Lineage Enabled Query Answering in Uncertain Knowledge Bases

Masterarbeit im Fach Informatik
Master’s Thesis in Computer Science
von / by
Javeria Iqbal

angefertigt unter der Leitung von / supervised by
Dr. Martin Theobald

begutachtet von / reviewers
Dr. Martin Theobald
Dr.-Ing. Sebastian Michel

August 2011
Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum ___________________________ Unterschrift ___________________________
Abstract

We present a unified framework for query answering over uncertain RDF knowledge bases. Specifically, our proposed design combines correlated base facts with a query driven, top-down deductive grounding phase of first-order logic formulas (i.e., Horn rules) followed by a probabilistic inference phase. In addition to static input correlations among base facts, we employ the lineage structure obtained from processing the rules during grounding phase, in order to trace the logical dependencies of query answers (i.e., derived facts) back to the base facts. Thus, correlations (or more precisely: dependencies) among facts in a knowledge base may arise from two sources: 1) static input dependencies obtained from real-world observations; and 2) dynamic dependencies induced at query time by the rule-based lineage structure of the query answer.

Our implementation employs state-of-the-art inference techniques: We apply exact inference whenever tractable, the detection of shared factors, shrinkage of Boolean formula when feasible, and Gibbs sampling in the general case. Our experiments are conducted on real data sets with synthetic expansion of correlated base facts. The experimental evaluation demonstrates the practical viability and scalability of our approach, achieving interactive query response times over a very large knowledge base. The experimental results provide the success guarantee of our presented framework.
ACKNOWLEDGMENT

First and the foremost, I thank Allah almighty who provided me the strength, health and wisdom to complete this work. It is an honor for me to work with Dr. Martin Theobald and I owe my deepest gratitude to him for providing me the opportunity to work with him and for his consistent help and suggestions. It would not have been possible without his guidance, feedback and suggestions. I am also grateful to Mrs. Jennifer Gerling and Mrs. Stephanie Jorg of the International Max Planck Research School for Computer Science for their help. I am grateful to my friends at MPI and Saarland University. Specifically, I would like to thank my friend Iftikhar Ahmed who acted as my brother away from home. I deeply appreciate his consistent support, cooperation and encouragement during my downtimes. Lastly, but most importantly, my parents and siblings, who has always been a source of motivation and encouragement. I dedicate this thesis to my beloved parents.
## Contents

1 Introduction 1

1.1 Motivation and Problem Statement 1

1.2 Contributions 4

1.3 Outline 4

2 Related Work 6

2.1 Deductive Databases and Datalog 6

2.2 Probabilistic Databases 7

2.3 ULDBs and Trio System 8

2.4 Probabilistic Graphical Models 10

2.5 Markov Logic Networks 11

2.6 URDF 11

3 Lineage Enabled Deductive Reasoning with Correlations 14

3.1 Preliminaries 14

3.1.1 Probabilistic Graphical Models 14

3.1.2 Factor Graphs 15

3.2 Our Design 16

3.2.1 Unified Model: Combine Directed and Undirected 16

3.2.2 Data and Representation Model 17

3.2.3 Data Base Schema Modeling 23

3.2.4 Probabilistic Inference over Query Lineage Models 24

3.2.5 Intermediate Factors 27

3.2.6 Correlated Base Facts 30

4 Optimizations 36

4.1 Boolean Formula Shrinkage 36

4.2 Shared Factors Reuse 37

5 Experimental Evaluation and Results 38

5.1 Experimental Settings 38

5.1.1 System Specification 38

5.1.2 Data Sets 38

5.1.3 Experimental Categorization 39
# List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Sample x-tuple</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Sample maybe x-tuple</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Confidence values for sample x-tuples</td>
<td>9</td>
</tr>
<tr>
<td>3.1</td>
<td>Factor Table for A</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>Factor Table for B</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>Factor Table for C</td>
<td>20</td>
</tr>
<tr>
<td>3.4</td>
<td>Factor Table for D</td>
<td>21</td>
</tr>
<tr>
<td>3.5</td>
<td>Sample Tuples in the Facts Table</td>
<td>23</td>
</tr>
<tr>
<td>3.6</td>
<td>Sample Tuples in the Factor Tables Table</td>
<td>24</td>
</tr>
<tr>
<td>3.7</td>
<td>Sample Tuples in the Factors Table</td>
<td>24</td>
</tr>
<tr>
<td>3.8</td>
<td>Sample Data in Factor Table</td>
<td>30</td>
</tr>
<tr>
<td>3.9</td>
<td>Sample Data in Factors</td>
<td>30</td>
</tr>
<tr>
<td>3.10</td>
<td>Truth Table With Full Joint Distribution</td>
<td>32</td>
</tr>
<tr>
<td>5.1</td>
<td>Comparative Analysis of Runtime for Different Queries with Varying % DOC</td>
<td>41</td>
</tr>
<tr>
<td>5.2</td>
<td>Runtime, Number of Hops, Result Sets, Random Variables with [81-100] % DOC</td>
<td>42</td>
</tr>
<tr>
<td>5.3</td>
<td>Grounding Time, Processing Time, Execution Time for 30 FTs</td>
<td>46</td>
</tr>
<tr>
<td>5.4</td>
<td>Grounding Time, Processing Time, Execution Time for 50 FTs</td>
<td>46</td>
</tr>
<tr>
<td>5.5</td>
<td>Execution Time vs. Varying Number of Factor Tables</td>
<td>46</td>
</tr>
<tr>
<td>5.6</td>
<td>Base Facts, Result Sets, Sampling Time vs. varying number of Factor Tables</td>
<td>47</td>
</tr>
<tr>
<td>5.7</td>
<td>Random Variables, Result Sets and Execution Time with 50 Factor Tables</td>
<td>49</td>
</tr>
<tr>
<td>5.8</td>
<td>Execution time with 30 Factor Tables for with &amp; without optimizations</td>
<td>50</td>
</tr>
<tr>
<td>5.9</td>
<td>Execution time with 50 Factor Tables for with &amp; without optimizations</td>
<td>50</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Query Answering over RDF Knowledge Bases . . . . . . . . . . . 2
1.2 Output of SLD Resolution . . . . . . . . . . . . . . . . . . . . . 3

3.1 Representation for Combined Model . . . . . . . . . . . . . . . . 17
3.2 Query Derivation Structure . . . . . . . . . . . . . . . . . . . . . 18
3.3 Query Lineage DAG with Expanded Base Facts . . . . . . . . . . 20
3.4 Sample Factor Graph . . . . . . . . . . . . . . . . . . . . . . . . 22
3.5 The Proposed Framework . . . . . . . . . . . . . . . . . . . . . . 23
3.6 Sample Independent Query Lineage Model by Tree . . . . . . . . 25
3.7 Sample Dependent Query Lineage Model by DAG . . . . . . . . 27
3.8 Intermediate Factors in the Lineage Structure . . . . . . . . . . 28
3.9 The Factor Table Manipulation Process . . . . . . . . . . . . . . 31
3.10 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

4.1 Boolean Formula Shrinkage (BFS) in Query Derivation Tree . . . 36
4.2 Shared Factors Reuse (SFR) in Query Derivation Tree . . . . . . 37

5.1 Varying Degree of Correlations among Base Facts . . . . . . . . 39
5.2 Comparative Analysis of Runtime with varying % DOC . . . . . 41
5.3 Run Time, Number of Hops, Result Sets, Random Variables with [81-100] % DOC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
5.4 Execution Time vs. Varying Number of Factor Tables . . . . . . 47
5.5 Sampling Time vs. Varying Number of Factor Tables . . . . . . 48
5.6 Query Execution Time vs. RVs, RSS . . . . . . . . . . . . . . . . 48
5.7 Comparative Analysis for Execution Time for 30 FTs . . . . . . 49
5.8 Comparative Analysis for Execution Time for 50 FTs . . . . . . 50
Chapter 1

Introduction

In this section, we present the motivation and problem statement of this work with a brief introduction to uncertain knowledge bases and probabilistic databases. Further, we state our main contributions and outline for the presented work.

1.1 Motivation and Problem Statement

Correlations and dependencies exist in everyone’s life.

For example:
If we know that Al_Gore is married with Tipper_Gore, then we also know that Tipper_Gore is married with Al_Gore and vice versa. These two statements are dependent on each other, as if A is married with B then B is also married with A. This forms a symmetric or bi-implication correlation between known statements.

Another example is:
Let statement 1 be true which states that Anna lives in Frankfurt, then statement 2 which states that Anna studies in Frankfurt, has higher probability for being true. Hence, statement 1 and statement 2 are no more independent. We apply this idea of correlations and dependencies in the URDF reasoning framework. This forms a non-symmetric implication correlation between statement 1 and statement 2.

URDF is a framework for efficient reasoning over uncertain RDF knowledge bases. This is developed at Max Planck Institute for Informatics. But it does not handle correlations or dependencies among base facts.

Let us consider Figure 1.1. It illustrates the URDF query answering procedure over YAGO knowledge base. YAGO is a large RDF knowledge base which consists of 20 million facts extracted from Wikipedia. URDF framework takes
an input query and pre-specified soft rules. The query is grounded using SLD resolution. The URDF reasoner gives the answer against the input query. The final answer against the query is derived from underlying query lineage. The lineage for the specified query evaluation presents the way, how a particular answer is derived on the basis of pre-specified soft rules. The whole process returns the answer for the input query by the assumption that all facts in the knowledge base are independent. This mechanism assumes that all facts are independent. But this is not the reality, information extraction is done using various existing knowledge sources, e.g., Wikipedia. The facts extracted from knowledge source can be dependent on each other.

The goal of our presented work is to provide an expressive URDF framework with dependencies among base facts. The work-flow of existing URDF is composed of base facts, soft rules and hard rules, where all facts are assumed to be independent. The processing of URDF framework is as follows:

2. Grounds input query using SLD resolution and creates dependency graph. Here, dependency graph has all base facts which are required to answer input query.
4. Outputs final answer with corresponding query lineage.
The output of SLD resolution step for input query “Where Does Al Gore Live?” is presented in Figure 1.2. Here, the facts A, B, D are independent base facts. The URDF reasoner performs reasoning tasks on these base facts only. The input query is grounded by the reasoner using specified hard and soft rules:

**Hard rules** form mutually exclusive sets of facts. In this framework, a hard rule of the form bornIn(Tipper Gore, x) can be used to represent all the potential birth places of the entity Tipper Gore. These possible birth places that can be bound to variable x in the grounding phase are mutually exclusive.

**Soft rules** are represented using definite Horn clauses, e.g., rules where exactly one literal is positive. Horn rules with exactly one positive literal can equivalently be written as implications, in which all literals are positive. When written as implication, the body of rule is a conjunction and the head consists of a single literal.

A grounded soft rule over a set of facts F is a set S such that S ⊆ F. Each atomic fact a ∈ C becomes a literal, when each a ∈ C is marked as either positive or negative. For example, a grounded rule can be:

marriedTo(Anna, Deen), livesIn(Anna, Italy) → livesIn(Deen, Italy)

We present details for soft rules, hard rules, reasoner and inference engine processing in Chapter 2.6.
Our proposed model introduces correlations among base facts. It takes a set of soft rules and an input query.

We do not handle hard rules in this thesis. The detailed processing of our designed system is presented in Chapter 3.

1.2 Contributions

Our proposed framework incorporates deductive grounding, query lineage and correlated base facts. The probabilistic inference is applied over both correlated base facts and query lineage. The correlations in base facts are represented using a Markov Random Field which is an undirected representation. The deductive grounding and lineage at query time are represented using a directed representation. However, the final model with correlated base facts is undirected, hence our proposed framework can be represented as a factor graph.

Further, we provide optimization schemes for efficient query answering, e.g., using shared correlations. The detailed discussion for optimizations are provided in Chapter 4. We are exploring the challenges for further optimization in the query processing over uncertain probabilistic knowledge bases. Our main contributions in this thesis are presented as follows:

1. We provide a unified framework which combines correlated base facts, lineage of query derivation, and deductive grounding. Probabilistic inference is applied over correlated base facts and deductive grounding. The proposed framework is a mapping over generalized representation, factor graphs.

2. Our framework applies Markov Chain Monte Carlo sampling technique using Gibbs sampler for probabilistic inference.

3. We present an extensive experimental evaluation over large knowledge base (YAGO) with 20 million facts, correlations among base facts and soft rules.

1.3 Outline

This thesis is organized as follows:

Chapter 2 presents the state-of-the-art for probabilistic databases, probabilistic graphical models, and applications of factor graphs. It provides a brief introduction to deductive databases and Datalog. Chapter 2.6 presents the URDF framework for efficient reasoning in RDF knowledge bases with applied rules.
(soft and hard). The URDF framework employs Weighted MAX-SAT Solver for the particular combination of soft and hard rules in order to resolve inconsistencies. Chapter 3 presents the proposed framework in detail. It elaborates the data representation model, database modeling schema, major modules in the designed framework, proposed model as factor graphs, the query lineage models for independent base facts, and dependent base facts and the sampling scheme. Chapter 4 presents the optimizations performed on the designed framework, it includes the Boolean formula shrinkage and shared factors reuse approaches. We provide experimental evaluation of our implemented framework and a discussion on the results in Chapter 5. Finally, in Chapter 6, we present summary and conclusions with possible future directions.
Chapter 2

Related Work

This chapter presents the role of probabilistic databases and focuses on ULDBs and the Trio system. It discusses major constructs of ULDBs and presents real-world applications of probabilistic databases. Further, deductive databases and Datalog language is discussed. Then, we present a categorization of probabilistic graphical models and Markov Logic Networks. Finally, the URDF framework with soft rules, hard rules, propositional reasoning and inference engine is presented.

