URDF: Efficient Reasoning in Uncertain RDF Knowledge Bases with Soft and Hard Rules

Martin Theobald
Mauro Sozio
Fabian Suchanek
Ndapandula Nakashole

MPI–I–2010–5-002 February 2010

Authors' Addresses

Martin Theobald
Max-Planck-Institut für Informatik
Campus E 1.4
D-66123 Saarbrücken

Mauro Sozio
Max-Planck-Institut für Informatik
Campus E 1.4
D-66123 Saarbrücken

Fabian Suchanek
Max-Planck-Institut für Informatik
Campus E 1.4
D-66123 Saarbrücken

Ndapandula Nakashole
Max-Planck-Institut für Informatik
Campus E 1.4
D-66123 Saarbrücken

Abstract

We present URDF, an efficient reasoning framework for graph-based, non-schematic RDF knowledge bases and SPARQL-like queries. URDF augments first-order reasoning by a combination of *soft rules*, with Datalog-style recursive implications, and *hard rules*, in the shape of mutually exclusive sets of facts. It incorporates the common possible worlds semantics with independent base facts as it is prevalent in most probabilistic database approaches, but also supports semantically more expressive, probabilistic first-order representations such as Markov Logic Networks.

As knowledge extraction on the Web often is an iterative (and inherently noisy) process, URDF explicitly targets the resolution of *inconsistencies* between the underlying RDF base facts and the inference rules. Core of our approach is a novel and efficient approximation algorithm for a generalized version of the Weighted MAX-SAT problem, allowing us to dynamically resolve such inconsistencies directly at query processing time. Our MAX-SAT algorithm has a worst-case running time of $O(|\mathcal{C}|\cdot|\mathcal{S}|)$, where $|\mathcal{C}|$ and $|\mathcal{S}|$ denote the number of facts in grounded soft and hard rules, respectively, and it comes with tight approximation guarantees with respect to the shape of the rules and the distribution of confidences of facts they contain. Experiments over various benchmark settings confirm a high robustness and significantly improved runtime of our reasoning framework in comparison to state-of-the-art techniques for MCMC sampling such as MAP inference and MC-SAT.

Keywords

Reasoning under uncertainty, MAX-SAT with soft and hard rules, SLD resolution.

Contents

1 Introduction

1.1 Incorrectness, Incompleteness and Inconsistency in Information Extraction

Open-domain information extraction on the Web unavoidably leads to a certain amount of *incorrect*, *incomplete*, and even *inconsistent* results. As for the first, even despite the great progress in this research area in recent years, information extraction still often yields *incorrect* results, and it is highly unlikely that even the best extraction tools will ever achieve perfect precision and recall. As a possible remedy, various state-of-the art extraction tools provide confidence values along with the facts they deliver. For example, assume that the user asks for the birth place of *Albert Einstein*, and the knowledge base contains two contradictory facts *bornIn(Albert_Einstein, Ulm)*$_{[0.7]}$ and *bornIn(Albert_Einstein, Munich)*$_{[0.8]}$. Without any other context (or formal rule) that relates these facts to other facts in the knowledge base, we need to rely on the attached confidence values as the only hint on correctness when answering the query. Extraction tools however are mostly limited to operating on a "local" view of the data, typically a single document or even a single sentence, such that the reliability of these confidences remains rather limited. In a more "global" view on the available data, we could relate the extracted facts also to their larger context (e.g., the place where Albert's parents lived when he was born), which typically is a strength of a knowledge base but also requires significantly more sophisticated forms of reasoning which cannot be done at extraction time alone.

Second, information extraction on the Web inherently is *incomplete*. Consider, for example, the query: "*Where does Al Gore live?*" A knowledge base built on general-purpose Web corpora might not directly know a fact telling us where *Al Gore* actually lives. However, if we have a (first-order) rule expressing that "*married couples usually live at the same place*", and we happened to know that *Al Gore*'s wife, *Tipper Gore*, lives in *Washington, D.C.*, we might be able to infer—again with some degree of confidence—that also *Al Gore* lives in *Washington,*

D.C.

Third, and maybe most strikingly, such a knowledge base, consisting of either hand-crafted or inductively learned rules and millions of facts extracted from the Web, is highly likely to also run into numerous *inconsistencies*. Consider, for example, the next query: "*Who is Woody Allen married to?*" Having a fairly noisy rule saying that "*people who co-acted together in the same movie might also be married*" (even when having a very low confidence) would provide a major amount of false answers as candidate spouses of *Woody Allen*.

In summary, reasoning over Web data inherently is *uncertain*, and viable approaches for query answering and reasoning over this kind of data need to be robust with respect to noise in both the extracted facts and inference rules they consider as input.

1.2 URDF – Efficient Reasoning in Uncertain RDF Knowledge Bases

Recent projects such as DBpedia [7], YAGO [32], Intelligence in Wikipedia [34] and KnowItAll [14] have successfully used information extraction techniques to build large semantic knowledge bases from unstructured (or very loosely structured) Web pages such as Wikipedia. The very nature of information extraction entails that the knowledge in these databases may exhibit a high degree of uncertainty or even inconsistency. For example, disambiguation problems may introduce false assertions, or inaccurate claims on Web pages may be integrated into the knowledge base. A similar observation holds for knowledge bases that have been constructed by other automated means such as data integration, fusion (aka. alignment) of different ontological sources, or the integration of user-generated content. This problem is not new. Even carefully hand-crafted knowledge bases such as SUMO [25] are known to exhibit inconsistencies [8]. One approach to deal with semantic query answering is to use first-order logic reasoning. This is for example the approach favored by the Semantic Web community. In this setting, the Web Ontology Language [4] (OWL) (or rather: one of its decidable subsets OWL-DL or OWL-lite) is used to describe inference rules and the query language SPARQL is used to formulate queries. First-order logic, however, cannot deal with inconsistencies. Once the knowledge base contains a single contradiction, virtually any fact could be inferred (*ex falso quod libet*).

Often, inconsistencies are not obvious at first glance but could only be uncovered through intricate and possibly expensive inference steps, generally referred to as *refutation* in logics [29]. These intricate relationships among data objects, as well as the lack of a strict relational schema, call for the use of graph-based,

non-schematic representation models like RDF/S, as well as novel—relaxed but efficient—query processing techniques for SPARQL, the W3C standard query language for RDF data. With RDF and SPARQL becoming the de facto standards for representing and querying light-weight knowledge representations, also novel forms of uncertain reasoning and new challenges for efficient query processing arise. That is, while OWL, the W3C Web Ontology Language, is known to be too heavyweight with powerful but expensive (i.e., undecidable in the general case) inference mechanisms, we focus on weaker variants based on OWL-lite which is itself embedded into RDF/S (see also results from [29]). These are substantially more lightweight, yet expressive enough as semantic knowledge representation, by providing a generic graph model for entities or so-called RDF resources (nodes) that can be connected via differently typed relations (edges), represented as atomic facts. In such ontology graphs, uncertainty can be modeled as a statistically quantified confidence weight for measuring the strength of a relationship between two entities (see, e.g., KnowItAll [14] and YAGO [32]).

Intuitively, facts that have been extracted from trustworthy sources or highly structured data could be assigned a greater confidence than facts that have been extracted by heuristic means from arbitrary Web documents. For example, assume that the user asks for the birth place of a person, and the knowledge base contains two contradictory facts. It is unclear which fact the system shall provide as an answer, unless there are confidence weights provided with these facts that indicate the correct answer. If, however, the correct fact has been assigned a lower confidence than the false answer in the original extraction step, additional inference rules may still help to produce evidence that favors the right answer (see the example in Subsection 1.4).

In most real-world settings, there are strict rules (often imposed as axiomatic constraints over the knowledge base) that may never be violated. For example, a person cannot be born in multiple places or on multiple dates. In most domains, specific instances of entities can be only of a single type (e.g., species) out of a mutually exclusive set of possible choices (as in the well-known Christmas bird-counting example [35]), unmovable objects can only be located at one place, movable objects can only be located at one place at a time, and so on. In URDF, a hard rule of the shape *bornIn(Al_Gore, x)* can be used to denote that all the potential birth places of the entity *Al Gore* (that can be bound to x in order to ground this pattern over the knowledge base) are mutually exclusive. Hard rules in the form of such mutually exclusive sets of facts can be employed to capture functional properties in OWL-lite, while other basic OWL-lite reasoning capabilities like transitivity and type inference can be captured by Datalog-style implications, which form soft rules that may be violated in our reasoning. For example, the transitivity of the *type* relation in RDFS/OWL-lite can easily be captured by a recursive rule *type*(x,y) $\wedge$ *type*(y,z) $\rightarrow$ *type*(x,z) in URDF.

1.3 Problem Statement

The goal of URDF is efficiently answering queries over potentially inconsistent knowledge bases with quantified degrees of uncertainty. Consequently, our approach for resolving inconsistencies is to cast this form of uncertain reasoning into a *generalization of the Weighted MAX-SAT problem* over Boolean formulas in conjunctive normal form with both soft and hard rules. Since this problem is NP-hard, we propose a novel and efficient approximation algorithm that is specifically tailored to the combination of soft and hard rules we investigate in this paper. While various of the aforementioned aspects have been studied in the context of probabilistic databases [5, 9, 11, 12], probabilistic extensions to Datalog [26], as well as efficient approximation algorithms for the classical Weighted MAX-SAT problem [19, 21], our focus in this work is on efficient query answering. Moreover, adding hard rules to this reasoning framework requires a generalization of the Weighted MAX-SAT problem to be taken into account, which has not been studied in prior literature over this combination of soft and hard rules.

1.4 Example

Consider the small example knowledge base depicted in Figure 1.1, which models a number of facts about computer scientists.

