof continues as in Proposition 4.3. $\square$

There are several techniques studied in the literature (e.g., conductance and coupling [19]) for characterizing Markov Chains with mixing time that is polynomial in the number of states of the chain. In such cases, approximated query evaluation may be performed in PTIME. Identifying syntactic restrictions on probabilistic Datalog that are the counterparts of such characterizations is an intriguing topic of future research.

6. CONCLUSION AND RELATED WORK

We conclude with a brief overview of related work.

Probabilistic Databases, in various flavors, were studied extensively (e.g. [20, 13, 2, 21]). The probabilities there are given as part of the database; [16] suggests a generalized model where randomization may be introduced as part of the query, via the use of the repair-key construct, and shows that a PTIME approximation is possible for (the positive fragment of) the query language studied. In contrast to our work, the query language studied in [16] does not allow recursion, and is thus less expressive. In particular, it cannot express probabilistic processes of unbounded length (e.g. random walks) studied here.

A (restricted) use of Datalog over probabilistic input data was studied in [11], as a tool for Information Retrieval applications. The setting studied in [11] allows to introduce probabilities only over the ground facts, and consequently the choice of "possible world" is in fact performed once, to choose the set of ground facts. Consequently, the model is less expressive than proposed here, where new worlds may be generated arbitrarily many times. In particular, the model of [11] does not allow to capture random walks in-between database instances, and their applications depicted above. Similarly, a restrictive probabilistic version of Prolog [9] allows to specify, for each rule, the probability that it belongs to a randomly sampled program. But again, the semantics is that a program is sampled only once.

We have studied here highly expressive query languages with an iteration construct that allow to perform probabilistic changes to the database (state). We have illustrated how these query languages may be used to declaratively specify random walks over complex Markov Chains, and other stochastic processes, and studied the complexity of query evaluation for the proposed query languages. Future work includes the design of generic optimization techniques for query evaluation, the study of restricted versions with lower complexity of query evaluation, and specifically syntactic counterparts of cases that allow for efficient sampling algorithms.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] P. Agrawal, O. Benjelloun, A. Das Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom. "Trio: A System for Data, Uncertainty, and Lineage". In *VLDB*, 2006.

[3] L. Antova, C. Koch, and D. Olteanu. "From Complete to Incomplete Information and Back". In *Proc. SIGMOD*, 2007.

[4] L. Antova, C. Koch, and D. Olteanu. "Query Language Support for Incomplete Information in the MayBMS System". In *Proc. VLDB*, 2007.

[5] V. Bansal. *"Computational Methods for Analyzing Human Genetic Variations"*. PhD thesis, University of California, San Diego, 2009.

[6] O. Benjelloun, A. D. Sarma, C. Hayworth, and J. Widom. "An Introduction to ULDBs and the Trio System". *IEEE Data Engineering Bulletin*, 2006.

[7] D. P. Bertsekas and J. N. Tsitsiklis. *Introduction to Probability.* MIT Press, 2008.

[8] N. Dalvi and D. Suciu. "Efficient query evaluation on

probabilistic databases". *VLDB Journal*, **16**(4):523–544, 2007.

[9] L. De Raedt, A. Kimmig, and H. Toivonen. "ProbLog: A Probabilistic Prolog and Its Application in Link Discovery". In *IJCAI*, 2007.

[10] D. Freedman. *Markov Chains.* Springer-Verlag, 1983.

[11] N. Fuhr. "Probabilistic Datalog – A Logic For Powerful Retrieval Methods". In *Proc. SIGIR*, pages 282–290, 1995.

[12] M. Goetz and C. Koch. "A Compositional Framework for Complex Queries over Uncertain Data". In *Proc. ICDT*, 2009.

[13] T. J. Green and V. Tannen. "Models for Incomplete and Probabilistic Information". *IEEE Data Eng. Bull.*, 29(1):17–24, 2006.

[14] M. Jerrum and A. Sinclair. "The Markov chain Monte Carlo method: an approach to approximate counting and integration". *Approximation algorithms for NP-hard problems*, 1997.

[15] C. Koch. "Approximating Predicates and Expressive Queries on Probabilistic Databases". In *Proc. PODS*, 2008.

[16] C. Koch. "On Query Algebras for Probabilistic Databases". *SIGMOD Record*, **37**(4):78–85, 2008.

[17] C. Koch. "A Compositional Query Algebra for Second-Order Logic and Uncertain Databases". In *Proc. ICDT*, 2009.

[18] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[19] D. Randall. "Mixing (a tutorial on Markov Chains)". In *FOCS*, 2003.

[20] C. Re, N. Dalvi, and D. Suciu. "Efficient Top-k Query Evaluation on Probabilistic Data". In *ICDE*, 2007.

[21] P. Sen and A. Deshpande. "Representing and Querying Correlated Tuples in Probabilistic Databases". In *ICDE*, 2007.

[22] D. Sorensen and D. Gianola. *"Likelihood, Bayesian, and MCMC Methods in Quantitative Genetics".* Springer-Verlag, New York, July 2002.

[23] Stanford Trio Project. *"TriQL – The Trio Query Language"*, 2006.

[24] L. Valiant. "The complexity of computing the permanent". *Theoretical Computer Science*, 8(2):189–201, 1979.

[25] M. Y. Vardi. "The Complexity of Relational Query Languages". In *Proc. STOC*, pages 137–146, 1982.

[26] V. V. Vazirani. *Approximation Algorithms.* Springer, 2004.