2.1 Deductive Databases and Datalog

A deductive database system is a database system which introduces new facts based on specified rules and facts stored in the (deductive) database. This process of introducing new facts is called deduction. Deductive database combines logic programming with relational databases. Deductive database is a merge of conventional databases representing facts, a knowledge base representing rules and inference engine for deriving new facts (inferred or derived facts). The language used for specifying facts, rules and corresponding queries in deductive databases is called Datalog. The query evaluation in Datalog is based on first-order logic. Datalog differentiates between intensional and extensional predicate symbols. Extensional predicate symbols represent facts and intensional predicate symbols represent rules. The following example presents a sample Datalog program:

Example 2.1. The following lines present two facts. These facts state that child of Anna is Dina and child of Dina is Rose.

\[
\begin{align*}
\text{childOf}(\text{Anna}, \text{Dina}). \\
\text{childOf}(\text{Dina}, \text{Rose}).
\end{align*}
\]

The rule in Datalog has two parts separated with :- symbol. The part on left
hand side of this symbol is referred as Head and part on right hand side of this symbol is referred as Body. We define two rules which represent successor relationship.

\[\text{successor}(X, Y) : \neg \text{child}(X, Y).\]
\[\text{successor}(X, Y) : \neg \text{child}(X, Z), \text{successor}(Z, Y).\]

Hence in Example 2.1, first rule can be taken as: \(X\) is successor of \(Y\) if it is known that \(X\) is child of \(Y\); second rule can be taken as: \(X\) is successor of \(Y\) if it is known that \(X\) is child of \(Z\) and \(Z\) is successor of \(Y\). The order of clauses does not matter in Datalog.

Datalog is applied in variety of reasoning applications. Authors in [2] deal with problem of translating schemas from a model to another, in a model independent framework. Translations are specified as Datalog programs. Service-oriented applications facilitate the exchange of business services among participants. A business level conceptual model is presented in [13] which gives primacy to the participants in a service-oriented application. This model provides reasoning using Datalog. Recently, a family of expressive extensions of Datalog, called Datalog+/−, is introduced as a new paradigm for query answering in [10]. Datalog+/− extends plain Datalog by features, e.g., existentially quantified rule heads. It also restricts the rule’s body to achieve decidability and tractability.

### 2.2 Probabilistic Databases

Today, a large number of organizations deal with uncertain data. Uncertain data comes from various sources, e.g., probabilistic data management and analysis, data collection and integration etc. Uncertainty of each tuple in a knowledge base is represented by numeric values which represents a confidence value of the corresponding tuple. A higher value denotes greater “truthness” than a lower value, e.g., John is married with Sonia is assigned confidence value of 0.9 (high confidence value) and John is married with Bob is assigned confidence value of 0.1 (low confidence value).

Traditional databases store solid facts that can be considered certain. In many cases, we do not know things precisely. For example: “I saw a bird, but i am not sure if it was a dove or pigeon.”

Hence, probabilistic databases have been proposed where attributes and tuples may be probabilistic. Probabilistic data bases are databases where the value of some attributes or the existence of some tuples are uncertain and known only with some probability.

A high level classification scheme is as follows:
• Tuple level uncertainty: All attributes in a tuple are known precisely, but existence of the tuple is uncertain. For example, Tuple ("IIT Bombay", ...) will be present in the answer with some uncertainty.

• Attribute level uncertainty: Tuples (identified by keys) exist for sure, however the attribute value is uncertain. For example, the temperature of Frankfurt will be in between 20C and 30C tomorrow.

Applications in many areas such as information extraction, RFID and scientific data management, data cleaning, data integration, and financial risk assessment produce large volumes of uncertain data, which are best modeled and processed by a probabilistic database.

Probabilistic databases encode a large number of possible instances of deterministic databases. Each deterministic instance contains a specific possible world and probabilities of all possible worlds (events which are possible to occur) form a distribution which sum up to 1. Inference procedure over such databases is #P-Complete [12]. Hence, a variety of approximation techniques are proposed. Probabilistic databases present a general uncertainty model in databases.

Probabilistic databases are designed for storage and retrieval of uncertain data. Hence, these databases are uncertain and all possible worlds in probabilistic databases have assigned probabilities of truthness.

2.3 ULDBs and Trio System

ULDBs [5] provide an extension of relational databases with uncertainty and lineage support. ULDBs present the conceptual relationships between uncertainty and lineage. Moreover, it discusses the performance gains achieved by combination of uncertainty and lineage. However, it leads to expensive query evaluation without approximation algorithm. The work presented in [16, 40] addresses various issues of handling uncertain data. Authors in [3, 11, 8, 27] present ULDBs without lineage. The problems of data lineage tracing are discussed in [41, 17, 3]. Further, ULDBs come up with a lineage extension over these databases. The lineage in ULDBs is captured by a Boolean formula which recursively unrolls the logical dependencies from derived tuple to base tuples. This Boolean formula represents lineage graph which is a directed acyclic (DAG). Probabilistic inference over DAGs is #P-Complete [12]. However, a specific class of Markov networks provide polynomial time query evaluation. Here, probabilistic graphical models come into the game. Some system for such models are: MystiQ for approximate matching [8], Trio based on ULDB data model [6], MayBMS as an extension to PostgreSQL [1], and PrDB based upon Markov Random Fields for handling correlations [36].

The major constructs of ULDBs are the following:
Alternatives

Relations in ULDBs are composed of \(x\)-tuples. Each \(x\)-tuple has one or more alternatives. For example, Ahmed visited Berlin, Paris or Luxembourg. Table 2.1 presents the corresponding \(x\)-tuple:

\[
\begin{array}{ccc}
(Visitor, Places) \\
(\text{Ahmed, Berlin}) & (\text{Ahmed, Paris}) & (\text{Ahmed, Luxembourg})
\end{array}
\]

Table 2.1: Sample \(x\)-tuple

May be (\(\sim\)) Annotations

Uncertainty in ULDBs is represented by \(\sim\) symbol. It shows that a tuple may or may not be present. Such tuple is called as maybe \(x\)-tuple. Table 2.2 presents the data with a Maybe \(x\)-tuple.

\[
\begin{array}{ccc}
(Visitor, Places) \\
(\text{Ahmed, Berlin}) & (\text{Ahmed, Paris}) & (\text{Ahmed, Luxembourg}) \\
(\text{Atiqa, France}) & \sim
\end{array}
\]

Table 2.2: Sample maybe \(x\)-tuple

Confidence

Confidence values (numerical form) are attached with each alternative of a tuple. For example, confidence for information that Ahmed visited Berlin is 0.1, confidence for information that Ahmed visited Paris is 0.2, and confidence of information that Ahmed visited Luxembourg is 0.7. These confidence values depict the level of truthness for corresponding piece of information. Table 2.3 depicts tuples with attached confidence values.

\[
\begin{array}{ccc}
(Visitor, Places) \\
(\text{Ahmed, Berlin}) : 0.1 & (\text{Ahmed, Paris}) : 0.2 & (\text{Ahmed, Luxembourg}) : 0.7 \\
(\text{Atiqa, France}): 0.6 & \sim
\end{array}
\]

Table 2.3: Confidence values for sample \(x\)-tuples
Lineage

Lineage shows, how each x-tuple is derived. It represents a Boolean formula for a tuple existing in database, where the Boolean formula consists of all possible derivations of the corresponding tuple. Probability of lineage formula is evaluated given probability distribution of the input variables against each lineage query.

The ULDBs provide the basis for the Trio system developed at Stanford. Trio is a database system which manages data with lineage. TriQL is the underlying query language used for the Trio system. It is SQL based query language with addition of built in functions, predicates for confidence values and lineage handling. TriQL queries are evaluated after translation into SQL commands. Currently, a single Trio metadata catalog keeps records of lineage and uncertainty, e.g., which relations are uncertain and which x-tuples have lineage to which x-tuples.

2.4 Probabilistic Graphical Models

Probabilistic models and computational algorithms are proposed by [44] for genomic analysis. The genetic regularity networks are treated as probabilistic boolean networks in [37]. The knowledge corroboration with logical rules and user feedback [26] lies in application of Bayesian approach. Probabilistic graphical models facilitate us with uniform way of handling uncertainties and correlations in underlying data.

Probabilistic graphical models are categorized as follows:

1. Directed Models: Bayesian Networks

2. Undirected Models: Markov Random Field

Factor Graphs are a generalized representation for Bayesian networks and Markov Random Fields. Many applications using graphical models are proposed, e.g., probabilistic databases [35] [24] [28], information extraction [7] [44] [15] [19], sensor systems [32] [30] [18] [14] [25], computer vision [42], query approximation [31] and selectivity estimation [22]. The neural networks in [45] are utilized for learning conditional probabilities using Bayesian network. Beal et al. [4] present a new framework for modeling and processing multimedia data using probabilistic graphical models. It tracks the object location from data using Bayesian inference. The inference in biological networks observed in [20] is based on probabilistic graphical models.
2.5 Markov Logic Networks

Markov Logic Networks combine first-order logic and probabilistic graphical models [33]. A first-order knowledge base is a set of strict constraints which cannot be violated at all. If some world violates any hard constraint, it has zero probability. The Markov Logic Networks relax hard rules to soft rules. Relaxation does not assign zero probability to any world which violates rule(s), instead makes it less probable. The probabilistic inference problem is a reduction of query processing in probabilistic databases using all possible worlds [33]. The probabilistic databases assume that tuples are independent.

The correlation among tuples are introduced by Amol Deshpande in [35]. These dependencies are captured with notion of factors and random variables. However, it has no deduction capability for query answering.

2.6 URDF

URDF is a framework for efficient query answering over potentially inconsistent and uncertain RDF knowledge bases. This is a top-down query answering framework which takes pre-specified soft rules and hard rules. The reasoner resolves inconsistencies between rules (soft and hard) and the knowledge bases dynamically. The URDF reasoner populates a dependency graph for each input query. This dependency graph consists of those facts which are required for specified query answering. The underlying approximation algorithm is generalized form of Weighted MAX-SAT problem which results in 50% performance gain over state-of-the-art techniques, e.g., MC-SAT, MAP. The experimental evaluation is presented in Chapter 5.

URDF employs rules formulated in first-order logic (FOL) for reasoning. The basic components for the framework are soft rules, hard rules, a grounding phase, and propositional reasoning. A knowledge base is defined as a triple, \( KB = F, C, S \). This triple consists of extensional (base) facts \( F \), soft clauses \( C \) and hard rules \( S \).

Soft Rules

The soft rules are used for inference in URDF in a deductive style. These rules drive new facts or reinforce the strength (confidence) of existing facts. These soft rules form weighted Horn clauses. Each soft rule has real-valued weight which can be negative for MAX-SAT. An example of a non grounded soft rule is:

\[
is\text{MarriedTo}(?Y, ?X) \rightarrow is\text{MarriedTo}(?X, ?Y) : [0.96]
\]

This rules states that if person \( Y \) is married to person \( X \), then person \( X \) is also married to person \( Y \), and the weight for this rule is 0.96. A higher
weight indicates that a fact reached using this rule has higher confidence than one reached through a rule with a smaller weight. The following rule denotes a grounded soft rule if we assign constants to $X, Y$ variables as given below:

$$\text{isMarriedTo}(\text{Anna}, \text{Mathew}) \rightarrow \text{isMarriedTo}(\text{Mathew}, \text{Anna}) : [0.96]$$

The weight $w_i$ of a soft rule $S_i = R \leftarrow X_1 \land \ldots \land X_n$ is estimated using the extensional database (base facts) as follows:

$$w_i(S_i) = \frac{\text{numberOfGroundings}(R \land X_1 \land \ldots \land X_n)}{\text{numberOfGroundings}(X_1 \land \ldots \land X_n)}$$

Here, $\text{numberOfGroundings}(F)$ represents the number of distinct extensional groundings that exist for $F$.

**Hard Rules**

The hard rules define consistency and integrity constraints e.g., mutually exclusive events, functional or inverse-functional properties of facts, implication, bi-implication. A grounded hard rule is a set of facts (competitor set). A non grounded hard rule forms a binary predicate. A non grounded hard rule which states that a person can be married to only one person at most is presented as follows:

$$\text{isMarriedTo}(?X, ??Y)$$

This rule states that each $X$ can be married to only one $Y$. Here, $X, Y$ represents finite strings where $X \in \text{DOM}_a$ and $Y \in \text{DOM}_b$. $\text{DOM}_a$ is a finite number of constants. Each constant starts with ? and consists of letters from [a-z,A-Z]. $\text{DOM}_b$ is a finite number of constants. Each constant starts with ?? and consists of letters from [a-z,A-Z].

**Propositional reasoning**

Truth values are assigned to the facts as a principal form of resolving inconsistencies. URDF framework employs a weighted MAX-SAT Solver for particular combination of soft and hard rules. The worst-case complexity of MAX-SAT algorithm is $O(|C| \cdot |S|)$ where $C$ and $S$ denote number of facts in grounded soft rules and hard rules respectively.

**The Inference Engine**

Let us consider an example query: Where Does Al Gore Live?. Four base facts are encountered by this query as given below.