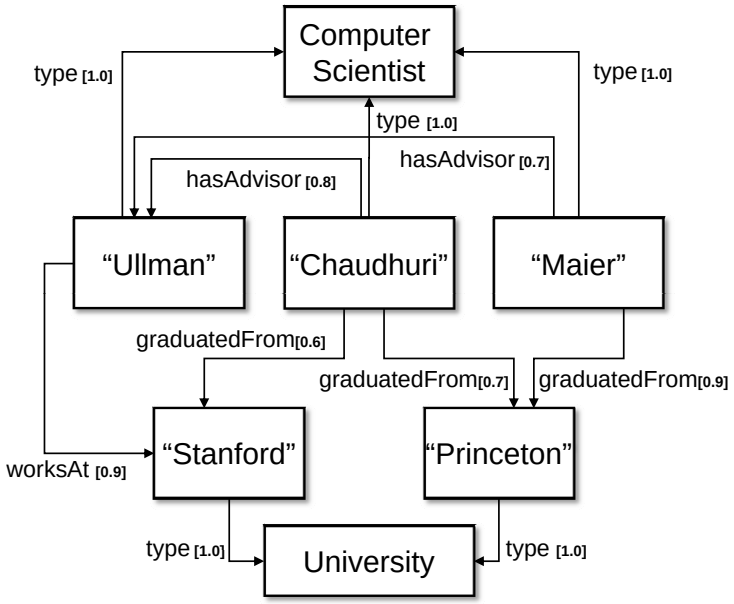

Figure 1.1: An example knowledge base in graph form.

In this scenario, we know, for example, that *Ullman* works at *Stanford* and he supervised both *Chaudhuri* and *Maier*. As we might have extracted from some Web page, *Maier* likely graduated from *Princeton*, but *Chaudhuri* may have graduated from both *Stanford* and *Princeton*, each with a given confidence that mirrors the certainty of the fact in the knowledge base. Notice that in the absence of any further constraints, there is no formal inconsistency in this representation. Let us assume, however, that we know for sure that both *Chaudhuri* and *Maier* each graduated from only one university.

Next, let us consider a SPARQL-like, conjunctive query as depicted in Figure 1.2: *Which computer scientist was supervised by Ullman, and where did he or she graduate from?*

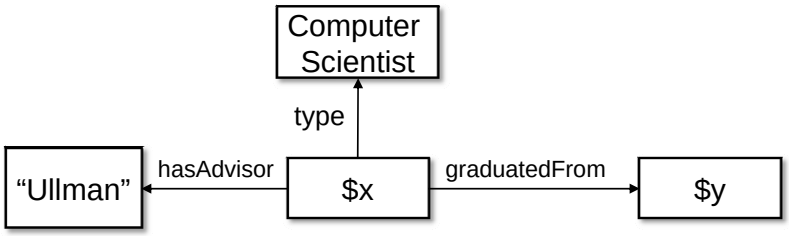

Figure 1.2: A query graph.

To answer this query, the engine needs to find isomorphic embeddings of the query graph in the knowledge base, each yielding a distinct binding of the query variables. In our case, the query will have two answers, since our knowledge base contains two computer scientists who each graduated from one of the two universities. Note that it may be impractical to precompute and exclude inconsistencies from the knowledge base upfront. Firstly, this precomputation may be prohibitively expensive for large knowledge bases—even when using an approximation algorithm. Secondly, and more importantly, excluding the inconsistent facts may be premature if the knowledge base is still evolving. A fact with a low certainty that is contradictory at the current point of time may receive a higher certainty if trustworthy sources are discovered for it at a later point in time. This is why URDF aims at resolving inconsistencies at *query time*. The goal is to always expose the most consistent interpretation of the current knowledge to the user.

In order to incorporate more expressive forms of first-order reasoning in URDF, we introduce a *soft inference rule* (i.e., an implication in Horn clause form) with a given confidence weight of 0.4 (C_7 taken from Table A.1 of Appendix A):

$$\textit{hasAcademicAdvisor(a,b)} \wedge \textit{worksAt(b,c)} \rightarrow \textit{graduatedFrom(a,c)}\ _{[0.4]}$$

By binding the variable a to either *Chaudhuri* or *Maier*, b to *Ullman*, and c to *Stanford*, we obtain two possible groundings of this rule over the base facts depicted

in Figure 1.1:

C_1 : $\neg$*hasAcademicAdvisor(Chaudhuri, Ullman)* $\vee$
$\neg$*worksAt(Ullman, Stanford)* $\vee$
graduatedFrom(Chaudhuri, Stanford) $_{[0.4]}$

C_2 : $\neg$*hasAcademicAdvisor(Maier, Ullman)* $\vee$
$\neg$*worksAt(Ullman, Stanford)* $\vee$
graduatedFrom(Maier, Stanford) $_{[0.4]}$

Notice that the fact *graduatedFrom(Maier, Stanford)* is *inferred* through the grounding of our inference rule over the knowledge base. Moreover, to avoid inconsistent answers of the type described above, URDF allows for specifying *hard mutual-exclusion constraints*. Such a constraint takes the form of a *competitor set*, i.e., a set of mutually exclusive facts. Here, two grounded competitor sets over base and derived facts in our knowledge base are:

S_1 : {*graduatedFrom(Chaudhuri, Stanford)*,
graduatedFrom(Chaudhuri, Princeton)}

S_2 : {*graduatedFrom(Maier, Stanford)*,
graduatedFrom(Maier, Princeton)}

S_1 and S_2 denote that only one out of the two possible answers can each be valid, i.e., we are now not only able to express that we are uncertain about the place where *Chaudhuri* and *Maier* graduated, but we can also express that we know for sure that *at most one* out of each of the two possible answers can be *true*. The elements of a competitor set do not have to be enumerated explicitly. Our framework allows for specifying competitor sets also more abstractly, stating, for example, that people graduate from at most one university, or people are not born in more than one place. Conceptually, however, we are dealing in both cases with a set of facts, out of which *at most one* can be *true*. Our algorithm is designed for hard rules of this shape. In Horn clause notation, this mutual exclusion can be expressed as a clause with only negated literals.

We can now combine all the grounded clauses and query atoms that are relevant for answering the query into a single Boolean formula in conjunctive normal form (CNF) as shown in Figure 1.3. Notice that all relevant base facts are expanded into unit clauses for the CNF, each consisting only of a single literal with a positive sign to aggregate more weight if base facts are matched. Inferred facts like *graduatedFrom(Maier, Stanford)* are not taken into this CNF, since setting *graduatedFrom(Maier, Stanford)* to *true* would already result in accumulating the weight of the grounded rule C_2 that generated this fact. Moreover, not all

$$
\begin{aligned}
& (\ \neg \textit{graduatedFrom(Chaudhuri, Stanford)} \vee \\
& \quad \neg \textit{graduatedFrom(Chaudhuri, Princeton)}\ )\ \blacksquare \\
& (\ \neg \textit{graduatedFrom(Maier, Stanford)} \vee \\
& \quad \neg \textit{graduatedFrom(Maier, Princeton)}\ )\ \blacksquare \\
\wedge\ & (\ \neg \textit{hasAcademicAdvisor(Chaudhuri, Ullman)} \vee \\
& \quad \neg \textit{worksAt(Ullman, Stanford)} \vee \\
& \quad \textit{graduatedFrom(Chaudhuri, Stanford)}\ )_{[0.4]} \\
\wedge\ & (\ \neg \textit{hasAcademicAdvisor(Maier, Ullman)} \vee \\
& \quad \neg \textit{worksAt(Ullman, Stanford)} \vee \\
& \quad \textit{graduatedFrom(Maier, Stanford)}\ )_{[0.4]} \\
\wedge\ & \textit{type(Chaudhuri, Computer\_Scientist)}_{[1.0]} \\
\wedge\ & \textit{type(Maier, Computer\_Scientist)}_{[1.0]} \\
\wedge\ & \textit{worksAt(Ullman, Stanford)}_{[0.9]} \\
\wedge\ & \textit{hasAcademicAdvisor(Chaudhuri, Ullman)}_{[0.8]} \\
\wedge\ & \textit{hasAcademicAdvisor(Maier, Ullman)}_{[0.7]} \\
\wedge\ & \textit{graduatedFrom(Chaudhuri, Stanford)}_{[0.6]} \\
\wedge\ & \textit{graduatedFrom(Chaudhuri, Princeton)}_{[0.7]} \\
\wedge\ & \textit{graduatedFrom(Maier, Princeton)}_{[0.9]}
\end{aligned}
$$

Figure 1.3: CNF formula with grounded rules and base facts over the knowledge base in Figure 1.1 and the query in Figure 1.2.

facts of Figure 1.1 are necessary for answering the query. The facts *type(Ullman, Computer_Scientist)*, *type(Stanford, University)* and *type(Princeton, University)* are not involved in grounding any query or rule atoms. We thus refer to facts in the CNF as the *dependency graph* of this query (see Section 4), which typically is much more compact than the entire knowledge base and thus will be key for efficient query answering.

From hard rule S_1, we immediately see that the above CNF is *unsatisfiable*, i.e., there is no truth assignment to atomic facts such that the entire CNF is satisfied. Solving this problem thus resolves to a generalization of the Weighted MAX-SAT problem: find a truth assignment to all atomic facts that maximizes the sum of the weights for the satisfied clauses in the CNF, without violating any of the hard constraints (marked by $\blacksquare$). A formal definition of this problem is deferred to Section 5. In this still fairly compact example, it is easily verifiable that the maximum weight is given by assigning *false* to *graduatedFrom(Chaudhuri, Princeton)* and *graduatedFrom(Maier, Stanford)*, and *true* to all other facts, even

though the fact *graduatedFrom(Chaudhuri, Princeton)* has a higher weight than *graduatedFrom(Chaudhuri, Stanford)* in the knowledge base. That is, both the soft and hard rules were crucial for finding the two correct answers. That is, in the absence of soft rule C_1 (and C_2), the MAX-SAT optimum would have been given by assigning *true* to *graduatedFrom(Chaudhuri, Princeton)* (and to *graduatedFrom(Maier, Stanford)*, resp.). Without the hard constraints S_1 (and S_2), on the other hand, the optimal MAX-SAT solution would have assigned *true* to both answers for *Chaudhuri* (and *Maier*, resp.). In both cases, we would have returned a wrong answer. Finally notice that the above example itself is a counter example for our soft inference rule: *Maier* indeed graduated from *Princeton*, although he was supervised by *Ullman* who worked at *Stanford*.