1. $\text{marriedTo}(\text{Al Gore}, \text{Tipper Gore})$
2. $\text{bornIn}(\text{Al Gore}, \text{Washington DC})$
The URDF inference engine produces the result of the input query according to the specified soft and hard. For this example query, the inference engine encounters the following two rules from the knowledge base.

1. \( \text{bornIn}(x, y), \text{isMarriedTo}(z, x), \text{bornIn}(z, y) \rightarrow \text{livesIn}(x, y) \)

2. \( \text{isMarriedTo}(z, x), \text{livesIn}(z, y) \rightarrow \text{livesIn}(x, y) \)

Each associated predicate in given rule is based on existing facts of knowledge base. Each fact has an associated confidence value of being true or false.

The lineage graph for this example query is shown in Figure 3.2. It is obvious in Figure 3.2 that lineage is a representation of the underlying Boolean formula which is evaluated by the URDF reasoner. We extend this work of the existing URDF framework, and integrate correlated base facts for capturing dependencies. The probabilistic inference is done over all facts including correlated base facts with query grounding and lineage.
Chapter 3

Lineage Enabled Deductive Reasoning with Correlations

3.1 Preliminaries

Our presented framework is an integration of deductive grounding, query lineage and correlated base facts. Correlations are captured using an undirected Markov Random Field, while deductive grounding is directed. Final model can be shown using generalized factor graph approach which is undirected.

We introduce probabilistic graphical models and factor graphs with few basic definitions which form the basis of our framework.

3.1.1 Probabilistic Graphical Models

Probabilistic databases differentiate among logical data modeling, and physical representation, like relational databases. Probabilistic databases have designated probabilities for all possible worlds. These databases make assumptions about underlying data, e.g., tuples are independent. Hence, applications which tend to utilize existing correlations among data cannot rely on these databases. Similarly, probabilistic graphical models (PGM) enforce conditional independence between random variables. These models are based on graphical structure. This graphical structure encodes a complete distribution. Two main classifications for such models are Bayesian networks and Markov networks. Both types encode different set of independencies. Bayesian network forms a directed acyclic graph. While Markov Random Field is similar to a Bayesian network in its representation of dependencies but it is undirected representation. Applications of graphical models include information extraction, computer
vision, manipulation of low-density parity-check codes and modeling in genetic networks.

3.1.2 Factor Graphs

Factor graphs [29] provide a generic representation formalism for graphical models. The roots of factor graphs are from coding theory. These graphs are a modified form of graphical models presented by Wiberg, Loeliger and Koetter. Factor graphs provide an easy descriptive environment for iterative algorithms. These are applied in a variety of real-world applications, e.g., dense stereo reconstruction in spatial signal processing [39], artificial intelligence and machine learning, error-correcting codes [39], as computational framework for biological systems [21], statistical models and probabilistic graphical models (Bayesian networks and Markov Random Fields).

Factor graphs provide a generalized view of different graphical models [29]. Incorporating lineage in relational models is presented in [10, 9]. Traditional graphical models express decomposition of a joint distribution as product of functions over dependent variables. ULDBs [6] provide an extension of relational databases with uncertainty and lineage support. The ULDBs present conceptual relationships between uncertainty and lineage. Moreover, it discusses performance gains achieved by combination of uncertainty and lineage. However, it leads to expensive query evaluation. The work presented in [16, 40] addresses various issues of handling uncertain data. The problems of data lineage tracing is discussed in [41, 17, 3]. The framework proposed in [35] applies probabilistic graphical models for capturing correlations among tuples in a probabilistic database. However, this work does not incorporate the lineage structure of queries. The approach presented in [43] applies factor graphs for representing uncertainty in relational databases without lineage.

Here, we present formal definition for Factor, Factor graphs, and Markov Random Field. A factor is a basic unit of probabilistic graphical models. Each factor is a boolean valued function over input random variables. Hence, probabilistic graphical models capture uncertainty and correlations using these boolean valued functions. Let \( R \) denotes a random variable and \( \text{domain}(R) \) denotes corresponding domain of random variable.

**Definition 3.1.** Factor \( f(R) \) is a function over a set of random variables (base facts) \( R = R_1, ..., R_n \) where \( f(r) \in \{0, 1\}, \forall r \in \text{domain}(R_1) \times ... \times \text{domain}(R_m) \). We denote a set of random variables with \( S(R) \). Each element in set \( S(R) \) has corresponding set of attached factor table(s) from knowledge base. The set of factor tables is denoted by \( F^i_{\text{factor}} \), here \( i \) denotes a random variable for which \( F^i_{\text{factor}} \) is defined.

Factor graphs provide a generic representation mechanism for both directed and un-directed models.

**Definition 3.2.** A factor graph is a bipartite graph that presents the structure
of underlying factorization over a set of random variables. Each factor denotes a potential function denoted as $\phi_k$.

**Definition 3.3.** A Markov Random Field (MRF) is an undirected graphical model also called Markov network. This is a graphical model for joint distribution over a set of random variables $S(R)$. Here, $S(R) = (R_1, R_2, R_3, ..., R_n) \in S(R)$ such that $\forall a, R_a \subseteq S(R)$ (Pearl, 1988). Markov Random Field is composed of two parts, 1: An undirected graph $G$: the vertices of graph $G$ are elements from $S(R)$, 2: A set of potential functions called cliques, denoted by $\phi_k$. The joint probability distribution over a set of random variables $S(R)$ with a set of potential functions $\phi_k$ using Markov Random Field is presented by Equation \[3.1\]

$$P(R = r) = \frac{1}{Z} \prod_c \phi_c(r_c)$$

Here, $Z$ denotes the normalization factor given by Equation \[3.2\]

$$Z = \sum_{r \in R} \prod_c \phi_c(r_c)$$

### 3.2 Our Design

#### 3.2.1 Unified Model: Combine Directed and Undirected

Our designed framework combines directed and undirected approaches. Figure 3.1 presents this idea. The proposed framework is unified as it integrates a deductive grounding phase, a query lineage, and correlated base facts. The deductive grounding phase results in a dependency graph which is directed, and correlated base facts are incorporated using Markov Random Field which is undirected, finally incorporating query lineage on top of deductive grounding and correlated base facts leads to an undirected representation, factor graph approach.
3.2.2 Data and Representation Model

We have a knowledge base $KB$ which consists of RDF base facts $F$ and soft rules $S$. The factor graph is created after deductive grounding phase which represents query lineage DAG. Base facts are called extensional facts while soft rules represent intensional facts. The query lineage DAG is composed of extensional and intensional facts. We define lineage in terms of conjunction and disjunction nodes, where a conjunction node represents base (extensional) facts and disjunction node represents soft rule(s) (intensional) facts.

**Lineage:**

The query lineage DAG is composed of two kinds of nodes: Conjunction Node and Disjunction Node, as presented in Figure 3.2. When a soft rule $S_i$ occurs in lineage, we have disjunction node which immediately follows a conjunction node composed of facts (extensional/intensional) occurring in soft rule $S_i$.  

Figure 3.1: Representation for Combined Model
We can formally define conjunction and disjunction node as follows:

**Conjunction Node**: The conjunction node $C_i$ is a composition of base facts. It is represented as $\land$ notation and combines random variables $R = R_1, R_2, ..., R_n$. Here, $\forall R_i \in R$ are facts which represent base facts occurred in a soft rule $S_i$ or derived facts inferred from soft rules’ tracing.

**Disjunction Node**: The disjunction node represents soft rule $S_i$ and always follows one or more conjunction nodes. We describe both types of nodes (conjunction and disjunction) formally by Equation (3.3) and (3.4) respectively.

$$
C_i = \begin{cases} 
1 & \forall R_j = 1 \\
0 & \text{otherwise} 
\end{cases} 
$$

(3.3)

Where, $i$ represents an identifier of a conjunction node, $j$ represents identifier of a random variable or base/derived fact and $i, j \in 1, 2, 3, ..., n$.

$$
D_i = \begin{cases} 
1 & \exists C_j = 1 \\
0 & \text{otherwise} 
\end{cases} 
$$

(3.4)

Where, $C_j$ is successor node of $D_i$ and represents a conjunction node(s) where $i, j \in 1, 2, 3, ..., n$. The leaf-nodes represent base facts in the lineage structure. The framework picks rule $S_1$ and goes to head fact of $S_1$, then head fact is traversed recursively among other available soft rules. If this head fact is found in body of another soft rule $S_2$, then $S_2$ is processed. The same process continues
until the current head fact does not occur in the body of any other soft rule. Hence, the lineage structure is built using recursive tracing of the soft rules.

Query lineage DAG (factor graph) with logical connectives \( \land \) (conjunction operator), \( \lor \) (disjunction operator) forms a Boolean formula. This Boolean formula is evaluated by URDF with truth assignment to base facts occurring in lineage DAG. Truth values depend upon number of the input base facts which are required to answer a particular query. For example, if a query encounters 10 base facts. Then, total number of combinations are \( 2^{10} = 1024 \), which is equal to all possible worlds. Hence, URDF reasoner evaluates the Boolean formula for all possible worlds. This query lineage formula is evaluated against a given assignment of truth values to base facts which returns 0 or 1 in top level node of lineage DAG, see Figure 3.2 for our example query, “Where Does Al Gore Live?”.

Lineage in URDF comes from query derivation structure with applied soft rules which corresponds to a Boolean formula as previously discussed. Now, we explain recursive processing of input soft rules in our framework. For example, we have a rule \( S_1 \) as follows:

\[
\text{Married}(\text{Tipper Gore}, \text{Al Gore}) \rightarrow \text{Married}(\text{Al Gore}, \text{Tipper Gore})
\]

This lineage DAG gives answers of following questions:

- Which extensional facts are encountered in query derivation tree?
- Which intensional facts are encountered in query derivation tree?
- How extensional and intensional facts are interconnected during input query processing?
- Which level in lineage DAG corresponds to which intensional facts?
- Which level in lineage DAG corresponds to which extensional facts?

Figure 3.2.2 presents the sample lineage DAG with expanded correlated base facts.

**Factor Graphs and Correlations:**

The database schema contains factor tables for each existing base fact. Each base fact has \( 2 \times 2 \) factor table in the knowledge base, which contains two factor values against 0 and 1, factor value against 0 is used when corresponding base fact is false and factor value against 1 is used otherwise. Such binary factor tables are called priors which exist for each base fact in knowledge base. The cardinality of the factor table increases as the number of correlated base facts increases (as explained in Section 3.2.3). URDF retrieves all factor tables against the input base facts and its correlated base facts for query answering.
Sample factor tables for base facts $A, B, C, D$ are presented below as Tables 3.1, 3.2, 3.3, 3.4 in top down sequence.

<table>
<thead>
<tr>
<th></th>
<th>$\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>1</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Table 3.1: Factor Table for $A$

<table>
<thead>
<tr>
<th></th>
<th>$\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>1</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 3.2: Factor Table for $B$

<table>
<thead>
<tr>
<th></th>
<th>$\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 3.3: Factor Table for $C$
Table 3.4: Factor Table for $D$

<table>
<thead>
<tr>
<th>D</th>
<th>$\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>1</td>
<td>0.9</td>
</tr>
</tbody>
</table>

URDF enforces correlations using factor tables which correspond to generalized factor graph approach. We categorize correlations as follows:

- Offline or static correlations.
- Online or dynamic correlations.

We also distinguish between rules. A rule can be either deduction or correlation.

**Deduction Rules:**

Rules which occur in query lineage DAG are known as deduction rules, which enforce correlations among base facts on runtime. Such, correlations are called online or dynamically induced correlations.

**Correlation Rules:**

Correlations can be induced before query time using correlation rules. A correlation rule is a rule which is designed for correlating desired base facts before URDF starts its execution. Such, correlations are called offline or statically induced correlations. Offline correlations are induced by populating factor tables in knowledge base before query execution process initiates.

Let us consider an example for inducing correlation among base facts. The knowledge base in Figure 1.1 has 4 base facts $A, B, C, D$. Sample factor graph representation for given base facts is shown in Figure 3.4. Four grey boxes correspond to factor tables of specified base facts $A, B, C, D$. These factor tables are called priors which exist for each base fact in knowledge base. Further, these base fact can be correlated, and correlations can be induced among several base facts statically before query execution starts or dynamically from query lineage. For example, $B, D$ can be correlated statically by introducing factor table for $B, D$ using following correlation rule:

\[
isMarried(Tipper\_Gore, Al\_Gore) \rightarrow isMarried(Al\_Gore, Tipper\_Gore)
\]

Figure 3.4 shows corresponding factor table. Factor values $\phi$ in this Table are randomly chosen and are normalized by $Z$ factor if necessary, such that all factor values in a factor table sum up to 1.
Deductive Grounding and Probabilistic Inference:

Query driven interface provides capability for providing input queries with specific soft rules. Input query and soft rules are specified and closed world reasoning (what is currently not known to be true, is false) is assumed. Input query is grounded with soft rules using SLD resolution which creates dependency graph. This dependency graph is expanded for each base fact found in dependency graph recursively in depth first manner to retrieve all correlated base facts. Here, dependency graph has all base facts which are required to answer input query. As probabilistic inference is expensive mechanism [12], we perform inference over all base facts and soft rules using full joint distribution when number of random variables are less than 16, otherwise Markov Chain Monte Carlo technique (MCMC) using a Gibbs sampler is applied and the answer is derived with a confidence value. Details for probabilistic inference are given in Section 3.2.6. Major processing phases of our proposed framework are represented in Figure 3.5.
3.2.3 Data Base Schema Modeling

We present our schema for efficiently handling correlations among base facts in knowledgebase. We propose a vertically growing schema. This schema consists of three main tables as follows:

- Facts
- Factor Tables
- Factors

Facts table contains Fact Identifier(ID), Relation(Relation), Argument1(Arg1), Argument2(Arg2) and Confidence level(Confidence). Sample tuples in Facts are presented by Table 3.5.

<table>
<thead>
<tr>
<th>ID</th>
<th>Relation</th>
<th>Arg1</th>
<th>Arg2</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>401568313</td>
<td>isMarriedTo</td>
<td>Al_Gore</td>
<td>Tipper_Gore</td>
<td>0.944</td>
</tr>
<tr>
<td>401559001</td>
<td>isMarriedTo</td>
<td>Tipper_Gore</td>
<td>Al_Gore</td>
<td>0.944</td>
</tr>
<tr>
<td>400192109</td>
<td>bornIn</td>
<td>Al_Gore</td>
<td>Washington_D.C.</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.5: Sample Tuples in the Facts Table

The first tuple demonstrates the fact that *Al Gore acted in An_Inconvenient_Truth* with the confidence of 0.966.

Factor Tables contains Factor Table Identifier (Factor_Table_ID), Fact Identifier (FactID) and Position. Table 3.6 presents sample data in the Factor Tables.
Sample data shows that facts having identifiers 401568313 and 401559001 are correlated. Position column maintains a unique ordering for base facts' storage. This position value is mapped in the first column of table Factors in order to pick the factor value for a given truth assignment of corresponding base facts in Factor_Tables table. The values for correlation factors between 401568313 and 401559001 are shown in Table 3.7 for each possible truth assignment. This table presents sample tuples in Factors Table which contains Factor Identifier (Factor_ID), Factor Table Identifier (Factor_Table_ID) and corresponding Factor value (Factor).

### Table 3.6: Sample Tuples in the Factor_Tables Table

<table>
<thead>
<tr>
<th>Factor_Table_ID</th>
<th>FactID</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>401568313</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>401559001</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 3.7: Sample Tuples in the Factors Table

<table>
<thead>
<tr>
<th>Factor_ID</th>
<th>Factor_Table_ID</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>5</td>
<td>0.4</td>
</tr>
<tr>
<td>01</td>
<td>5</td>
<td>0.1</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>0.1</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>0.4</td>
</tr>
</tbody>
</table>

The data in above tables represent process for capturing correlations between two base facts. If we want to correlate more than two variables, we add three tuples with same factor table identifier for each individual fact identifier in Factor_Tables. Additionally, we add eight tuples with distinct factor identifiers ranging from 0 – 7, a corresponding factor table identifier and factors. Number of columns in this schema is independent of number of correlated RV’s. This schema can be utilized for storing arbitrarily many correlated facts/potential random variables (RV’s) in proposed framework.

### 3.2.4 Probabilistic Inference over Query Lineage Models

Our framework applies probabilistic inference over a dependency graph which is result of deductive grounding phase. The dependency graph consists of all required base facts for query answering. After grounding, we retrieve lineage DAG for input query. This lineage structure is depicted in Figure 3.2 for our example query, “Where Does Al Gore Live?”. Dependency graph is populated with correlated base facts of each base fact occurred in query lineage.

Factor tables are used for capturing dependencies and correlations in base facts. This model induces correlations among base facts by attaching factor
tables with correlated base facts. The URDF framework starts execution and query is grounded with specified soft rules. We present working of independent query lineage model (without correlations among base facts) and dependent query lineage model (with correlations among base facts).

**Independent Query Lineage Model:**

This model is applicable to a tree, as shown in Figure 3.6 where each base fact is independent of each other, e.g., $A, B, C, D$ are independent of each other. Answer derivation tree/query lineage is a Boolean formula over the encountered base facts during reasoning process. Boolean formula for Figure 3.6 is as follows:

$$(C \land B) \lor (A \land D)$$

In Figure 3.2, we see various conjunction ($\lor$) and disjunction ($\land$) nodes. The conjunction node is composed of base facts. The disjunction node is composition of conjunction or disjunction nodes. We introduce factors for each conjunction and disjunction node encountered over the query lineage structure, called as intermediate factors. These intermediate factors correspond to intermediate random variables in underlying factor graph and encode soft rules. This process is represented in Figure 3.8. Notice that, top of answer derivation tree always returns a disjunction node. The populated probability against this node is final resulting probability. Answer derivation tree is a big sum of products as shown
in Equation (3.5).

\[
\mu(\text{result}) = \sum f_{\text{result}}(\text{result}|C_1, C_2) 
\]

\[
f_{C_1}(C_1|B, C)
\]

\[
f_{C_2}(C_2|A, D)
\]

Let us execute example query over answer derivation structure for calculating the resulting probability. Here, we assume that all base facts are independent. In this case, each base fact has two confidence values. The confidence value \( P \) denotes a strength for the fact being true. The confidence value \( (1 - p) \) denotes strength for fact being false. Each conjunction node is true with resulting confidence, if all attached base facts are true with some confidence. If any of base fact is false, the corresponding conjunction node results in false. The resulting confidence value of each conjunction node for being true or false is calculated according to Equation (3.7).

\[
P(A \land B) = P(A) \times P(B) \quad (3.7)
\]

Each disjunction node is evaluated according to Equation (3.8).

\[
P(A \lor B) = [1 - (1 - P(A))(1 - P(B))] \quad (3.8)
\]

\( P(A), P(B), (1 - P(A)), (1 - P(B)) \) represents the confidences for base fact \( A \) being true, the confidence for base fact \( B \) being true, the confidence for base fact \( A \) being false, the confidence for base fact \( B \) being false respectively. Assume that, \( P(A) = 0.7, P(B) = 0.9, P(C) = 0.8, P(D) = 0.9 \) are confidences for the \( A, B, C, D \) base facts respectively. The converse probabilities for \( A, B, C, D \) are \( 0.3, 0.1, 0.2, 0.1 \), respectively. Now, we process answer derivation tree from bottom to top. First conjunction node from left side is composed of \( B, C, D \) base facts. This node returns 1 when all input facts to this node are true. The resulting confidence value of this node being true is calculated as follows:

\[
P(B) \times P(C) \times P(D) = 0.9 \times 0.8 \times 0.9 = 0.648
\]

This result is input of conjunction node at second level in derivation tree. Hence, we get \( 0.648 \times P(A) = 0.454 \). We encounter first disjunction node at third level, which is input of two conjunction nodes. We calculate resulting confidence for this node as follows:

\[
[1 - (1 - 0.454)(1 - 0.454)] = 0.702
\]

The disjunction node on top is based on two input nodes (one is the conjunction of \( B, C, D \) and the other is the resulting value 0.702 calculated above). The final resulting value on top level disjunction node is calculated as below:

\[
P(D_1) = [1 - (1 - 0.298)(1 - 0.352)] = 0.896
\]

The independent model cannot capture correlations among base facts. Our designed model eliminates this deficiency.
Dependent Query Lineage Model:

This model is applicable to a directed acyclic graph (DAG), as shown in Figure 3.7 where base fact \( A \) is dependent.

\[
\begin{align*}
&f_{a_1} \\
&\lor \\
&f_{c_2} \\
&\land \\
&C \\
&\land \\
&A \\
&\land \\
&D
\end{align*}
\]

Figure 3.7: Sample Dependent Query Lineage Model by DAG

Boolean formula for Figure 3.7 demonstrates this dependency, which is as follows:

\[(C \land A) \lor (A \land D)\]

We provide a uniform framework for handling independent query lineage model and dependent query lineage model. The URDF framework starts execution and query is grounded with specified soft rules. We get a dependency graph which consists of all required base facts for query answering. This graph is expanded by traversing all correlated base facts for each base fact in the existing dependency graph and the dependency graph is populated until no new base facts are found. However, URDF processing using factor tables is explained in Section 3.2.6.

3.2.5 Intermediate Factors

The independent model cannot capture the correlations among base facts. We propose a unified framework which combines the independent base facts model with correlations. Each base fact is a random variable in underlying lineage DAG, categorized as a base fact random variable. This framework is based upon intermediate factors with corresponding factor tables. These intermediate factors are characterized as intermediate random variables which capture
soft rules. Figure 3.8 represents base fact random variables \((A, B, C, D)\) and intermediate random variables listed below:

\[(C_1, C_2, C_3, C_4, C_5, D_1, f_{result})\]

Intermediate factors are Boolean valued variables. As, Figure 3.8 demonstrates that intermediate random variable \(C_1\) is based upon three base fact random variables \(B, C, D\). Result of \(f_{C_1}\) depends upon truth values of input base facts. The resulting is 1 with 0.7 probability if all input base facts \(B, C, D\) are true, otherwise it is 0 with 0.3 probability. This can be represented by Equation (3.9).

\[
f_{C_1}(C_1|B, C, D) = \begin{cases} 
0.7 & C_1 \leftrightarrow B = 1 \land C = 1 \land D = 1 \\
0.3 & \text{otherwise}
\end{cases} \tag{3.9}
\]

Figure 3.8: Intermediate Factors in the Lineage Structure

Resulting value of 1 indicates existence of intermediate factor \(f_{C_1}\), which represents that corresponding soft rule is true. Similarly, further processing in lineage structure generates the intermediate factors \(C_2, C_3, C_4, C_5, D_1\). Finally, resulting value exists in factor \(f_{\text{result}}\). Intermediate factors \((C_1, C_2, C_3, C_4, C_5, D_1, f_{\text{result}})\) are represented with Equations (3.10), (3.11), (3.12), (3.13), (3.14), (3.15), respectively.

\[
f_{C_2}(C_2|C_1) = \begin{cases} 
0.8 & C_2 \leftrightarrow C_1 = 1 \land A = 1 \\
0.2 & \text{otherwise}
\end{cases} \tag{3.10}
\]

Resulting value of \(f_{C_2}\) depends upon the existence of two random variables \(f_{C_1}\) (intermediate factor) and \(A\) (base fact). Resulting value is 1 with 0.8 probability if both dependent variables exist, otherwise 0 with 0.2 probability is assigned.
to intermediate factor \( f_{C_2} \).

\[
f_{C_3}(C_3|A, B, C) = \begin{cases} 
0.9 & C_3 \leftrightarrow A=1 \land B=1 \land C=1 \\
0.1 & \text{otherwise} 
\end{cases}
\] (3.11)

Similarly proceeding, output from intermediate factor \( f_{C_3} \) depends upon the existence of three random variables \( A, B, C \) (base facts). Intermediate factor \( f_{C_3} \) is 1 with 0.9 probability if all base facts exist, otherwise 0 with 0.1 probability is assigned to this intermediate factor.

\[
f_{C_4}(C_4|C_3, D) = \begin{cases} 
0.7 & C_4 \leftrightarrow C_3=1 \land D=1 \\
0.3 & \text{otherwise} 
\end{cases}
\] (3.12)

Intermediate factor \( f_{C_4} \) depends upon the existence of intermediate factor \( f_{C_3} \) and base fact \( D \). If both random variables exist, \( f_{C_4} \) results in 1 with 0.7 probability, otherwise 0 with 0.3 probability is output of intermediate factor \( f_{C_4} \).

\[
f_{C_5}(C_5, B, C, D) = \begin{cases} 
0.9 & C_5 \leftrightarrow B=1 \land C=1 \land D=1 \\
0.1 & \text{otherwise} 
\end{cases}
\] (3.13)

The value of the intermediate random variable \( f_{C_5} \) depends upon the existence of three base facts \( B, C, D \). The existence of all these base facts results in 1 with 0.9 probability produced by factor \( f_{C_5} \), otherwise 0 with 0.1 probability is returned.

\[
f_{D_1}(D_1|C_2, C_4) = \begin{cases} 
0.6 & D_1 \leftrightarrow C_2=1 \lor C_4=1 \\
0.4 & \text{otherwise} 
\end{cases}
\] (3.14)

The value of the intermediate factor \( f_{D_1} \) relies on result of two intermediate factors \( f_{C_2} \) and \( f_{C_4} \). If any of these two intermediate factors exist, resulting value from \( f_{D_1} \) is 1 with 0.6 probability, otherwise 0 with 0.4 probability.

\[
f_{\text{result}}(\text{result}, C_5, D_1) = \begin{cases} 
0.7 & \text{result} \leftrightarrow C_5=1 \lor D_1=1 \\
0.3 & \text{otherwise} 
\end{cases}
\] (3.15)

Finally, result factor \( f_{\text{result}} \) produces 1 with 0.7 probability, if any one of its dependent intermediate factors \( f_{C_5} \) or \( f_{D_1} \) exist, else 0 with 0.3 probability is assigned to \( f_{\text{result}} \).

\[
\mu(\text{result}) = \sum f_{\text{result}}(\text{result}|C_5, D_1) \bigg/ \sum f_{\text{result}}(\text{result}|C_5, D_1) + f_{C_5}(C_5|B, C, D) + f_{D_1}(D_1|C_2, C_4) + f_{C_4}(C_4|C_3, D) + f_{C_2}(C_2|C_1, A) + f_{C_5}(C_5|A, B, C) + f_{C_1}(C_1|B, C, D)
\]
$C_1$ forms a conjunction node whose value depends upon base fact random variables $B, C, D$. Hence, the base fact random variables $B, C, D$ are required to be true for the intermediate random variable $C_1$ to be true.