1.5 Contributions

URDF introduces a novel algorithm for answering *recursive queries* over uncertain and potentially inconsistent knowledge bases. More specifically, we summarize our contributions as follows:

- We formally define a novel *top-down reasoning framework* for graph-based, non-schematic RDF knowledge bases with both soft and hard rules. The reasoning is triggered upon query time and is able to *dynamically* resolve potential inconsistencies between the knowledge base and our inference rules. Unlike simpler bottom-up approaches, it is restricted to a compact subset of the knowledge base, the *dependency graph*, consisting only of facts that are relevant for answering the query. The framework works by grounding first-order formulas against the knowledge base. As such, it also supports probabilistic interpretations based on the common possible worlds semantics (with independent base facts), as well as probabilistic first-order models based on Markov Logic Networks.
- Core of our approach is a novel and efficient approximation algorithm for a generalization of the Weighted MAX-SAT problem that directly exploits the presence of hard rules, with more than two orders of magnitude performance gain compared to state-of-the-art MCMC sampling techniques such as MAP inference and MC-SAT.
- We present an extensive experimental evaluation, including a real-world, large-scale knowledge base with more than 20 million facts, using both handcrafted (common-sense) rules and synthetic rule expansions to empirically evaluate the approximation guarantee and asymptotic runtime of our algorithm.

2 Related Work

State-of-the-art query engines for SPARQL (see, e.g., [2, 24]) primarily focus on conjunctive query processing techniques on top of a relational encoding of RDF data. They often employ a so-called "triple-store" technique with multiple (redundant) index structures over different permutations of attributes in the RDF triplets. These engines do not have a notion of rule-based, recursive inference or uncertain reasoning with inconsistencies.

Approaches for managing uncertainty in the context of probabilistic databases [5, 9, 11, 12] focus on relational data with fixed schemata and often require strong independence assumptions among data objects (the base data). Moreover, Uncertainty and Lineage Databases (ULDBs) [9, 35] have been shown to provide a closed and complete representation formalism for modeling uncertainty in databases. Lineage-enabled databases (as a special form of intensional databases [13, 16]) provide a means for encoding "hard" Boolean constraints among relational tuples. Although lineage allows for the encoding of any finite set of possible instances of the uncertain database, ULDBs have no notion of recursive queries or "soft" inference. Resolving recursive inferencing rules with an unknown number of joins would quickly result in a rather circuitous relational encoding of data lineage. Recently, efficient top-k query processing and unified ranking approaches [23] have investigated different semantics of ranking queries over uncertain data. Moreover, the modeling of correlated tuples [30] with probabilistic graphical models such as Bayesian [10] and Markovian approaches [18, 28] is finding increasing attention in the database community. In [22], the authors define a class of Markov networks where query evaluation with soft rules can be done in polynomial time, while the case with hard rules is considered separately in [13]. In [33], the authors provide a self-calibrating framework for probabilistic inference over RFID streams to clean noisy and incomplete input also from streaming data.

Statistical relational learning (SRL) [17] is winning a gaining momentum in both the database and machine learning communities, with Markov Logic Networks [28] probably being the most generic approach for combining first-order logic and probabilistic graphical models into a single representation formalism.

Markov Logic however only allows for simulating hard constraints via automatically assigned high weights. Here, Markov Chain Monte Carlo (MCMC) [18, 27, 31] sampling methods provide a family of efficient approximation algorithms for these classes of probabilistic graphical models. Moreover, probabilistic extensions to Datalog have been studied in [26], but no management of inconsistencies has been discussed in this context.

Finally, the maximum satisfiability problem over Horn clauses (coined MAX-HORN-SAT) has been studied in detail in [21]. In [19], the authors provide a 3/4 approximation algorithm for the more general Weighted MAX-SAT problem of Boolean CNF formulas. However, none of these approaches considers a distinction between soft and hard constraints, which radically impacts the problem setting. To the best of our knowledge, our approach is the first to tackle this generalization of the Weighted MAX-SAT problem for Horn clauses with soft and hard rules.

3 Data and Representation Model

We define a *knowledge base* $\mathcal{KB} = \{\mathcal{F}, \mathcal{C}, \mathcal{S}\}$ as a triple consisting of RDF base facts $\mathcal{F}$, soft clauses $\mathcal{C}$, and hard (i.e., strict) rules $\mathcal{S}$. An *RDF graph* [1] is a directed, labeled multi-graph, in which nodes are entities (such as individuals and literals), and labeled edges represent relationships between the entities. For example, an RDF graph can have an edge between the entity *Ullman* and the entity *Stanford*. This edge would be labeled with the relation name *worksAt*. More formally, an RDF graph over a finite set of relations Rel and a finite set of entities $Ent \supseteq Rel$ is a set of triplets (or facts) $\mathcal{F} \subset (Rel \times Ent \times Ent)$. RDF allows two entities to be connected by multiple relations (e.g., two people can be colleagues and friends at the same time). Thus facts express binary relationships between entities. For readability, we will write a fact *(x,r,y)* in common prefix notation as *r(x,y)*.

3.1 Soft Rules

We consider first-order logic rules over RDF facts. A *grounded soft rule* over a set $\mathcal{F}$ of RDF facts is a set $C \subseteq \mathcal{F}$ of facts, where each atomic fact $f \in C$ is marked as either positive or negative and thus becomes a *literal*. For example, a grounded rule could be:

$$\{\neg \textit{worksAt(Ullman, Stanford)},\ \textit{livesIn(Ullman, Stanford)}\}_{[0.4]}$$

Each soft rule is assigned a non-negative, real-valued weight. A higher weight indicates that matching the rules is of higher importance than matching a rule with a lower weight. To simplify talking about grounded rules of the same shape, we introduce *non-grounded rules*. A non-grounded rule C' is a grounded rule C over a set of facts in $\mathcal{F}$, where one or more entities are replaced by variables. We denote variables by lowercase identifiers (with all variables implicitly being *universally quantified*). Thus, a non-grounded rule C' over $\mathcal{F}$ implicitly stands for all grounded rules C that can be obtained by substituting the variables in C' by entities. Thus, the following rule subsumes the aforementioned grounded rule:

$\{\neg$*worksAt(Ullman, x)*, *livesIn(Ullman, x)*$\}_{[0.4]}$

When grounded, the weight of the ungrounded rule is propagated to all its groundings. We use non-grounded rules purely to increase readability. We allow only *Horn rules*, i.e., rules where at most one literal is positive. Horn rules with exactly one positive literal can equivalently be rewritten as implications, in which all literals are positive. When written as implication, the *body* of a rule is a conjunction and the *head* consists of a single literal. In a first-order representation only simple literals with no nested predicates or function calls are allowed in the rules. We can however extend the expressiveness of our reasoner (and yet remain in first-order) by allowing also rules with simple *arithmetic predicates*, which are "closed" within the rule, i.e., they can be evaluated (and thus be grounded) on-the-fly from the given variable bindings when the rule is processed. Table A.1 in the Appendix depicts a set of hand-crafted soft rules we used for our baseline experiments.

A grounded soft rule corresponds to a disjunction of literals, a so-called *clause*. Given a set of facts $\mathcal{F}$, a *possible world* is a total function $p : \mathcal{F} \rightarrow \{true, false\}$, i.e., an assignment of facts to Boolean truth values. A clause is called *satisfied* in a possible world p, if for at least one positive literal $f \in C$, $p(f) = true$, or for at least one negated literal $\bar{f} \in C$, $p(\bar{f}) = false$.

3.2 Hard Rules

Hard rules are a distinct set of rules which define *mutually exclusive* sets of facts. Similarly to soft rules, hard rules can be expressed both in grounded and non-grounded form. A *grounded hard rule* is a set of facts $S \subseteq \mathcal{F}$ (also called a *competitor set*) that enforces the following constraint: a possible world $p : \mathcal{F} \rightarrow \{true, false\}$ can assign $true$ to at most one fact $f \in S$. We shall denote the set of hard rules with $\mathcal{S}$. For example the following hard rule

{ *bornIn(Al_Gore, USA)* ,
bornIn(Al_Gore, Italy) ,
bornIn(Al_Gore, Spain) } ■

specifies that *Al Gore* could be born in at most one out of the above countries. Hard rules strictly partition $\mathcal{F}$, i.e., every fact $f \in \mathcal{F}$ is contained in exactly one of the competitor sets. A fact f that is not in conflict with any other fact in $\mathcal{F}$ is assigned to a unit competitor set $S = \{f\}$, consisting only of that fact (see also Section 5). Similarly to soft rules, we introduce *non-grounded hard rules* where constants may be replaced by variables. For example, *bornIn(Al_Gore,x)* may be used to mark all the possible birth places of *Al Gore* in the knowledge base as mutually exclusive.

A hard rule may equivalently be rewritten as a number of conjunctions of binary Horn clauses with only negated literals. So is, for example, the above grounded hard rule equivalent to the following conjunction of binary Horn clauses:

$$
\begin{aligned}
&(\ \neg \textit{bornIn(Al\_Gore, USA)} \\
&\quad \vee \neg \textit{bornIn(Al\_Gore, Italy)} \ ) \ \blacksquare \\
\wedge \ &(\ \neg \textit{bornIn(Al\_Gore, USA)} \\
&\quad \vee \neg \textit{bornIn(Al\_Gore, Spain)} \ ) \ \blacksquare \\
\wedge \ &(\ \neg \textit{bornIn(Al\_Gore, Spain)} \\
&\quad \vee \neg \textit{bornIn(Al\_Gore, Italy)} \ ) \ \blacksquare
\end{aligned}
$$

Since competitor sets are used as a distinct input to our MAX-SAT algorithm, there is no need to explicitly flatten these sets into their Horn clause form (which would otherwise yield formulas that are quadratic in the sizes of the competitor sets). Hard rules may not be violated and thus have no weights assigned (hence they are marked by a $\blacksquare$).

We remark that only pairwisely disjoint competitor sets $S \in \mathcal{S}$ are allowed as hard rules in our reasoning framework. If hard rules $\mathcal{S}$ do not form a proper partitioning of $\mathcal{F}$, we cannot guarantee that there is always a truth assignment to all facts $f \in \mathcal{F}$ such that none of the hard rules is violated. As a simple counter example, consider a single fact A with $\mathcal{F} = \{A\}$ and the following set of two hard rules $\mathcal{S} = \{\{A\}, \{\neg A\}\}$ which would be translated into CNF as $A \ \blacksquare \ \wedge \ \neg A \ \blacksquare$.

3.3 Soft Rules vs. Hard Rules

From the above discussion, one might argue that every hard rule $S \in \mathcal{S}$ could rather be turned into a set $\mathcal{C}_S$ of soft rules, each with a high weight:

$$
\begin{aligned}
&(\ \neg \textit{bornIn(Al\_Gore, USA)} \\
&\quad \vee \neg \textit{bornIn(Al\_Gore, Italy)} \ )_{[100]} \\
\wedge \ &(\ \neg \textit{bornIn(Al\_Gore, USA)} \\
&\quad \vee \neg \textit{bornIn(Al\_Gore, Spain)} \ )_{[100]} \\
\wedge \ &(\ \neg \textit{bornIn(Al\_Gore, Spain)} \\
&\quad \vee \neg \textit{bornIn(Al\_Gore, Italy)} \ )_{[100]}
\end{aligned}
$$

Observe that any truth assignment assigning *true* to two or more facts in S would still violate at least one of the newly introduced soft rules in $\mathcal{C}_S$.

We might then wonder if this "trick" could be used when our objective is to maximize the total weight of satisfied clauses. The answer is 'yes' if we had an exact algorithm for the weighted MAX-SAT problem. The main idea is to turn each hard rule S into a set of "artificial" soft rules $\mathcal{C}_S$, each of them having a sufficiently large weight (the total weight of the original soft clauses is sufficient). Let w be the total weight of all the artificial soft rules. It can be shown that there is

a solution of weight W for the original instance of the problem (with hard rules), if and only if there is a solution with weight $W + w$ for the new problem (without hard rules).

Unfortunately, things are different if we must use an approximation algorithm, as in our case. The main problem is that it is unclear how to set the weights of the artificial clauses. If they are too large, then the weights of the original soft clauses might not be relevant anymore and so the solution returned might be meaningless. On the other hand, if the weights for $\mathcal{C}_S$ are too small, then we might violate some of the original hard rules which were turned into soft rules. This is illustrated by the following example. Let us consider again the aforementioned hard rules converted into soft rules $\mathcal{C}_S$ each of them having weight 100. In addition, assume we have the following original soft rules

$\wedge$ *bornIn(Al_Gore, Italy)*$_{[0.01]}$
$\wedge$ *bornIn(Al_Gore, Spain)*$_{[0.01]}$
$\wedge$ *bornIn(Al_Gore, USA)*$\}_{[0.98]}$
$\wedge$ *ranForPresident(Al_Gore, USA)*$_{[0.99]}$
$\wedge$ ($\neg$*bornIn(Al_Gore, USA)* $\vee$ *ranForPresident(Al_Gore, USA)*$)_{[0.99]}$

expressing some evidence that *Al Gore* was born in *Italy*, *Spain*, or *USA* and that he ran for president of the United States. There is also another rule specifying that he could run for president only if he was born in the United States. The optimum solution is to infer that *Al Gore* was born in *USA* and not in *Italy* nor in *Spain*, and that he indeed ran for the presidency of the United States of America. The total weight of the satisfied clauses by this assignment is 302.96. Suppose we have an algorithm for Weighted MAX-SAT (without hard rules) with an almost perfect approximation guarantee of 0.99 that receives in input the soft clauses above. Such an algorithm might infer that *Al Gore* was actually born in *Italy* and he did not run for president of *USA*, in that, the approximation ratio of this solution is $300.01/302.96 > 0.99$.

Besides this fundamental issue there is also an efficiency issue in replacing hard rules with soft rules: the number of soft clauses becomes quadratic in the size of the largest hard rule, which of course should be avoided.

3.4 Estimating Weights for Soft Rules

Estimating the weight of a clause for the MAX-SAT setting relates to the field of Inductive Logic Programming [29] (i.e., learning rules from data). Here, we do not consider this problem in full, we just aim for a reasonable estimation for the weight of a clause, given a knowledge base $\mathcal{KB}$ as prior knowledge. One possible option for estimating these weights is to count the number of grounded instances $numGround$ of each rule in the knowledge base. That is, consider a

non-grounded rule $C: \ f_1 \wedge \cdots \wedge f_{n-1} \rightarrow f_n$. Using just the absolute number of groundings of this rule as weight may lead to highly biased rule weights. That is, a rule may have very many grounded instances, but it may still be very generic. For example, the rule *bornIn(x,a)* $\rightarrow$ *diedIn(x,y)* has very many grounded instances, because very many people indeed died where they were born. However, a majority of people did not die where they were born, such that the rule may be wrong in a majority of cases. Rather, the weight should reflect the likelihood that the rule is correct, because, intuitively, a MAX-SAT solver should give higher priority to satisfying a rule that is often right. We thus consider the proportion of right cases over false cases—in other words—a conditional probability denoting the likelihood that the entire rule is *true*, given that the body of the rule is *true*. That is, for a soft rule C with head f_n, we compute the weight $w(C)$ as:

$$w(C) \quad = \quad \frac{numGround(f_1 \wedge \cdots \wedge f_{n-1} \wedge f_n)}{numGround(f_1 \wedge \cdots \wedge f_{n-1})}$$

Table A.1 in the Appendix depicts these ratios for a number of rules, with weights estimated over a large knowledge base [32].

4 URDF Reasoning Framework

The URDF reasoning framework combines classic Datalog-style first-order reasoning with a generalized Weighted MAX-SAT solver for both soft and hard rules. That is, given a query in the form of a set of non-grounded atoms, we aim to find an assignment of truth values to the grounded query atoms (and all other grounded facts that are relevant for answering the query), such that the sum of the weights over the satisfied soft rules is maximized, without violating any of the hard constraints. In the absence of any rules, URDF conforms to a standard (conjunctive) SPARQL engine, with the relevant facts consisting only of grounded query atoms over base facts $\mathcal{F}$. URDF however allows for the formulation of recursive rules (i.e., with the same predicate occurring in the head as well as in the body of a rule), as well as mutually recursive sets of rules (i.e., with one rule producing grounded facts as input to another rule). For this purpose, we define the *dependency graph* $\mathcal{D}$ as follows.

4.1 Dependency Graph Construction

We formally define the *dependency graph* $\mathcal{D}$ as follows (recall the example in Section 1.4 for a complete illustration of the following steps).

Definition 1 *Dependency Graph. Given a knowledge base $\mathcal{KB} = \{\mathcal{F}, \mathcal{C}, \mathcal{S}\}$ and a conjunctive query Q, then:*

- *All possible groundings $f \in \mathcal{F}$ of the query atoms $q \in Q$ are facts in the dependency graph $\mathcal{D}$.*
- *If a grounded fact f_n is in $\mathcal{D}$, then all grounded facts $f_1, \ldots, f_{n-1}$ of all grounded soft rules $C \in \mathcal{C}$, in which f_n is the head, are also in $\mathcal{D}$.*
- *If a grounded fact f is in $\mathcal{D}$, then all grounded facts $f_1, \ldots, f_k$ of the grounded hard rule $S \in \mathcal{S}$, which are mutually exclusive to f, are also in $\mathcal{D}$.*

Notice that Definition 1 already yields a recursive algorithm to compute the dependency graph, called SLD resolution [6] in Prolog and Datalog. Moreover, in

order to keep query processing (and in particular the recursive rule evaluations) as tractable as possible, we enforce the following restrictions on queries and rules, yielding a more restricted setting than regular Datalog:

- *Query Binding.* Each query needs to have at least on atom with at least one constant (i.e., we reason about at least one known entity).
- *Domain Restriction.* In each rule, every variable (and constant) that appears in a positive literal (head) must also appear in at least one of the negated literals (body). All bindings of variables in a rule (or query) need to be chained, i.e., be reachable from any other atom in the same rule (or query).
- *Closed-World Assumption.* Every fact that cannot ultimately be grounded in the knowledge base (including the recursive resolution of the soft rules), or be resolved as a simple arithmetic predicate, cannot be *true* and thus is not expanded into the dependency graph.

The query binding constraint ensures that the dependency graph always forms a connected subgraph in $\mathcal{F}$ and is the same for all distinct groundings of the query. It also ensures that resolution starts with a non-empty set of variable bindings. Due to the domain restriction, it is never necessary to instantiate a variable with all possible entities in the knowledge base, which conforms to SLD resolution in Datalog. The closed-world assumption finally is a common restriction also in Datalog. Moreover, allowing only strict Horn clauses with at most one positive literal guarantees *stratifiable* Datalog programs [26] programs.

4.2 Reasoning Framework

Our general reasoning framework is summarized in Algorithm 1. The grounding

Algorithm 1 URDF Reasoning Framework

Require: A knowledge base $\mathcal{KB}$ with base facts $\mathcal{F}$, soft rules $\mathcal{C}$ and hard rules $\mathcal{S}$
Require: A non-grounded conjunctive query Q
1: Initialize the dependency graph $\mathcal{D} = \emptyset$
2: Ground all $q \in Q$ via SLD resolution and add results to $\mathcal{D}$
3: $\cdot$ Expand $\mathcal{C}$ by each grounded soft rule C embedded in $\mathcal{D}$
4: $\cdot$ Expand $\mathcal{S}$ by each grounded hard rule S embedded in $\mathcal{D}$
5: Construct the CNF from all grounded soft rules $C \in \mathcal{C}$, hard rules $S \in \mathcal{S}$, and base facts $f \in \mathcal{D} \subseteq \mathcal{F}$
6: Solve Weighted MAX-SAT over the CNF
7: **return** $\mathcal{D}$ with truth assignments $p(f)$ to all facts $f \in \mathcal{D} \subseteq \mathcal{F}$

step in Line 2 of Algorithm 1 calls an SLD resolution (Algorithm 2) for all query atoms. This conforms to a standard (top-down) SLD resolution over the soft rules,

which has additionally been extended by a grounding phase for the hard rules at each recursion level. Both steps have a well-known polynomial complexity for exact inferencing over Horn clauses (known as HORN-SAT). Lines 3 and 4 denote the rules that were grounded during this resolution phase in order to construct the CNF. These grounded rules are already available from the previous SLD resolution and can be kept in a simple buffer of the algorithm. The CNF construction in Line 5 itself is linear in the size of $\mathcal{D}$ and $\mathcal{C}$ for our MAX-SAT algorithm, because all grounded soft rules are already in their clause form. Hard rules can be input into our MAX-SAT solver (see Section 3.2) directly as mutually exclusive sets of facts without having to flatten them into pairwise mutual exclusions (which could otherwise result in a quadratic inflation of the CNF). Line 6 however is a generalization of the Weighted MAX-SAT problem and thus is NP-hard.