We have one table named as **Facts** which stores a fact identifier, fact description and fact confidence for each fact in knowledge base. We add two more tables (Factor_Tables and Factors). The **Factor_Tables** stores factor table identifiers and fact identifiers. The data in Factor_Tables looks as given below:

<table>
<thead>
<tr>
<th>Factor Table ID</th>
<th>Fact ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
</tr>
</tbody>
</table>

Table 3.8: Sample Data in Factor Table

<table>
<thead>
<tr>
<th>Factor ID</th>
<th>Factor Table ID</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.7</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0.1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0.9</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0.2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0.8</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3.9: Sample Data in Factors

### 3.2.6 Correlated Base Facts

In Table 3.8 Factor Table ID 1, 2, 3 show that their corresponding facts are independent. Factor Table ID 4 represents that facts $A, D$ are dependent, hence $A$ and $D$ are correlated. Corresponding factors for correlated facts are stored in Factors table. The **Factors** table stores factor identifiers, factor table identifiers and factors. The data inFactors table is presented in Table 3.9.

Let us execute the example input query, “Where Does Al Gore Live?” using correlations among input base facts. We have four base facts during sample query evaluation. As, input query encounters four base facts. We retrieve all correlated factor tables for each base fact. Retrieval of factor tables is a crucial step to understand as the overall confidence computation for the query answer.
depends upon this. We encounter four base facts for this query. We retrieve factor tables for each base fact one by one. For extracting factor tables against $A$, we find correlation identifiers and factor table identifiers from \textit{Factor Table}. This is depicted by Table 3.6. Then, we retrieve the corresponding factors from Table 3.7 for each previously extracted factor table identifier. This returns a list of tables filled with factor identifiers and factors corresponding to a specified factor table identifier. If a base fact is not correlated with any other base fact, we get single factor table with two factors (which contains factors for truth values of 1 and 0).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{factor_table_manipulation_process.png}
\caption{The Factor Table Manipulation Process}
\end{figure}

The factor tables for base facts $A, B, C, D$ are represented in Figure 3.9. We introduce intermediate factors for conjunction and disjunction nodes as shown in Figure 3.8. These factors represent soft rules occurred in query lineage. Table 3.10 presents joint distribution over input base facts. Our Gibbs sampler assigns truth values to all base facts and intermediate factors which occur in query lineage and Boolean formula is evaluated. Final resulting node named as $f_{\text{result}}$ helps in identifying worlds (rows) against which the Boolean formula evaluates to true. Each row forms a Boolean formula which is evaluated
with corresponding truth values of input base facts \((A, B, C, D)\) and intermediate factors \((C_1, C_2, C_3, C_4, C_5)\). Rows against which we get zero in last node \(f_{\text{result}}\) are filtered out. Here, Equation (3.5) shows big multiplication of all random variables in underlying network. This corresponds to a truth table with \(2^9\) possible worlds for exact inference as presented in Table 3.10. We apply Gibbs sampler for sampling over all possible worlds as truth table grows exponentially with number of base facts and soft rules.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>C_1</th>
<th>C_2</th>
<th>C_3</th>
<th>C_4</th>
<th>C_5</th>
<th>D_1</th>
<th>Result</th>
<th>(\phi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.009</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>.041</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>.032</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>.044</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.094</td>
</tr>
</tbody>
</table>

Table 3.10: Truth Table With Full Joint Distribution

Here, major problem is that, number of possible words is exponential in the number of encountered base facts during query evaluation process. So, we evaluate queries with a limited number of base facts using all possible worlds manipulation. This problem is now a probabilistic inference issue as queries involving a large number of base facts need sampling process. We apply MCMC sampling using Gibbs sampler. The following section presents the details for probabilistic inference.

**Probabilistic Inference using Gibbs Sampler:**

Inference process in probabilistic graphical models is expensive [12]. It is \#P-complete in Markov networks (Roth, 1996). The manipulation in probabilistic graphical models using the possible world semantic becomes impossible, when dealing with large amount of data. MCMC methods provide widely applicable inference approximation schemes. These methods facilitate with a class of computational algorithms based upon repeated sampling. These methods are a specialized class of algorithms which are used for sampling from a given probability distributions. Input probability distribution is based on underlying Markov chain model and state of chain after \(N\) number of steps is used as a sample from desired distribution. Here, \(N\) denotes a positive integer which is fully dependent on the conditions where Markov chain is used. Crucial issue is to determine value of \(N\), which indicates number of times to sample over input probability distribution. The value of \(N\) depends on convergence criteria. The sampling process stops when the convergence criteria is met.

We implement Gibbs sampling for our correlated base facts model. We customize Gibbs sampling as follows:

- Convergence criteria using Geweke test [23].
Evaluating query lineage/Boolean formula for assigned truth values of input base facts and soft rules.

The complete execution process of our framework is represented by Figure 3.10.

Figure 3.10:

The algorithm presents the working of our Gibbs sampler. Our algorithm samples over truth values of all base facts involved in query lineage DAG. All correlated factor tables are attached to each base fact before passing the base facts as input to the Gibbs sampler. Intermediate random variables in the lineage DAG represent soft rules. Gibbs sampler utilizes intermediate random variables distinguished as intermediate factors for filtering out invalid states (where the Boolean formula evaluates to false). Each valid state corresponds to a row with non-zero resulting factor in random variable \( f_{\text{result}} \). The idea is to sum up over all rows where this resulting factor \( f_{\text{result}} \) exists. Hence, we customized Gibbs sampler with a Boolean evaluation function for intermediate factors/random variables. This can be done by traversing the lineage structure of answer derivation recursively and checking whether the overall Boolean formula evaluates to true. Factors to be multiplied are selected for each such sampled state. These factors are chosen from input base facts using given truth assignment of the corresponding sampled state.

This algorithm returns an array of samples against which \( f_{\text{result}} \) evaluates to true. Sampled rows are retained by using sample_array. We sum up the factors over all retained rows which is \( \sum (RR) \). Then, we sum up the factors over all sampled rows which is \( \sum (SR) \). In final step, we take the ratio of both results as shown by Equation (3.16).

\[
P(S) = \frac{\sum (RR)}{\sum (SR)}
\]  

(3.16)
**Algorithm 1** Gibbs Sampler

**Require:** $n$ Base Facts, Query Lineage Structure/Boolean Formula

$i \leftarrow 1$

$j \leftarrow 0$

while Convergence met OR $i \equiv 100000$ do

Generate sample $S$ according to $P(S|\text{neighbors}(S))$ in between total possible worlds

Generated sample $S$ is converted to binary number with equal number of bits as number of base facts

Query lineage structure is traversed to check whether we get 1 in result factor $f_{\text{result}}$. This step corresponds to boolean evaluation of input query formula.

if Boolean evaluation returns 1 then

$j \leftarrow j + 1$

end if

$i \leftarrow i + 1$

end while

Return $j/i$

**Convergence Criteria: Geweke Test**

Geweke proposed a method for Markov chain convergence in 1992 \[23\]. The test statistic is a standard Z-score. The Geweke test partitions the samples in two portions. The means of two partitions is approximately the same if chain is at its stationary status. The calculation of Z-score depends upon assumption that two partitions of Markov chain are asymptotically independent. We partition the samples in two parts with 10% samples in one partition and 40% samples in second partition. We set threshold value/error bound to be 5% which means that difference between two means does not exceed this error bound.

**Answer Derivation Procedure: An Example**

URDF query answering with correlated base facts is presented by algorithm \[2\]. We elaborate the process of calculating the confidence of a query answer with the help of an example query, “Where Does Al Gore Live?”. As confidence value calculation for this query depends upon four base facts, mentioned in Section \[2.6\]. The steps of answer derivation process are carried out in sequence by starting with soft rules’ grounding. The dependency graph is created and expanded with correlated base facts. This is composed of random variables for each base fact $A, B, C, D$, intermediate random variables for (conjunction, disjunction) nodes, and the final random variable $f_{\text{result}}$.

The total number of intermediate random variables depends upon the
Algorithm 2 URDF Execution with Correlated Base Facts

Require: Input Query $Q$, Soft Rules $S$

Ground $Q$ using SLD resolution and create dependency graph
Each base fact in dependency graph is expanded to get all of its correlated base facts
Retrieve factor tables for each base fact from knowledge base
Introduce intermediate factors for each soft rule in query lineage
Get one row by Gibbs sampler, for truth assignment of base facts in the query lineage DAG
For all rows where Boolean formula evaluates to 1, we sum over all stored multiplications and marginalize it
Final result after marginalization returns a confidence value which is confidence value for answer of input query

query lineage DAG, as depicted in Figure 3.2. Factor tables are retrieved from the knowledge base using the tables: Factor_Table and Factors.
Chapter 4

Optimizations

This chapter presents optimizations performed over the extended URDF framework. We perform two optimization schemes given as follows:

- Boolean formula shrinkage
- Shared factors reuse

4.1 Boolean Formula Shrinkage

The answer derivation for a particular query results in potentially large Boolean form composed of conjunctions and disjunctions. Subsumption removal algorithms are NP-hard. We perform various optimizations over this framework.

Figure 4.1: Boolean Formula Shrinkage (BFS) in Query Derivation Tree
This step (BFS) eliminates duplication among variables. The variable elimination step results in more efficient results over URDF queries. Figure 4.1 depicts such duplicated variables, e.g., $A$ and $B$. The conjunction of variables $A, B, C$ gets 1 when these variables get true value by truth assignment. The conjunction on top of it represents again conjunction with $A$. If $A$ is false, we have no need to evaluate each part of lineage DAG where $A$ exists in the conjunction, as final resulting truth value for conjunction node leads always to 0. The same is the case for variable $B$. We remove such variables which are repeated in the underlying conjunctions of lineage DAG.

4.2 Shared Factors Reuse

Figure 4.2: Shared Factors Reuse (SFR) in Query Derivation Tree

The second step (SFR) finds shared factors in the derivation tree and eliminates these factors [35]. Hence, results of shared factors are utilized throughout the lineage structure of query. Figure 4.2 highlights shared factors. Further step can be to skip the sub parts in tree like repeated conjunction groups (each conjunction is composed of same input base facts ultimately having no effect on final result). The part of lineage DAG is reused for reducing processing time for the input query.

Execution time of query reduces when BFS and SFR optimization schemes are applied. Detailed results for improved runtime of input queries are presented in Section 5.3.
Chapter 5

Experimental Evaluation and Results

5.1 Experimental Settings

This chapter presents the experimental results for our designed framework. We start with the experimental settings and present evaluation results with necessary discussion.

5.1.1 System Specification

The system specification for our conducted experiments are presented as follows:

1. 3.2 GB Memory
2. DELL Optiplex 760 PC using Windows Operating System
3. Intel Pentium Processor E5200

Each experiment is conducted 5 times and the average time for each query is recorded. The underlying development framework is Eclipse using Java 1.6.0 JDK.

5.1.2 Data Sets

We used data sets from YAGO database [38]. YAGO (Yet Another Great Ontology) is an ontology. The data in YAGO is automatically extracted from Wikipedia and WordNet. It consists of approximately 20 million facts. The accuracy of YAGO is 95%, which is explicitly (manually) derived. We utilized real as well as synthetic data sets for our experiments. We explain the process of creating synthetic datasets in Section 5.2.2.
Let us introduce a metric for inducing correlations called Degree of Correlations (DOC).

**Definition 5.1.** Degree of Correlations (DOC) is defined for each input query $Q_i$. This is based upon total number of base facts encountered in query lineage DAG $T_i$, which can be correlated to each other. For Example: $Q_1$ encounters 4 base facts: $A, B, C, D$. If we define correlations between $A, B$ using some correlation rule in database, this corresponds to 50% of 4 base facts. Hence, Percent DOC will be 50%. Now, we add another rule, which correlates $A, B, C$ as well. This is $3/4$ of all base facts. Hence, it represents the 75% of DOC among base facts. Similarly, we keep on adding more correlations, which keeps on increasing percent DOC among base facts. Finally, we can perform another correlation step (by adding another correlation rule), which correlates $A, B, C, D$ base facts. This ultimately, leads towards 100% DOC among base facts. The process of varying degree of correlation among base facts $A, B, C, D$ is depicted by Figure 5.1.

We specify three different ranges for percent DOC and compare runtime for different input queries. The ranges are as given below:

- 33-40% DOC
- 54-67% DOC
- 81-100% DOC

**5.1.3 Experimental Categorization**

We categorize experimental evaluation as follows:

1. We measure the runtime for different queries with varying degree of correlations on real and synthetic data sets.
2. We analyze the effect of varying degree of correlations on runtime.
3. We present the resulting number of hops encountered with varying degree of correlations.

4. We calculate number of hops and number of RV’s with varying degree of correlations.

5. We analyze grounding time, processing time and total execution time for various queries with varying degree of correlations.

6. We analyze the effect of the number of base facts and the number of result sets with varying degree of correlations on execution time.

5.2 Performance Evaluation for Real vs. Synthetic Data Sets

Our performance evaluation results are based on real as well as synthetic data sets. This section focuses on experimental results and discussion for both.

5.2.1 Performance Evaluation for Real Data Sets

We introduced the correlations for each query against each specified % DOC range and measured runtime. We order input queries with minimum to maximum runtime taken [using 33-40 % DOC].