4.3 SLD Resolution with Soft and Hard Rules

Datalog-style SLD resolution with soft and hard rules requires an adaptation, in particular to also work with non-grounded hard rules $\mathcal{S}$ (competitor sets). In our framework, hard rules can either be specified as predefined sets of grounded atoms or as non-grounded atoms with variables. Thus, whenever a newly grounded fact f is added to the dependency graph $\mathcal{D}$, the algorithm also needs to make sure that all facts f_i, which are in conflict with f, are also added to the dependency graph. That is, in these cases additional SLD resolutions steps are triggered for all f_i, using all non-grounded atoms q in hard rules $S \in \mathcal{S}$ that match f as queries, which may in turn recursively trigger further SLD grounding steps.

Algorithm 2 shows that queries and body atoms of soft rules are grounded in the exact same way, as both are processed as conjunctive sets of non-grounded atoms (denoted as Q). Consequently, the recursive resolution step in Algorithm 2 is called initially with a set of one or more conjunctive query atoms and then, at each recursion level, similarly with the atoms of the body of the rule that is currently processed. Moreover, Algorithm 2 creates a temporary dependency graph $\mathcal{D}'$ at each recursion step, which is only expanded into the dependency graph $\mathcal{D}$ of the superordinate recursion if all atoms in the body of the rule are matched conjunctively. That is, whenever a resolution step in Algorithm 2 finds a valid grounding for a conjunctive set of query or rule atoms, either in the form of a base fact $f \in \mathcal{F}$ (Line 5) or for a derived head f_n of a soft rule $C \in \mathcal{C}$ (Line 10), the temporary dependency graph $\mathcal{D}'$ is expanded by these grounded facts. Ungrounded atoms in hard rules, on the other hand, are grounded individually to form sets of mutually exclusive facts (Line 12). Alternatively, any combination of grounded (e.g., arbitrary, user-defined) competitor sets may be given directly as input to the Weighted MAX-SAT solver, thus skipping steps 12–14, as long as

Algorithm 2 SLD Resolution with Soft and Hard Rules

Require: A knowledge base $\mathcal{KB}$ with base facts $\mathcal{F}$, soft rules $\mathcal{C}$, and hard rules $\mathcal{S}$
Require: A non-grounded conjunctive query Q
Require: A dependency graph $\mathcal{D}$
Initialize a temporary dependency graph $\mathcal{D}' = \emptyset$
Sort all atoms $q \in Q$ in ascending order of estimated selectivity
For all $q \in Q$
· If there is a match f to q in $\mathcal{F}$
· · Add f to $\mathcal{D}'$
· For all $C \in \mathcal{C}$
· · Let f_n be the head and $f_1, \ldots, f_{n-1}$ be the body of C
· · If the head f_n of C matches q
· · · Ground all $f_1, \ldots, f_{n-1}$ recursively via SLD resolution
· · · and add results to $\mathcal{D}'$
· · · Add f_n to $\mathcal{D}'$
If there is a conjunctive grounding of all $q \in Q$
· For all $S \in \mathcal{S}$
· · If a fact $f \in \mathcal{D}'$ matches an ungrounded atom $q \in S$
· · · Ground q recursively via SLD resolution
· · · and add results to $\mathcal{D}'$
· Expand $\mathcal{D}$ by all grounded facts in $\mathcal{D}'$
return $\mathcal{D}$

they form disjoint sets of grounded atoms.

We remark that this form of dependency graph expansion guarantees consistent query answering. That is, the truth assignments to facts in the dependency graph $\mathcal{D}$ after MAX-SAT solving are equivalent to the truth assignments that would be obtained for these facts when running the MAX-SAT solver over the entire knowledge base $\mathcal{KB}$ (disregarding possible approximation errors due to the NP-hardness).

4.4 Propositional vs. Probabilistic Reasoning

Query processing in URDF works by grounding first-order rules against the base facts, using a conjunctive set of (ungrounded) query atoms as anchors. After the grounding step, the reasoner works over a purely propositional setting, i.e., no higher-order rules are involved in the final reasoning step anymore. This strategy is along the lines of probabilistic database approaches following a possible worlds semantics [5, 9, 11], as well as probabilistic graphical models such as Markov

Logic Networks [18, 28], which also work by grounding. Since SLD resolution is acyclic and $\mathcal{KB}$ is finite, the resulting grounded model is guaranteed to be finite and the derivation (also coined "provenance" or "lineage" in [9, 35]) of each inferred fact is acyclic, i.e., after the grounding phase, common possible worlds semantics is applicable to our model as well. More generally, our URDF framework supports a wide range of reasoning semantics:

- *Propositional Reasoning.* We present only the most consistent view (possible world) over a potentially inconsistent knowledge base as answer, using an efficient MAX-SAT approximation algorithm for resolving inconsistencies. This strategy returns truth assignments to all facts in the dependency graph.
- *Probabilistic Reasoning.* Assuming independence among base facts, we present a compact representation of all possible worlds (weighted by a probability) as answer. This strategy also allows for assigning result probabilities to inferred facts via marginalization. As a form of so-called intensional databases [12, 13], the confidences for derived facts can be computed using their acyclic lineage information, thus using only independent base facts as input.[1] Confidence computations in these settings are known to be $\#P$-complete [13].
- *Markov Logic Networks.* This framework combines Markov networks and first-order logic [28], which, as a form of probabilistic graphical models, allows for less strict independence assumptions for facts co-occurring in grounded formulas. Inference in probabilistic graphical models in general is NP-hard. This is why efficient sampling techniques such as Markov Chain Monte Carlo (MCMC) [27, 31] are devised as approximation.

In the following, we focus on the propositional representation as input for our MAX-SAT solver. This approach is motivated by real-world query answering in semantic knowledge bases with SPARQL-like queries, where the user often seeks a deterministic answer to a conjunctive SPARQL query with multiple ungrounded atoms at a time. In this setting, queries are used as templates for sets of answer facts which are typically rather specific. As such, they are likely to return only very few answers—or no answers at all—when a strict adherence to the data and rules would be used for inference. In particular with hard mutual exclusiveness constraints, assigning *true* to some facts may rule out the existence of other facts which in turn may have been required to derive further facts (including potential answers).

Cleaning inconsistencies from answers cannot easily be performed by assigning probabilities to answer facts, with a majority of facts still having non-zero probabilities. A MAX-SAT solver with hard rules in the form of these mutual-

[1] A rescaling of confidences may be required if mutually exclusive sets of facts are not guaranteed to form a probability distribution in $\mathcal{KB}$ a priory.

exclusiveness constraints, on the other hand, assigns *true* always to a consistent subset of answers facts. Enforcing strict mutual-exclusiveness constraints among facts can only be guaranteed by hard rules which may not be violated by any solution.

We next provide a detailed description and analysis of our MAX-SAT approximation algorithm, while a comparison to MCMC sampling techniques is provided in the experiments.

5 Weighted MAX-SAT with Soft and Hard Rules

Given a knowledge base $\mathcal{KB} = \{\mathcal{F}, \mathcal{C}, \mathcal{S}\}$, we define an instance of the MAX-SAT problem with hard and soft rules as follows. For every fact in $\mathcal{F}$, we introduce a Boolean variable f as well as a unit clause $C = \{f\}$ into $\mathcal{C}$. The *weight* $w(C)$ of such a newly introduced unit clause can be set equal to the initial confidence of the fact, e.g., using the confidence weight of the fact in the knowledge base as input. Moreover, we define the collection of competitor sets to be a *partitioning* of the set of facts $\mathcal{F}$. For convenience of notation, for each fact f that does not belong to any given competitor set, we analogously introduce a unit competitor set $\{f\}$ into $\mathcal{S}$, consisting only of that fact.

PROBLEM DEFINITION. *We are given a set of variables (facts)* $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$*, a collection* $\mathcal{C} = C_1, \ldots, C_m$ *of clauses in Horn form, and a collection of pairwise disjoint competitor sets* $\mathcal{S} = S_1, \ldots, S_t$*, where* $S_k \subseteq \mathcal{F},\ k = 1, \ldots, t$*. Each clause* $C \in \mathcal{C}$ *is associated with a positive weight* $w(C)$ *and consists of a disjunction of literals (each literal is either a variable* $f \in \mathcal{F}$ *or its negation* $\bar{f}$*). The objective is to find a truth assignment to each variable* $f \in \mathcal{F}$ *such that for each competitor set at most one fact is assigned the value* true*, and the sum of the weight of the satisfied clauses (i.e., clauses with at least one* true *literal) is maximized.*

As this problem is a generalization of MAX-SAT which is NP-Hard [19], it follows that also MAX-SAT with hard and soft rules is NP-Hard. Moreover, the variant when the soft rules are Horn clauses is also NP-Hard [21].

5.1 Weighted MAX-SAT Algorithm

Because of the intractability to compute an optimum solution for the above problem, we resort to devise an approximation algorithm. An approximation algorithm

for a problem has an approximation guarantee of α if the ratio between the value of the solution computed by the algorithm and the value of an optimum solution is larger than α, for every instance of that problem.

Our algorithm is based on a rather powerful technique in approximation algorithms theory, called the *method of conditional probabilities* [3]. In short, this technique is a special case of *derandomization* and consists of converting a randomized approximation algorithm into a deterministic one while preserving the approximation guarantee. The main advantage of this approach is that often the resulting deterministic algorithms are efficient. The randomized algorithm first computes a probability p_i for each fact with the following property: the sum of all p_i's over a same competitor set is at most one. Then, independently for each competitor set, a fact is picked accordingly to the computed probability distribution and its value is set to *true*. Note that it might be the case that all facts in the same competitor set might be set to false (as the sum of the corresponding p_i's might be less than one). The method of conditional probabilities then shows us the way to obtain a deterministic algorithm. Initially all facts are unassigned. At each step one fact is assigned *true* or *false* whichever value gives the largest value for a carefully defined potential function. Then, all satisfied clauses are removed and all false occurrences of facts are removed from our formula. Our algorithm terminates when all facts are assigned.

Our potential function can be interpreted as the expected total weight of satisfied clauses by the randomized algorithm. Formally, we denote with W_t the value of our potential function at step t. At the beginning of our algorithm all facts are unassigned and the value of our potential function (W_0) is defined by

$$W_0 = \sum_{C \in \mathcal{C}} w(C) \cdot \mathcal{P}(\text{C is satisfied}). \tag{5.1}$$

The algorithm successively finds a truth assignment to facts until all facts are assigned. At step $t-1$, let $\hat{f}_1, \ldots, \hat{f}_{t-1}$ be the truth assignment for the facts $f_1, \ldots, f_{t-1}$, and let S_k be a competitor set whose facts are all unassigned. Let $\bar{\eta}_k$ be the truth assignment that assigns *false* to all facts in S_k. For every f in S_k we define

$$W_{t,f} = \sum_{C \in \mathcal{C}} w(C) \cdot \mathcal{P}(\text{C is sat.}|\hat{f}_1, \ldots, \hat{f}_{t-1}, f = true), \tag{5.2}$$

where $\mathcal{P}(A|B)$ denotes a conditional probability, which is the value of the potential function when f is assigned *true*. When all facts in S_k are assigned *false* our potential function is denoted as

$$\bar{W}_t = \sum_{C \in \mathcal{C}} w(C) \cdot \mathcal{P}(\text{C is sat.}|\hat{f}_1, \ldots, \hat{f}_{t-1}, \bar{\eta}_k). \tag{5.3}$$

Then, our algorithm determines the largest value for our potential function among all $W_{t,f}$'s and $\bar{W}_t$, and assigns the corresponding truth values to the facts in S_k. Such a value for W shall be denoted with W_t. We remark that our algorithm is completely deterministic (i.e., it always produces the same output given the same input) even though this is obtained by starting from a randomized algorithm.

Algorithm 3 shows pseudocode for our algorithm. The quantity $w(f_i)$ is used to determine a tailored probability distribution over the set of facts (which is computed by Algorithm 4). This quantity consists of the total weight of unit clauses (denoted as "u.c.") that are satisfied by assigning *true* to f_i (and *false* to all other facts in the same competitor set of f_i). Formally, this is

$$w(f_i) = \sum_{\substack{j: f_i \in C_j \\ C_j \text{ is u.c.}}} w(C_j) + \sum_{\substack{l: f_l \neq f_i \\ f_l \in S_k}} \sum_{\substack{j: \bar{f}_l \in C_j \\ C_j \text{ is u.c.}}} w(C_j). \tag{5.4}$$

In Algorithm 3, q_i^j and $\bar{q}_k^j$ denote the probability that a clause C_j is satisfied when f_i is assigned *true* and the probability that C_j is satisfied when all facts in S_k are assigned *false*, respectively, while W_i^{true} and $\bar{W}_k$ denote the corresponding total expected weights of satisfied clauses (w.r.t. the aforementioned truth assignments). Algorithm 5 shows how to efficiently compute the probability that a clause is satisfied. Notice that truth assignments of facts in the same competitor set are never determined independently.

5.2 Algorithm Analysis

In this section, we prove the approximation guarantee of our algorithm. This is a function of the minimum number of negated literals in any clause, as shown in Theorem 1. Let $\mathcal{S} = \bigcup_{k=1}^{t} S_k$.

Theorem 1 *Let φ be a Horn Boolean formula with hard constraints and let λ be the minimum number of negated literals in any clause (of φ) having at least two literals. Our algorithm is a p-approximation algorithm for the MAX-SAT with hard rules problem, where p is obtained by solving the equation $p = 1 - p^{\lambda}$. The running time of our algorithm is $O(|\mathcal{C}| \cdot |\mathcal{S}|)$ in the worst case.*

PROOF. The proof goes along the lines of [3]. We show that our potential function is always within a constant fraction of the optimum. At the end of our algorithm, the value of our potential function is equal to the weight of the solution computed by the algorithm. Hence, the claim follows.

At the very beginning of the algorithm, a bound on W_0 is computed as follows. We shall deal first with the unit clauses. For each competitor set S_k, let $U_k \subseteq S_k$

Algorithm 3 Weighted MAX-SAT with Soft and Hard Rules

Let l be the minimum number of negated literals in any "soft" clause with at least two literals
Let p s.t. $p = 1 - p^l$;
Compute $w(f_i)$ for all f_i in F and remove all unit clauses from the Boolean formula
Execute Algorithm 4 for each competitor set
For each competitor set $S_k \in \mathcal{S}$
· Let $\mathcal{C}_k$ be the set of clauses containing at least one fact in S_k
· For each clause $C_j \in \mathcal{C}_k \subseteq \mathcal{C}$
· · Define q_i^j as follows. If $f_i \in C_j$ or $\bar{f}_l \in C_j$ then let $q_i^j = 1$
· · where $\bar{f}_l \in S_k$ and $f_l \neq f_i$; else set q_i^j to the value
· · computed by Alg. 5 with literals $C_j \setminus \left(\bar{f}_i \cup_{f_l \in S_k} f_l\right)$
· · Define $\bar{q}_k^j$ as follows. If there is $f_l \in S_k$ such that $\bar{f}_l \in C_j$
· · then let $\bar{q}_k^j = 1$; else set $\bar{q}_k^j$ to the value computed
· · by Alg. 5 with literals $C_j \setminus (\cup_{f_l \in S_k} f_l)$
· Let $W_i^{\textit{true}} = \sum_{C_j \in \mathcal{C}} q_j^i w(C_j)$ for each fact i in S_k
· Let $\bar{W}_k = \sum_{C_j \in \mathcal{C}} \bar{q}_j w(C_j)$
· Let f_i be the fact with the largest w_i among all facts in S_k
· If $W_i^{\textit{true}} > \bar{W}_k$ then set f_i to *true* and all other facts in S_k
· to *false*; else set all facts in S_k to *false*
· Remove all satisfied clauses and remove all literals
· whose value is *false*

be the set of facts to each of which Algorithm 4 assigns a probability p_i larger than zero. By definition of the p_i's, it follows that for each competitor set k

$$\sum_{f_i \in U_k} p_i \; w(f_i) = p \; w_{\max}, \tag{5.5}$$

where $w_{\max}$ is the weight of a fact f in S_k such that $w(f)$ is maximum. Now consider any non-unit clause with $\hat{l} \geq l$ negated facts. If there are at least two negated facts belonging to the same competitor set, then the probability that such a clause is satisfied is one. Otherwise, the probability that it is satisfied is at least $1 - p^{\hat{\lambda}} \geq 1 - p^{\lambda}$, as each negated literal becomes *true* with probability at most p and all negated literals belong to different competitor sets (hence their values are determined independently). Let X_t be the random variable indicating the total weight of satisfied clauses at step t. Let $\mathcal{C}_u \subseteq \mathcal{C}$ be the set of unit clauses. By

Algorithm 4 Computing Probabilities in Each Competitor Set

Require: A set S_k with t facts, a real value $0 \leq p \leq 1$;

Compute $w(f_i)$ for each f_i in S_k
Let $f_1, \ldots, f_t$ be the facts ordered by non-increasing $w(f_i)$'s
If $w(f_1) = 0$ then set $p_i = 0$ for each fact in S_k and return
Set $R = \emptyset$
For $i = 1, \ldots, t$
· $T = R \cup f_i$
· Let $x = p\frac{w(f_1)}{\sum_{f_i \in T} w(f_i)}$
· If $x|T| > 1$ then break
· $R \leftarrow T$
· $y \leftarrow x$
For each f_i if $f_i \in R$ set $p_i = y$ else $p_i = 0$
return A probability p_i for each fact f_i in S_k

Algorithm 5 Computing the Probability that a Clause is Satisfied

Require: A disjunctive clause $C = f_1 \vee f_2 \vee \cdots \vee f_t$, a probability p_l for each fact f_l

Partition the facts in C in $C_1, \ldots, C_k$ such that all facts in C_i belong to the same "competitor set"
For each set $C_i \subseteq S_j$ compute a value q_i as follows
· If C_i contains at least two negated facts then return 1
· If C_i contains exactly one negated fact $\bar{f}_l$, then
· set $q_i = 1 - p_l$; else set $q_i = \sum_{l \in C_i} p_l$
return $1 - \prod_{i=1}^{k}(1 - q_i)$, the probability that C is satisfied

putting all pieces together, we obtain

$$\begin{aligned} W_0 &= Exp[X_0] \\ &= \sum_{C \in \mathcal{C}} w(C) \cdot \mathcal{P}(C \text{ is satisfied}) \\ &= \sum_{C \in \mathcal{C}_u} w(C) \cdot \mathcal{P}(C \text{ is sat.}) + \sum_{C \in \mathcal{C} \setminus \mathcal{C}_u} w(C) \cdot \mathcal{P}(C \text{ is sat.}) \\ &\geq p \sum_{C \in \mathcal{C}_u} w(C) + \left(1 - p^{\lambda}\right) \sum_{C \in \mathcal{C} \setminus \mathcal{C}_u} w(C) \\ &\geq p \cdot OPT, \end{aligned} \tag{5.6}$$

where the last inequality follows from the way p has been chosen. We now show that $W_{t-1} \leq W_t$ for all steps t of our algorithm. Let $\hat{f}_1, \ldots, \hat{f}_{t-1}$ be the assignment

computed at step t by our algorithm and let S_k be a competitor set whose facts are still unassigned. We have that:

$$\begin{aligned} W_{t-1} &= Exp[X_t | f_1 = \hat{f}_1, \ldots, f_{t-1} = \hat{f}_{t-1}] \\ &= \sum_{f \in S_k} Exp[X_t | f_1 = \hat{f}_1, \ldots, f_{t-1} = \hat{f}_{t-1}, f = true] \cdot p_f \\ &+ Exp[X_t | f_1 = \hat{f}_1, \ldots, f_{t-1} = \hat{f}_{t-1}, \bar{\eta}_k] \cdot \sum_{f \in S_k} (1 - p_f) \\ &\leq \max(\max_{f \in S_k} Exp[X_t | f_1 = \hat{f}_1, \ldots, f_{t-1} = \hat{f}_{t-1}, f = true], \\ &Exp[X_t | f_1 = \hat{f}_1, \ldots, f_{t-1} = \hat{f}_{t-1}, \bar{\eta}_k) \\ &= W_t \end{aligned} \tag{5.7}$$

Finally, let $t_{\max}$ be the last step of our algorithm. At the end of $t_{\max}$, all facts are assigned. Hence, the value $W_{t_{max}}$ of the potential function is equal to the total weight of clauses that are satisfied by our algorithm. Let $\mathcal{C}_A$ be the set of clauses satisfied by the assignment computed by the algorithm. The following inequality holds:

$$\sum_{C \in \mathcal{C}_A} w(C) = W_{t_{\max}} \geq W_0 \geq p \cdot OPT. \tag{5.8}$$

□ As in a general Horn Boolean formula, every clause (with size larger than one) has at least one negated literal, we obtain the following result as a corollary.