Query 1: actedIn(?X, Total Recall, 1);
           bornIn(?X, Thal, Austria, 1)

Query 2: isMarriedTo(Woody Allen, ?X, 1);

Query 3: hasWonPrize(?X, Nobel Prize in Physics, 1);
           bornIn(?X, Ulm, 1);
           bornOnDate(?X, ?Z, 1);
           yearBefore(?Z, 1900, 1);
           type(?X, wikicategory Patent examiners, 1);

Query 4: bornIn(?X, Oxford, 1);
           graduatedFrom(?X, ?Y, 1);
           hasAcademicAdvisor(?X, ?Z, 1);
           graduatedFrom(?Z, University of Cambridge, 1)

Query 5: actedIn(Arnold Schwarzenegger, ?X, 1);
           actedIn(?Y, ?X, 1);
           bornIn(?Y, ?Z, 1);
           notEquals(Arnold Schwarzenegger, ?Y, 1)

Query 6: livesIn(Al Gore, ?X, 1)
Figure 5.2: Comparative Analysis of Runtime with varying % DOC

Query 7: directed(Martin_Scorsese, ?MOVIE, 1);
actedIn(?GUY1, ?MOVIE, 1);
actedIn(?GUY2, ?MOVIE, 1);
notEquals(?GUY1, ?GUY2, 1);
notEquals(?GUY1, Martin_Scorsese, 1);
notEquals(?GUY2, Martin_Scorsese, 1)

Query 8: isMarriedTo(Emma_Thompson, ?SPOUSE, 1);
actedIn(Emma_Thompson, ?MOVIE, 1);
actedIn(?SPOUSE, ?MOVIE, 1)

Table 5.1 presents query runtime in seconds for queries $Q_1$ to $Q_8$ with varying % DOC and corresponding chart is depicted by Figure 5.2

<table>
<thead>
<tr>
<th>QueryNumber</th>
<th>31 – 40% DOC</th>
<th>54 – 67% DOC</th>
<th>81 – 100% DOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1$</td>
<td>0.50</td>
<td>0.67</td>
<td>0.76</td>
</tr>
<tr>
<td>$Q_2$</td>
<td>0.57</td>
<td>0.64</td>
<td>1.10</td>
</tr>
<tr>
<td>$Q_3$</td>
<td>0.55</td>
<td>0.64</td>
<td>0.76</td>
</tr>
<tr>
<td>$Q_4$</td>
<td>1.40</td>
<td>1.70</td>
<td>1.95</td>
</tr>
<tr>
<td>$Q_5$</td>
<td>1.07</td>
<td>1.30</td>
<td>1.60</td>
</tr>
<tr>
<td>$Q_6$</td>
<td>1.00</td>
<td>1.60</td>
<td>2.5</td>
</tr>
<tr>
<td>$Q_7$</td>
<td>4.45</td>
<td>5.23</td>
<td>8.62</td>
</tr>
<tr>
<td>$Q_8$</td>
<td>2.95</td>
<td>3.04</td>
<td>3.50</td>
</tr>
</tbody>
</table>

Table 5.1: Comparative Analysis of Runtime for Different Queries with Varying % DOC

Our framework expands each base fact to other correlated base facts (treated as one hop). This scenario is depicted in Section 3.2.2 Figure 3.2.2
This is a major impact factor for calculating runtime of query. We start from base fact \( D \), and start picking correlated factor tables of \( D \), as \( D \) is correlated with \( E \). We pick binary factor tables for \( DE \). Further, \( E \) is correlated with \( F \) and \( G \), \( F \) is correlated with \( K \), and \( G \) is correlated with \( H \). This search is expanded in a depth-first manner and all correlated factor tables are picked. This process returns us factor tables for \( DE, EF, EG, FK \) and \( GH \) including the priors of base facts \( D, E, F, G, K, H \). The number of hops traversed for base fact \( D \) is 3.

The total number of random variables on which sampling is performed includes the base fact set and the non-grounded soft rules. Hence, the count of random variables directly influences the runtime of each query. Moreover, each query can involve more than one result set. The number of result sets also influence runtime of query. We analyze effect for number of hops, number of result sets and number of random variables (base facts and non-grounded soft rules). Table 5.2 presents experimental data for number of hops, result sets and corresponding random variables for each specified input query (\( Q_1 \) – \( Q_8 \)). The corresponding charts are presented by Figure 5.3.

<table>
<thead>
<tr>
<th>Query Number</th>
<th>81 – 100% DOC</th>
<th>Hops Count</th>
<th>Result Sets</th>
<th>Random Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q_1 )</td>
<td>2.50</td>
<td>42</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>( Q_2 )</td>
<td>1.10</td>
<td>20</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>( Q_3 )</td>
<td>0.76</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( Q_4 )</td>
<td>1.60</td>
<td>18</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>( Q_5 )</td>
<td>1.95</td>
<td>15</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>( Q_6 )</td>
<td>0.76</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( Q_7 )</td>
<td>3.50</td>
<td>48</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>( Q_8 )</td>
<td>8.62</td>
<td>104</td>
<td>22</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 5.2: RunTime, Number of Hops, Result Sets, Random Variables with [81-100] \% DOC

5.2.2 Performance Evaluation for Synthetic Data Sets

We evaluate different queries: \( Q_1, Q_2, ..., Q_7 \) with a varying number of factor tables, but here factor tables are created using synthetic expansion. Factor tables are created and manipulated in memory at runtime and the results are listed. Each query follows two rounds of execution (with a varying number of factor tables). In the 1st round of execution, input query creates 30 factor tables for each base fact encountered in specified query. The second execution round constitutes of manipulating 50 factor tables for each base fact during query evaluation.

Suppose, some query \( Q_c \) has 4 base fact \( F \) in its lineage, then population of 30 factor tables for this fact in underlying query is done by picking 30 facts
Figure 5.3: RunTime, Number of Hops, Result Sets, Random Variables with [81-100] % DOC

from YAGO knowledge base randomly and correlating each base fact from randomly picked 30 base facts with base fact $F$ in the query lineage. This results in 30 binary factor tables which denote 30 binary correlations ($F$ is correlated with each randomly picked base fact). Query inference loads these populated factor tables from memory on runtime.

The queries $Q_1$ to $Q_7$ for synthetic datasets are presented as follows:

**Query 1**: $\text{hasWonPrize}(\texttt{?X}, \texttt{Nobel\_Prize\_in\_Physics})$;
$\text{bornIn}(\texttt{?X}, \texttt{Ulm})$;

Base Facts Encountered: 2

Each fact has 30 factor tables in the 1st round, and 50 factor tables in the 2nd round.

Hence,

1. **Total Factor Tables in 1st round are**: $2 \times 30 = 30$ Factor Tables
2. **Total Factor Tables in 2nd round are**: $2 \times 50 = 50$ Factor Tables

**Query 2**: $\text{actedIn}(\texttt{?X}, \texttt{Total\_Recall})$;

Base Facts Encountered: 1

Each fact has 30 factor tables in the 1st round, and 50 factor tables in the 2nd round.
Hence,

1. **Total Factor Tables in 1st round are: (1 * 30 = 30 Factor Tables)**
2. **Total Factor Tables in 2nd round are: (1 * 50 = 50 Factor Tables)**

**Query 3:** \( actedIn(Arnold\_Schwarzenegger,?X) \);

Base Facts Encountered: 1

Each fact has 30 factor tables in the 1st round, and 50 factor tables in the 2nd round.
Hence,

1. **Total Factor Tables in 1st round are: (1 * 30 = 30 Factor Tables)**
2. **Total Factor Tables in 2nd round are: (1 * 50 = 50 Factor Tables)**

**Query 4:** \( livesIn(Al\_Gore,?X) \);

Base Facts Encountered: 12

Each fact has 30 factor tables in the 1st round, and 50 factor tables in the 2nd round.
Hence,

1. **Total Factor Tables in 1st round are: (12 * 30 = 360 Factor Tables)**
2. **Total Factor Tables in 2nd round are: (12 * 50 = 600 Factor Tables)**

**Query 5:** \( bornIn(?X,Oxford)\);
\( hasAcademicAdvisor(?X,?Z)\);
\( graduatedFrom(?Z,University\_of\_Cambridge)\);

Base Facts Encountered: 3

Each fact has 30 factor tables in the 1st round, and 50 factor tables in the 2nd round.
Hence,

1. **Total Factor Tables in 1st round are: (3 * 30 = 90 Factor Tables)**
2. **Total Factor Tables in 2nd round are: (3 * 50 = 150 Factor Tables)**
Query 6: `isMarriedTo(Woody_Allen, ?X)`

Base Facts Encountered: 4

Each fact has 30 factor tables in the 1st round, and 50 factor tables in the 2nd round.
Hence,

1. **Total Factor Tables in 1st round are: (4 * 30 = 120 Factor Tables)**
2. **Total Factor Tables in 2nd round are: (4 * 50 = 200 Factor Tables)**

Query 7: `directed(Martin_Scorsese, ?MOVIE); actedIn(?GUY1, ?MOVIE); notEquals(?GUY1, Martin_Scorsese);`

Base Facts Encountered: 3

Each fact has 30 factor tables in the 1st round, and 50 factor tables in the 2nd round.
Hence,

1. **Total Factor Tables in 1st round are: (3 * 30 = 90 Factor Tables)**
2. **Total Factor Tables in 2nd round are: (3 * 50 = 150 Factor Tables)**

We set the number of factor tables to 30 for each individual base fact occurred in input query. This number is updated to 50 factor tables (FTs) for each base fact and query is evaluated again. For example, query 7 encounters 3 base facts. Each base fact is correlated with 30 factor tables. Hence, in the first round, total binary factor tables are (3 multiplied by 30) 90 for evaluating this query. In the second round, this count of factor tables becomes 150 (3 multiplied by 50). The query is evaluated and the execution time is noted for both rounds. Tables 5.3 and 5.4 present grounding time, sampling time, data load time and total execution time for each query using, synthetic expansion.
Table 5.3: Grounding Time, Processing Time, Execution Time for 30 FTs

<table>
<thead>
<tr>
<th>Query</th>
<th>Grounding Time (sec)</th>
<th>Processing Time (sec)</th>
<th>Data Load from DB (sec)</th>
<th>Total Execution Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0.406</td>
<td>28.1</td>
<td>0.062</td>
<td>28.568</td>
</tr>
<tr>
<td>Q2</td>
<td>0.016</td>
<td>51.46</td>
<td>0.047</td>
<td>51.523</td>
</tr>
<tr>
<td>Q3</td>
<td>0.015</td>
<td>51.84</td>
<td>0.078</td>
<td>51.933</td>
</tr>
<tr>
<td>Q4</td>
<td>0.157</td>
<td>60.7</td>
<td>0.094</td>
<td>60.951</td>
</tr>
<tr>
<td>Q5</td>
<td>1.266</td>
<td>69.5</td>
<td>0.062</td>
<td>70.828</td>
</tr>
<tr>
<td>Q6</td>
<td>0.093</td>
<td>89.5</td>
<td>0.047</td>
<td>89.64</td>
</tr>
<tr>
<td>Q7</td>
<td>0.125</td>
<td>909.218</td>
<td>0.047</td>
<td>909.39</td>
</tr>
</tbody>
</table>

Table 5.4: Grounding Time, Processing Time, Execution Time for 50 FTs

<table>
<thead>
<tr>
<th>Query</th>
<th>Grounding Time (sec)</th>
<th>Processing Time (sec)</th>
<th>Data Load from DB (sec)</th>
<th>Total Execution Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0.391</td>
<td>54.81</td>
<td>0.078</td>
<td>55.279</td>
</tr>
<tr>
<td>Q2</td>
<td>0.016</td>
<td>110</td>
<td>0.047</td>
<td>110.063</td>
</tr>
<tr>
<td>Q3</td>
<td>0.015</td>
<td>105.9</td>
<td>0.063</td>
<td>105.978</td>
</tr>
<tr>
<td>Q4</td>
<td>0.188</td>
<td>89.3</td>
<td>0.047</td>
<td>89.535</td>
</tr>
<tr>
<td>Q5</td>
<td>0.219</td>
<td>2941</td>
<td>0.062</td>
<td>2941.281</td>
</tr>
<tr>
<td>Q6</td>
<td>0.078</td>
<td>171.9</td>
<td>0.047</td>
<td>172.025</td>
</tr>
<tr>
<td>Q7</td>
<td>0.157</td>
<td>1973.7</td>
<td>0.047</td>
<td>1973.904</td>
</tr>
</tbody>
</table>

Table 5.5 summarizes comparative execution runtime for different queries with varying number of factor tables.

<table>
<thead>
<tr>
<th>Query</th>
<th>30 Factor Tables</th>
<th>50 Factor Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>28.56 sec</td>
<td>55.27 sec</td>
</tr>
<tr>
<td>Q2</td>
<td>51.52 sec</td>
<td>110.06 sec</td>
</tr>
<tr>
<td>Q3</td>
<td>51.93 sec</td>
<td>105.97 sec</td>
</tr>
<tr>
<td>Q4</td>
<td>60.95 sec</td>
<td>89.53 sec</td>
</tr>
<tr>
<td>Q5</td>
<td>70.82 sec</td>
<td>2941.28 sec</td>
</tr>
<tr>
<td>Q6</td>
<td>89.64 sec</td>
<td>172.02 sec</td>
</tr>
<tr>
<td>Q7</td>
<td>909.39 sec</td>
<td>1973.90 sec</td>
</tr>
</tbody>
</table>

Table 5.5: Execution Time vs. Varying Number of Factor Tables

This is also represented by Figure 5.4. As, $Q_5$ and $Q_7$ takes more time compared with other queries. We omit these queries in the graphs for simplicity and clarification.

Figure 5.5 presents the chart for a varying number of factor tables vs. sampling time against the data shown in Table 5.6. Sampling time is on the
vertical axis, while the horizontal axis depicts the input query with the total number of base facts encountered and the number of result sets.

<table>
<thead>
<tr>
<th>Query</th>
<th>Base Facts of Query</th>
<th>Number of Result Sets</th>
<th>Sampling Time (sec)</th>
<th>RVs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q₁</td>
<td>2</td>
<td>1</td>
<td>54.81</td>
<td>52</td>
</tr>
<tr>
<td>Q₂</td>
<td>1</td>
<td>2</td>
<td>110</td>
<td>51</td>
</tr>
<tr>
<td>Q₃</td>
<td>1</td>
<td>2</td>
<td>105.9</td>
<td>51</td>
</tr>
<tr>
<td>Q₄</td>
<td>12</td>
<td>1</td>
<td>89.3</td>
<td>62</td>
</tr>
<tr>
<td>Q₅</td>
<td>3</td>
<td>72</td>
<td>2941</td>
<td>53</td>
</tr>
<tr>
<td>Q₆</td>
<td>4</td>
<td>3</td>
<td>171.9</td>
<td>54</td>
</tr>
<tr>
<td>Q₇</td>
<td>3</td>
<td>29</td>
<td>1973.7</td>
<td>53</td>
</tr>
</tbody>
</table>

Table 5.6: Base Facts, Result Sets, Sampling Time vs. varying number of Factor Tables

Figure 5.6 presents the chart for execution time vs. RVs and RSs for 50 factor tables against data shown in Table 5.7. The data for execution time is on vertical axis, while horizontal axis depicts the query with total number of random variables (RVs) encountered and number of result sets (RSs) for 50 factor tables.
Figure 5.5: Sampling Time vs. Varying Number of Factor Tables

Figure 5.6: Query Execution Time vs. RVs, RSs
Figure 5.7: Comparative Analysis for Execution Time for 30 FTs

<table>
<thead>
<tr>
<th>Query</th>
<th>Random Variables</th>
<th>Number of Result Sets</th>
<th>Execution Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>52</td>
<td>1</td>
<td>55.27</td>
</tr>
<tr>
<td>Q2</td>
<td>51</td>
<td>2</td>
<td>110.06</td>
</tr>
<tr>
<td>Q3</td>
<td>51</td>
<td>2</td>
<td>105.97</td>
</tr>
<tr>
<td>Q4</td>
<td>62</td>
<td>1</td>
<td>89.53</td>
</tr>
<tr>
<td>Q5</td>
<td>53</td>
<td>72</td>
<td>2941.28</td>
</tr>
<tr>
<td>Q6</td>
<td>54</td>
<td>3</td>
<td>172.02</td>
</tr>
<tr>
<td>Q7</td>
<td>53</td>
<td>29</td>
<td>1973.9</td>
</tr>
</tbody>
</table>

Table 5.7: Random Variables, Result Sets and Execution Time with 50 Factor Tables

5.3 Performance Results with Optimizations

We have discussed optimization schemes in Section 4. This section presents the execution time for different queries, given in Section 5.2.2. The datasets for optimization schemes are created synthetically. Tables 5.8 and 5.9 present experimental data for execution time (with and without optimizations) for 30 FTs and 50K FTs respectively.

Figures 5.7 and 5.8 present execution time taken by different queries as described earlier. Each query is executed more efficiently after variable elimination and shared factors reuse. Corresponding results are presented in Tables 5.8 and 5.9 with varying degree of correlations.
<table>
<thead>
<tr>
<th>Query</th>
<th>Total Execution Time (seconds) without Opt</th>
<th>Total Execution Time (seconds) with Opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q₁</td>
<td>21.23</td>
<td>28.56</td>
</tr>
<tr>
<td>Q₂</td>
<td>47.11</td>
<td>51.53</td>
</tr>
<tr>
<td>Q₃</td>
<td>42.31</td>
<td>51.93</td>
</tr>
<tr>
<td>Q₄</td>
<td>32.37</td>
<td>60.95</td>
</tr>
<tr>
<td>Q₅</td>
<td>70.82</td>
<td>46.23</td>
</tr>
<tr>
<td>Q₆</td>
<td>64.01</td>
<td>89.64</td>
</tr>
<tr>
<td>Q₇</td>
<td>909.39</td>
<td>718.67</td>
</tr>
</tbody>
</table>

Table 5.8: Execution time with 30 Factor Tables for with & without optimizations

<table>
<thead>
<tr>
<th>Query</th>
<th>Total Execution Time (seconds) without Opt</th>
<th>Total Execution Time (seconds) with Opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q₁</td>
<td>48.53</td>
<td>55.27</td>
</tr>
<tr>
<td>Q₂</td>
<td>76.12</td>
<td>110</td>
</tr>
<tr>
<td>Q₃</td>
<td>86.19</td>
<td>105.97</td>
</tr>
<tr>
<td>Q₄</td>
<td>78.02</td>
<td>89.53</td>
</tr>
<tr>
<td>Q₅</td>
<td>2941.28</td>
<td>1903.10</td>
</tr>
<tr>
<td>Q₆</td>
<td>123.41</td>
<td>172.02</td>
</tr>
<tr>
<td>Q₇</td>
<td>1973.90</td>
<td>1873</td>
</tr>
</tbody>
</table>

Table 5.9: Execution time with 50 Factor Tables for with & without optimizations
Chapter 6

Conclusion and Future Work

6.1 Summary

Our proposed framework is a combination of correlated base facts with deductive grounding phase and a probabilistic inference phase. We present lineage enabled query answering over uncertain RDF knowledge bases which incorporates correlations and dependencies among base facts. Our system incorporates correlations among base facts statically before query execution starts by explicit correlation rules or dynamically on query time from rule based structure of query answer. Our design presents a unified approach which combines undirected and directed model. We handle correlations using undirected model, e.g., Markov Random Field whereas deductive grounding is directed. Final model is an undirected representation, Factor Graph.

As, inference in probabilistic graphical models is #P-Complete, we use exact inference whenever tractable and otherwise Gibbs sampler samples over all possible worlds. We also perform optimizations: shared factors reuse and boolean formula shrinkage and conduct experiments on real data sets with synthetic expansion of correlated base facts.

6.2 Future Work

Our framework handles soft rules and do not incorporate hard rules. The idea is to skip the worlds where any hard rule is violated. Another challenge is to integrate module for learning based on specified rules which can result in higher confidence for query answer. Using results of pre executed queries for increasing efficiency in query answering is also an open question. Moreover, how can we manage the updates, e.g., an update in rules or facts is made during query
execution. The hall is also open for further optimizations which can improve the efficiency of the system.
Appendix A

Experimental Data

It presents process of collecting experimental data for each input query. It shows, how to change \%DOC for our used input queries. For each input query, we specify number of base facts and derived facts encountered in query lineage with varying \%DOC. **Query 1**: livesIn(Al_Gore,?X,1)

**Total Facts (Base and Derived):** 10 FACTS

**Total Base Facts Encountered:**
1. hasChild(Al_Gore,Kareena_Gore_Schiff)
2. hasChild(Tipper_Gore,Kareena_Gore_Schiff)
3. isMarried(Al_Gore,Tipper_Gore)
4. isMarried(Tipper_Gore,Al_Gore)
5. bornIn(Al_Gore,Washington_DC)
6. bornIn(Tipper_Gore,Washington_DC)

**Degree of Correlations (33.3\%):** 2/6

**Soft Rules Used for Inducing Correlations:**
- isMarriedTo(?Y,?X,1) → isMarriedTo(?X,?Y,1) : [.6]

**Degree of Correlations (66.6\%):** 4/6

**Soft Rules Used for Inducing Correlations:**
- isMarriedTo(?Y,?X,1) → isMarriedTo(?X,?Y,1) : [.6]
- isMarriedTo(?X,?Y,1),
  bornIn(?Y,?Z,1) → bornIn(?X,?Z,1); [0.100169779286927]
Degree of Correlations (99.9%): 6/6

Soft Rules Used for Inducing Correlations:

- \textit{isMarriedTo}(?, ?, X, 1) \rightarrow \textit{isMarriedTo}(?, ?, Y, 1) : [0.6]
- \textit{isMarriedTo}(?, ?, Y, 1),
  \textit{bornIn}(?, ?, Z, 1) \rightarrow \textit{bornIn}(?, ?, X, 1); [0.100169779286927]
- \textit{hasChild}(?, ?, Z, 1),
  \textit{hasChild}(?, ?, Y, 1),
  \textit{notEquals}(?, ?, X, 1) \rightarrow \textit{isMarriedTo}(?, ?, X, 1); [0.3775322283609576]

Query 2: \textit{isMarriedTo(Woody_Allen, ?X, 1)};

Total Facts (Base and Derived): 9 FACTS

Total Base Facts Encountered:

1. \textit{actedIn(Hugh_Grant, Small_Time_Crooks)}
2. \textit{isMarriedTo(Woody_Allen, Soon − Yi_Previn)}
3. \textit{actedIn(Woody_Allen, Small_Time_Crooks)}
4. \textit{actedIn(Sharon_Stone, Picking_Up_the_Pieces)}
5. \textit{actedIn(Woody_Allen, Picking_Up_the_Pieces)}

Degree of Correlations (40.0%): 2/5

Soft Rules Used for Inducing Correlations:

- \textit{isMarriedTo}(?, ?, X, 1) \rightarrow \textit{isMarriedTo}(?, ?, Y, 1) : [0.6]

Degree of Correlations (60.0%): 3/5

Soft Rules Used for Inducing Correlations:

- \textit{isMarriedTo}(?, ?, X, 1) \rightarrow \textit{isMarriedTo}(?, ?, Y, 1) : [0.6]
- \textit{actedIn}(?, ?, Y, 1),
  \textit{actedIn}(?, ?, Z, 1),
  \textit{notEquals}(?, ?, X, 1) \rightarrow \textit{isMarriedTo}(?, ?, X, 1); [1.0]

Degree of Correlations (100%): 5/5

Soft Rules Used for Inducing Correlations:

- \textit{isMarriedTo}(?, ?, X, 1) \rightarrow \textit{isMarriedTo}(?, ?, Y, 1) : [0.6]
• actedIn(?X, ?Y, 1),
  actedIn(?Z, ?Y, 1),
  notEquals(?X, ?Z, 1) → isMarriedTo(?X, ?Z, 1); [1.0]

• isMarriedTo(?X, ?Y, 1),
  actedIn(?Z, ?Y, 1),
  notEquals(?X, ?Z, 1) → actedIn(?X, ?Z, 1); [1.0]

Query 3: actedIn(?X, Total_Recall, 1);
  bornIn(?X, Thal, _Austria, 1)

Total Facts (Base and Derived): 18 FACTS

Total Base Facts Encountered:

1. actedIn(Arnold_Schwarzenegger, Total_Recall)
2. bornIn(Maria_Shriver, Chicago)
3. isMarriedTo(Arnold_Schwarzenegger, Maria_Shriver)
4. isMarriedTo(Maria_Shriver, Arnold_Schwarzenegger)
5. actedIn(Ronny_Cox, Total_Recall)
6. actedIn(Arnold_Schwarzenegger, Predator_(franchise))
7. actedIn(Kevin_Peter_Hall, Predator_(franchise))
8. bornIn(Arnold_Schwarzenegger, Thal, _Austria)

Degree of Correlations (37.5%): 3/8

Soft Rules Used for Inducing Correlations:

• actedIn(?X, ?Y, 1),
  actedIn(?Z, ?Y, 1),
  notEquals(?X, ?Z, 1) → isMarriedTo(?X, ?Z, 1); [1.0]

Degree of Correlations (62.5%): 5/8

Soft Rules Used for Inducing Correlations:

• actedIn(?X, ?Y, 1),
  actedIn(?Z, ?Y, 1),
  notEquals(?X, ?Z, 1) → isMarriedTo(?X, ?Z, 1); [1.0]

• isMarriedTo(?Y, ?X, 1) → isMarriedTo(?X, ?Y, 1); [.6]

• isMarriedTo(?X, ?Y, 1),
  actedIn(?Z, ?Y, 1),
  notEquals(?X, ?Z, 1) → actedIn(?X, ?Z, 1); [1.0]
Degree of Correlations (100%): 8/8

Soft Rules Used for Inducing Correlations:

• actedIn(?X, ?Y, 1),
  actedIn(?Z, ?Y, 1),
  notEquals(?X, ?Z, 1) → isMarriedTo(?X, ?Z, 1); [1.0]

• isMarriedTo(?Y, ?X, 1) → isMarriedTo(?X, ?Y, 1); [.6]

• isMarriedTo(?X, ?Y, 1),
  actedIn(?Z, ?Y, 1),
  notEquals(?X, ?Z, 1) → actedIn(?X, ?Z, 1); [1.0]

• isMarriedTo(?X, ?Y, 1),
  bornIn(?Y, ?Z, 1) → bornIn(?X, ?Z, 1); [0.100169779286927]

Query 4: actedIn(Arnold_Schwarzenegger, ?X, 1);
  actedIn(Y, ?X, 1);
  bornIn(?Y, ?Z, 1);
  notEquals(Arnold_Schwarzenegger, ?Y, 1)