Corollary 1 *Our algorithm has an approximation guarantee of* $1/2$ *for general Horn formulas.*

As for formulas with many negated literals, we are not aware of any closed form formula to express the solutions of the equation $p = 1 - p^\lambda$ as a function of λ. In the case $\lambda = 2$ we obtain $p = \sqrt{5}/2 - 1/2 \approx 0.618$, while in the case $\lambda = 3$ we obtain a ratio of roughly 0.68. The approximation guarantee of our algorithm approaches one (the optimum solution) as λ increases. Another parameter which determines the approximation guarantee of our algorithm is the ratio between the largest and second largest weight of facts in any competitor set. This is formalized as follows:

Theorem 2 *Given a competitor set $S_k \in \mathcal{S}$. Let τ_k be the ratio between $w(f_1)$ and $w(f_2)$, which are the facts with largest and second largest weight in S_k, respectively. Let τ be the minimum among all τ_k, and let λ be the minimum number of negated literals in any Horn clause $C \in \mathcal{C}$. The approximation guarantee of our algorithm is p such that $p = 1 - \left(\frac{p\,\tau}{\tau+1}\right)^\lambda$.*

The proof is very similar to the proof of Theorem 1. Also for the solution of the equation used in Theorem 2, we are not aware of any closed form formula. An interesting case is when $\tau = 1$ and $\lambda = 2$, for which we obtain an approximation guarantee of 0.83.

Our experiments indicate that the MAX-SAT setting is fairly stable over the dependency graphs and type of inferencing we considered in Section 6, i.e., often the optimum is indeed reached by setting the fact with the highest weight $w(f_i)$ (see Equation 5.4) in a competitor set to *true*. Even forcing the algorithm to choose a lower-weighted fact in one competitor set typically does not affect the majority of facts in the remaining part of the dependency graph.

6 Experiments

URDF is implemented as a fairly compact Java prototype in about 3,000 lines of code. All experiments were run on an AMD Opteron Quad-Core 2.6 GHz server with 16 GB RAM, using Oracle 11g as storage back-end for the underlying knowledge base. Physical I/O's were cached (thus aiming to eliminate variances due to disk operations) by running each query once and then taking the average runtime over 5 immediately repeated runs. Memory consumption was generally not a delimiting factor, with up to 501 MB overall memory consumption for our Java implementation (including the high overhead of the Java VM) and less than 10 MB for the Alchemy package (Subsection 6.1.2), implemented in C++.

6.1 YAGO Knowledge Base

For our experiments, we used the semantic graph of YAGO [32], which is a knowledge base that has automatically been extracted from Wikipedia and WordNet [15]. It knows 2 million entities and contains 19 million facts about them. The facts include a class hierarchy of 200,000 classes with about 100 distinct relationships. These include for example biographic relationships such as *bornIn* and *bornOnDate*, geographic information such as *hasCapital* and *locatedIn*, and information about movies such as *actedIn* and *hasProductionLanguage*. YAGO comes with precomputed confidence weights for base facts in the range of $[0, 1]$.

6.1.1 Rules and Queries

Table A.1 depicts 16, partially mutually recursive, hand-crafted soft rules for common-sense reasoning about people and their relationships, as well as a few movie/actor-specific reasoning rules (with both domains being strengths of YAGO). Table A.1 depicts the 10 real-world queries (following typical patterns: single-fact queries, as well as chains, stars and cliques of interconnected facts) that serve as our baseline for the following experiments. Moreover, we employ the following

hard rules (in their non-grounded form):

- $S_1 = \{$*bornIn(person, x)*$\}$
- $S_2 = \{$*bornOnDate(person, x)*$\}$
- $S_3 = \{$*diedIn(person, x)*$\}$
- $S_4 = \{$*diedOnDate(person, x)*$\}$
- $S_5 = \{$*marriedTo(person, x)*$\}$

Hard rules are grounded by binding the *person* variable to the given query context and leaving the right-hand variable unbound (see Algorithm 2). Note that for the *marriedTo* predicate, we ignore temporal aspects for simplicity.

So is for example query Q_1: *livesIn(Al_Gore,x)* interesting, because YAGO does not contain any information about where *Al Gore* actually lives. From rule C_{11}, however, we conclude that he may live in *Washington D.C.*, as both he and his wife *Tipper Gore* were born in *Washington D.C.* (which is indeed captured by YAGO). C_{11} shows that also rules with a weight of 0 are allowed, since there is now grounding (evidence) for the entire conjunction of these atoms in YAGO. Still the algorithm assigns *true* to the fact *livesIn(Al_Gore, Washington_D.C.)* which is correctly inferred from C_{11}. Moreover, Q_2: *marriedTo(Woody_Allen,x)* has a nice twist because *Woody Allen* has been married to his step-daughter *Soon-Yi Previn*, although this contradicts rule C_{13}, which indicates that no person is married to his or her daughter. Moreover, according to rule C_{16}, *Woody Allen* might also be married to *Sharon Stone* and *Hugh Grant* because he co-acted together with them in a movie. The MAX-SAT setting however correctly yields *Soon-Yi Previn* as the only answer (out of these mutually exclusive alternatives) that is assigned *true* because the weight for C_{16} is very low.

Note that the query predicates overlap with many head predicates in the soft rules, thus creating a recursion depth of up to 7 in our inferencing algorithm (Algorithms 1 and 2) when grounding the queries. Moreover, for this particular setting, with all soft rules consisting of at least two negated literals, our algorithm can be shown to provide a 0.68 approximation guarantee for finding the MAX-SAT optimum (see Subsection 5.2).

6.1.2 Markov Logic: MAP Inference and MC-SAT

The Alchemy package[1] provides a series of algorithms for statistical relational learning and probabilistic logic inference, based on the Markov Logic [28] representation. It implements various MCMC sampling techniques, including MAP inference [31] (a state-of-the-art, memory-efficient MAX-SAT solver) and MC-SAT [27]. MAP inference yields truth assignments to facts (which allows for

[1] `http://alchemy.cs.washington.edu/`

precision comparisons with URDF), whereas MC-SAT computes probability distributions over the underlying Markov network (which URDF does not do). Thus, we merely refer to MC-SAT for runtime comparisons as a state-of-art technique for MCMC sampling. Markov Logic (and hence Alchemy) only allows for simulating hard constraints by automatically assigning high weights to the respective clauses, whereas the URDF MAX-SAT solver inherently excludes multiple *true* assignments to facts within the same competitor set.

The `MAXSTEPS` parameter in Alchemy was reduced from the default value of 1,000 to 100 for faster inference, and both the `lazy` and `lifted` options were enabled. For all the following runs, we directly fed grounded formulas into Alchemy, i.e., all approaches operated over exactly the same input after the grounding.

6.1.3 YAGO Results

The first setting reports running times and result precision for the URDF reasoner compared to MAP inference and MC-SAT over YAGO, with the rules and queries depicted in Tables A.1 and A.2. Note that in this baseline setting, it is already infeasible to compute exact MAX-SAT solutions, with queries yielding up to 310 facts in the dependency graph $\mathcal{D}$, depicted as $|\mathcal{D}|$ in Table 6.1. As for approximation quality, we measure the *relative precision* of the URDF MAX-SAT solver compared to the MAP baseline: if the MAX-SAT weight computed by URDF (*URDF-W*) is greater or equal to the weight achieved by MAP inference (*MAP-W*), and none of the hard constraints are violated by either approach, we conclude that we found an equally good or even better solution. Grounding time (*SLD*) denotes the time needed to ground the query atoms, soft and hard rules, and to expand the dependency graph via a top-down SLD resolution of the non-grounded rules and queries depicted in Tables A.1 and A.2. This step conforms Lines 1–5 in Algorithm 1 and is used as input to both the URDF MAX-SAT solver and the Alchemy package. Running times for MAP and MC-SAT include only the actual running times reported by Alchemy, but not the time needed by our interface, i.e., for dumping the grounded formulas to temporary files. $|\mathcal{D}|$ denotes the size of the dependency graph, $\#\mathcal{C}$ and $\#\mathcal{S}$ the number of grounded soft and hard rules (including unit clauses), and $|\mathcal{C}|$ and $|\mathcal{S}|$ the number of occurrences of facts in all grounded soft and hard rules, respectively. Since the hard rules form a strict partitioning of the dependency graph, $|\mathcal{D}| = |\mathcal{S}|$ for all queries. That is, the actual query response time for URDF is the sum of the SLD grounding and the MAX-SAT solving times. The same holds for the two competitors, MAP and MC-SAT.

Table 6.1 shows that the URDF MAX-SAT solver achieves more than *two orders of magnitude* runtime improvement over MAP and MC-SAT, at 90 percent

	$\lvert\mathcal{D}\rvert$	$\#\mathcal{C}$	$\lvert\mathcal{C}\rvert$	$\#\mathcal{S}$	$\lvert\mathcal{S}\rvert$	URDF/MLN SLD (ms)	URDF MAX-SAT (ms)	MLN MAP (ms)	MLN MC-SAT (ms)	URDF MAX-SAT Weight	MLN MAP Weight
Q_1	25	49	109	22	25	243	1	80	65	19.92	19.92
Q_2	12	14	20	10	12	53	1	814	17	9.69	9.69
Q_3	28	32	40	27	28	25	7	814	17	20.56	20.56
Q_4	30	46	82	23	30	178	1	861	221	20.77	20.77
Q_5	167	176	203	167	167	3,062	13	1,564	60,970	161.93	161.93
Q_6	310	318	342	307	310	584	4	111	173	292.40	292.40
Q_7	48	100	220	41	48	222	4	1,344	1,032	**27.06**	**27.91**
Q_8	193	195	199	192	193	93	7	1,877	36,330	188.21	188.21
Q_9	35	44	71	27	35	143	7	1,407	142	26.37	26.37
Q_{10}	89	89	89	89	89	71	4	283	5,267	86.83	86.83
$\sum$	937	1,063	1,375	905	937	4,674	49	9,155	104,234	853.74	854.58

Table 6.1: Basic query processing results on YAGO, for the soft rules of Table A.1 and queries of Table A.2.

precision compared to the MAP baseline. That is, for 9 out of the 10 queries URDF finds the same MAX-SAT weight as MAP, while only for query Q_7, the weight returned by URDF is marginally worse. We also see that running MC-SAT is generally more expensive than MAP inference. In this basic setting, SLD grounding clearly dominates query response times for URDF.