Total Facts (Base and Derived): 23 FACTS

Total Base Facts Encountered:
1. actedIn(Arnold_Schwarzenegger, Total_Recall)
2. bornIn(Maria_Shriver, Chicago)
3. isMarriedTo(Arnold_Schwarzenegger, Maria_Shriver)
4. isMarriedTo(Maria_Shriver, Arnold_Schwarzenegger)
5. actedIn(Ronny_Cox, Total_Recall)
6. isMarriedTo(Kevin_Peter_Hall, Alaina_Reed_Hall)
7. actedIn(Arnold_Schwarzenegger, Predator_(franchise))
8. actedIn(Kevin_Peter_Hall, Predator_(franchise))
9. bornIn(Arnold_Schwarzenegger, Thal, Austria)

Degree of Correlations (33.3%): 3/9

Soft Rules Used for Inducing Correlations:

• actedIn(?X, ?Y, 1),
  actedIn(?Z, ?Y, 1),
  notEquals(?X, ?Z, 1) → isMarriedTo(?X, ?Z, 1); [1.0]
Degree of Correlations (66.6%): 5/9

Soft Rules Used for Inducing Correlations:

- $\text{actedIn}(?X, ?Y, 1), \text{actedIn}(?Z, ?Y, 1), \text{notEquals}(?X, ?Z, 1) \rightarrow \text{isMarriedTo}(?X, ?Z, 1); [1.0]
- $\text{isMarriedTo}(?Y, ?X, 1) \rightarrow \text{isMarriedTo}(?X, ?Y, 1); [0.6]

Degree of Correlations (100%): 9/9

Soft Rules Used for Inducing Correlations:

- $\text{actedIn}(?X, ?Y, 1), \text{actedIn}(?Z, ?Y, 1), \text{notEquals}(?X, ?Z, 1) \rightarrow \text{isMarriedTo}(?X, ?Z, 1); [1.0]
- $\text{isMarriedTo}(?Y, ?X, 1) \rightarrow \text{isMarriedTo}(?X, ?Y, 1); [0.6]
- $\text{isMarriedTo}(?X, ?Y, 1), \text{bornIn}(?Y, ?Z, 1) \rightarrow \text{bornIn}(?X, ?Z, 1); [0.100169779286927]

Query 5: $\text{bornIn}(?X, \text{Oxford}, 1); \text{graduatedFrom}(?X, ?Y, 1); \text{hasAcademicAdvisor}(?X, ?Z, 1); \text{graduatedFrom}(?Z, \text{University of Cambridge}, 1)$

Total Facts (Base and Derived): 12 FACTS

Total Base Facts Encountered:

1. $\text{graduatedFrom}(\text{John Kendrew, University of Cambridge})$
2. $\text{graduatedFrom}(\text{Francis Crick, University of Cambridge})$
3. $\text{hasAcademicAdvisor}(\text{Stephen Hawking, Dennis William Sciama})$
4. $\text{graduatedFrom}(\text{Max Perutz, University of Cambridge})$
5. $\text{bornIn}(\text{John Kendrew, Oxford})$
6. $\text{bornIn}(\text{Stephen Hawking, Oxford})$
7. $\text{hasAcademicAdvisor}(\text{Francis Crick, Max Perutz})$
8. $\text{graduatedFrom}(\text{Dennis William Sciama, University of Cambridge})$
9. $\text{hasAcademicAdvisor}(\text{John Kendrew, Max Perutz})$
10. $\text{hasAcademicAdvisor}(\text{James D. Watson, Max Perutz})$
11. \texttt{graduatedFrom(Stephen	extunderscore Hawking,University	extunderscore of	extunderscore Cambridge)}

\textbf{Degree of Correlations (36.36%): 4/11}

\textit{Soft Rules Used for Inducing Correlations:}

- \texttt{hasAcademicAdvisor(?X,?Y,1), hasAcademicAdvisor(?Z,?Y,1), graduatedFrom(?X,?A,1) \rightarrow graduatedFrom(?Z,?A,1); [0.5032863849765258]}

Single grounding of this rule is applied for 36.36\% DOC. \textbf{Degree of Correlations (54.54\%): 6/11}

\textit{Soft Rules Used for Inducing Correlations:}

- \texttt{hasAcademicAdvisor(?X,?Y,1), hasAcademicAdvisor(?Z,?Y,1), graduatedFrom(?X,?A,1) \rightarrow graduatedFrom(?Z,?A,1); [0.5032863849765258]}

Two groundings of this rule are applied for 54.54\% DOC. \textbf{Degree of Correlations (81.81\%): 9/11}

\textit{Soft Rules Used for Inducing Correlations:}

- \texttt{hasAcademicAdvisor(?X,?Y,1), hasAcademicAdvisor(?Z,?Y,1), graduatedFrom(?X,?A,1) \rightarrow graduatedFrom(?Z,?A,1); [0.5032863849765258]}

Four groundings of this rule are applied for 81\% DOC.

\textbf{Query 6: hasWonPrize(?X,Nobel\_prize\_in\_physics,1); bornIn(?X,Ulm,1); bornOnDate(?X,?Z,1); yearBefore(?Z,1900,1); type(?X,wikicategory\_patent\_examiners,1);}

\textbf{Total Facts (Base and Derived): 16 FACTS}

\textbf{Total Base Facts Encountered:}

1. hasChild(Albert\_Einstein,Eduard\_Einstein)
2. isMarriedTo(Mileva\_Mari,Albert\_Einstein)
3. bornIn(Albert\_Einstein,Ulm)
4. type(Albert\_Einstein,wikicategory\_Patent\_examiners)
5. yearBefore(1879 – 03 – 14,1900)
6. hasChild(Mileva\_Mari,Hans\_Albert\_Einstein)
7. bornIn(Mileva_Mari, Titel)
8. hasChild(Mileva_Mari, Eduard_Einstein)
9. hasWonPrize(Albert_Einstein, Nobel_Prize_in_Physics)
10. hasChild(Albert_Einstein, Hans_Albert_Einstein)
11. bornOnDate(Albert_Einstein, 1879 – 03 – 14)

Degree of Correlations (36.36%): 4/11

Soft Rules Used for Inducing Correlations:
- isMarriedTo(?Y, ?X, 1) → isMarriedTo(?X, ?Y, 1) : [.6]
- hasChild(?X, ?Z, 1),
  hasChild(?Y, ?Z, 1),
  notEquals(?X, ?Y, 1) → isMarriedTo(?X, ?Y, 1) : [0.3775322283609576]

Degree of Correlations (54.54%): 6/11

Soft Rules Used for Inducing Correlations:
- isMarriedTo(?Y, ?X, 1) → isMarriedTo(?X, ?Y, 1) : [.6]
- hasChild(?X, ?Z, 1),
  hasChild(?Y, ?Z, 1),
  notEquals(?X, ?Y, 1) → isMarriedTo(?X, ?Y, 1) : [0.3775322283609576]
- Static correlation between:
  bornIn(Albert_Einstein, Ulm)
  type(Albert_Einstein, wikicategory_Patent_examiners)

Degree of Correlations (81.81%): 9/11

Soft Rules Used for Inducing Correlations:
- isMarriedTo(?Y, ?X, 1) → isMarriedTo(?X, ?Y, 1) : [.6]
- hasChild(?X, ?Z, 1),
  hasChild(?Y, ?Z, 1),
  notEquals(?X, ?Y, 1) → isMarriedTo(?X, ?Y, 1) : [0.3775322283609576]
- Static correlation between:
  bornIn(Albert_Einstein, Ulm)
  type(Albert_Einstein, wikicategory_Patent_examiners)
- Static correlation between:
  bornOnDate(Albert_Einstein, 1879 – 03 – 14)
  hasWonPrize(Albert_Einstein, Nobel_Prize_in_Physics)
• Static correlation between:
  hasChild(Albert_Einstein, Eduard_Einstein)
  hasChild(Albert_Einstein, Hans_Albert_Einstein)

Query 7: isMarriedTo(Emma_Thompson, ?SPOUSE, 1);
actedIn(Emma_Thompson, ?MOVIE, 1);
actedIn(?SPOUSE, ?MOVIE, 1);

Total Facts (Base and Derived): 31 FACTS

Total Base Facts Encountered:
1. actedIn(Jonathan_Pryce, Carrington_(film))
2. isMarriedTo(Emma_Thompson, Kenneth_Branagh)
3. actedIn(Emma_Thompson, Last_Chance_Harvey)
4. actedIn(Emma_Thompson, Fortunes_of_War_(tv_series))
5. actedIn(Kenneth_Branagh, Fortunes_of_War_(tv_series))
6. actedIn(Emma_Thompson, Imagining_Argentina_(film))
7. actedIn(Alan_Bennett, Fortunes_of_War_(tv_series))
8. actedIn(Kathy_Baker, Last_Chance_Harvey)
9. actedIn(Robert_Stephens, Fortunes_of_War_(tv_series))
10. actedIn(Antonio_Banderas, Imagining_Argentina_(film))
11. actedIn(Emma_Thompson, Carrington_(film))
12. actedIn(Rupert_Graves, Fortunes_of_War_(tv_series))
13. actedIn(Jam0065s_Brolin, Last_Chance_Harvey)
14. actedIn(Ronald_Pickup, Fortunes_of_War_(tv_series))

Degree of Correlations (36.36%): 4/11

Soft Rules Used for Inducing Correlations:
• isMarriedTo(?Y, ?X, 1) → isMarriedTo(?X, ?Y, 1) : [.6]

• actedIn(?X, ?Y, 1),
  actedIn(?Z, ?Y, 1),
  notEquals(?X, ?Z, 1) → isMarriedTo(?X, ?Z, 1); [1.0]

Degree of Correlations (54.54%): 6/11

Soft Rules Used for Inducing Correlations:
• isMarriedTo(?Y, ?X, 1) → isMarriedTo(?X, ?Y, 1) : [.6]

• actedIn(?X, ?Y, 1),
  actedIn(?Z, ?Y, 1),
  notEquals(?X, ?Z, 1) → isMarriedTo(?X, ?Z, 1); [1.0]

• Static correlation between:
  actedIn(Emma_Thompson, Imagining_Argentino_(film))
  actedIn(Emma_Thompson, Last_Chance_Harvey)

Degree of Correlations (81.81%): 9/11

Soft Rules Used for Inducing Correlations:
• isMarriedTo(?Y, ?X, 1) → isMarriedTo(?X, ?Y, 1) : [.6]

• actedIn(?X, ?Y, 1),
  actedIn(?Z, ?Y, 1),
  notEquals(?X, ?Z, 1) → isMarriedTo(?X, ?Z, 1); [1.0]

• Static correlation between:
  actedIn(Emma_Thompson, Imagining_Argentino_(film))
  actedIn(Emma_Thompson, Last_Chance_Harvey)

• Static correlation among:
  actedIn(Rupert_Graves, Fortunes_of_War_(tv_series))
  actedIn(Kenneth_Branagh, Fortunes_of_War_(tv_series))
  actedIn(Alan_Bennett, Fortunes_of_War_(tv_series))

Query 8: directed(Martin_Scorsese, ?MOVIE, 1);
actedIn(?GUY1, ?MOVIE, 1);
actedIn(?GUY2, ?MOVIE, 1);
notEquals(?GUY1, ?GUY2, 1);
notEquals(?GUY1, Martin_Scorsese, 1);
notEquals(?GUY2, Martin_Scorsese, 1)

Total Facts (Base and Derived): 38 FACTS

Total Base Facts Encountered:
1. actedIn(Keb′_Mo′, The_Blues_(film))
2. actedIn(Alec_Baldwin, The_Departed)
3. actedIn(Common_(rapper), The_Blues_(film))
4. actedIn(Leonardo_DiCaprio, The_Departed)
5. directed(Martin_Scorsese, The_Departed)
6. actedIn(Ike_Turner, The_Blues_(film))
7. `actedIn(Ali_Farka_Tour, The_Blues_(film))`
8. `directed(Martin_Scorsese, The_Blues_(film))`
9. `actedIn(Pinetop_Perkins, The_Blues_(film))`

**Degree of Correlations (33.3%): 3/9**

*Soft Rules Used for Inducing Correlations:* There is not soft rule in the query lineage, hence we add static correlations among following base facts:

- `actedIn(Pinetop_Perkins, The_Blues_(film))`
- `actedIn(Alec_Baldwin, The_Departed)`
- `actedIn(Common_(rapper), The_Blues_(film))`

**Degree of Correlations (66.6%): 6/9**

*Soft Rules Used for Inducing Correlations:* There is not soft rule in the query lineage, hence we add static correlations among following base facts:

- `actedIn(Pinetop_Perkins, The_Blues_(film))`
- `actedIn(Alec_Baldwin, The_Departed)`
- `actedIn(Common_(rapper), The_Blues_(film))`
- `actedIn(Leonardo_DiCaprio, The_Departed)`
- `directed(Martin_Scorsese, The_Departed)`
- `actedIn(Ike_Turner, The_Blues_(film))`

**Degree of Correlations (100%): 9/9**

*Soft Rules Used for Inducing Correlations:* There is not soft rule in the query lineage, hence we add static correlations among following base facts:

- `actedIn(Pinetop_Perkins, The_Blues_(film))`
- `actedIn(Alec_Baldwin, The_Departed)`
- `actedIn(Common_(rapper), The_Blues_(film))`
- `actedIn(Leonardo_DiCaprio, The_Departed)`
- `directed(Martin_Scorsese, The_Departed)`
- `actedIn(Ike_Turner, The_Blues_(film))`
- `actedIn(Ali_Farka_Tour, The_Blues_(film))`
- `directed(Martin_Scorsese, The_Blues_(film))`
- `actedIn(Pinetop_Perkins, The_Blues_(film))`
Bibliography