Asymptotical Behavior

To systematically investigate the asymptotical behavior of our algorithm for large dependency graphs and more complex rules, we employ synthetic expansions over the basic setting. We thus simulate very complex CNF formulas and competitor constraints with more than 10^5 facts by synthetic fact expansions of C and S. In the following plots, the grounding time also includes the time for expanding the CNF formulas with these noisy facts and thus is not constant for all runs. Furthermore, we will from now on only refer to aggregated metrics over the above 10 queries.

In the first synthetic setting, we start with the grounded soft and hard rules obtained by the setting described in Subsection 6.1.3 and expand the soft rules by noisy ("dummy") facts with a weight of 0. That is, we expand each original grounded soft rule by *10 distinct facts* as noise per step, for each of the query results depicted in Table 6.1. Here, we do not yet change the hard rules or the amount of non-unary soft rules. This setting conforms to a simplest-possible expansion strategy which merely serves to increase the complexity of the CNF formula for the MAX-SAT solver without changing the MAX-SAT optimum, which allows us to further keep track of the result precision for large expansions with $|\mathcal{C}| + |\mathcal{S}|$ ranging from 2,312 to 36,332. The x-axes in Figures 6.1 and 6.2 denote the sum of the occurrences of facts in both the soft and hard rules, i.e., the size of

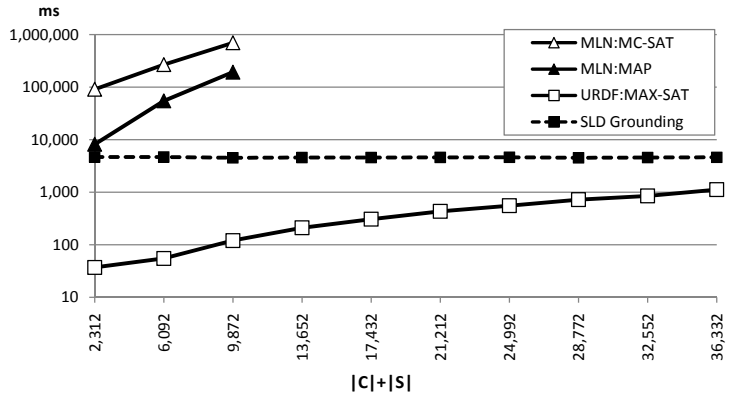

Figure 6.1: URDF vs. MAP and MC-SAT for soft rule expansions (summed over all 10 queries in Table A.2).

all constraints involved, on a linear scale. The y-axis of Figure 6.1 displays the running time in milliseconds of the SLD grounding and MAX-SAT solving steps for URDF, MAP and MC-SAT, using a logarithmic scale due to their huge runtime differences. URDF-MAX-SAT-0 in Figure 6.2 shows a more detailed plot for even larger expansions (with $|\mathcal{C}| + |\mathcal{S}|$ ranging from 2,312 to 342,512), showing the runtime needed by URDF on a linear scale. For all these runs, the measured precision of the URDF MAX-SAT solver remains at 90 percent, i.e, the same as depicted in Table 6.1.

We next investigate uniform weights and weights following a Gaussian distribution (with σ^2=1) around the original weight of the YAGO fact. Uniform weights

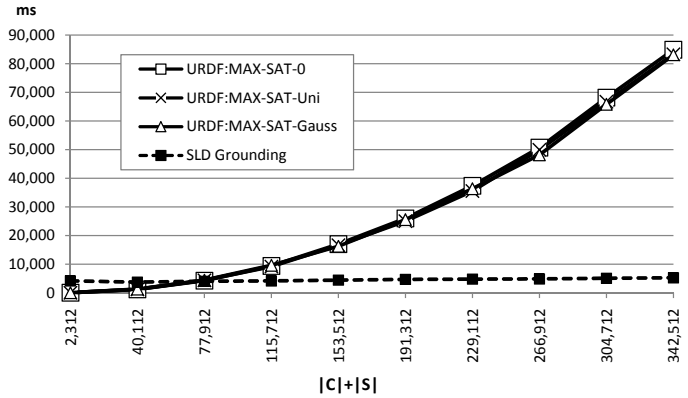

Figure 6.2: URDF MAX-SAT with different weighting schemes (summed over all 10 queries in Table A.2).

can be thought of as a worst case for a weighted MAX-SAT solver, i.e., when many expanded facts have the same weight as the original fact. As shown in Fig-

ure 6.2, the runtime of our algorithm is however hardly affected by the weighting schemes and remains almost equally efficient.

Expanding Soft and Hard Rules

We now introduce noisy facts in both the soft and hard rules over our original setting. This time, for each original fact in the body of a soft rule, we introduce *10 mutually exclusive, distinct facts* as noise per step. We keep a Gaussian distribution of weights for the noisy facts. Since this may change the MAX-SAT optimum, we measure precision in comparison to MAP for moderate expansion sizes only. We observe that runtime increases much more moderately for URDF

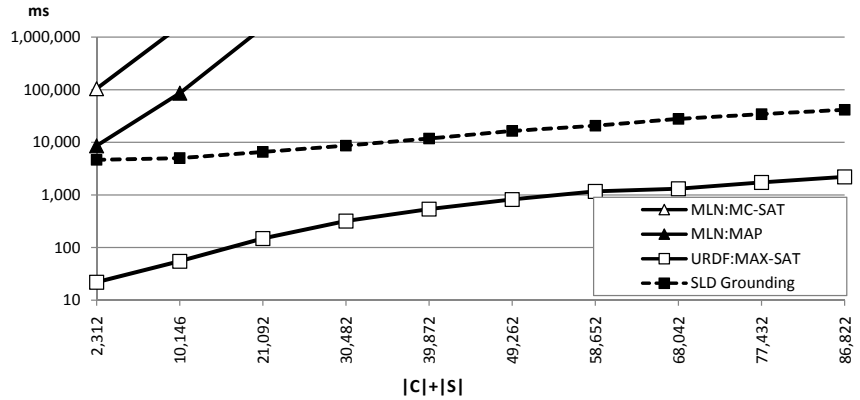

Figure 6.3: URDF vs. MAP and MC-SAT for soft and hard rule expansions (summed over all 10 queries in Table A.2).

than for MAP and MC-SAT when compared to the previous setting without hard rule expansions. Precision (still inferred from the resulting MAX-SAT weights) this time even increases for URDF in comparison to MAP, as URDF partly even yields higher MAX-SAT weights than MAP (see also Subsection 6.1.3).

Varying the Number of Soft Rules

Next we do not only vary the number of facts per soft and hard rule, but also the number of soft rules by replicating rules with different combinations of noisy facts. That is, for each original grounded fact, we introduce *10 mutually exclusive facts* as noise and, for each soft rule, we expand the CNF by introducing *10 randomly expanded soft rules* at each step. Overlap among soft rules is achieved by uniformly-randomly drawing the noisy facts from a static set of 1,000 synthetic facts for all expansions. We keep Gaussian weights for facts. Again, both the fact and rule expansions may affect the MAX-SAT optimum.

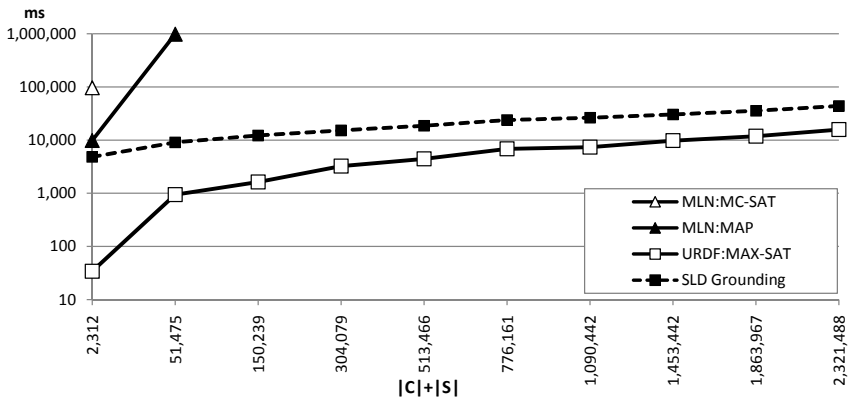

Figure 6.4: URDF vs. MAP and MC-SAT for varying numbers of soft rules (summed over all 10 queries in Table A.2).

$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{S}\rvert$	URDF/MLN SLD (sec)	URDF MAX-SAT (sec)	MLN MAP (sec)	URDF MAX-SAT Weight	MLN MAP Weight
1,375	937	4.84	0.03	9.83	**853.74**	**854.58**
43,396	8,079	9.06	0.94	981.17	**975.02**	**937.74**
140,578	9,661	12.17	1.63	2,174.71	**1,099.34**	**1,069.70**
294,163	9,936	15.18	3.24	n/a	1,225.44	n/a
503,530	9,937	18.63	4.43	n/a	1,351.10	n/a
766,224	9,937	23.85	6.82	n/a	1,477.60	n/a
1,080,505	9,937	26.31	7.37	n/a	1,604.10	n/a
1,443,505	9,937	30.29	9.76	n/a	1,730.59	n/a
1,854,030	9,937	35.44	11.80	n/a	1,857.08	n/a
2,311,551	9,937	43.62	15.77	n/a	1,983.58	n/a

Table 6.2: URDF vs. MAP for very large rule expansions (summed over all 10 queries in Table A.2).

This setting yields extremely large CNF formulas with $|\mathcal{C}| + |\mathcal{S}|$ ranging from 2,312 to 2,321,488. More specifically, we create constraints with up to 21,277 soft rules and 2,311,551 occurrences of facts in all soft rules, over an overall amount of only 9,937 distinct facts for the last expansion step in Figure 6.4. In that step, each fact on average occurs in more than 232 rules, each with an average rule length of 108 facts. URDF solves the CNF formulas for this expansion in 15.7 seconds, while remaining at an even higher precision in computing the MAX-SAT weight than MAP. Approximation precision is measured only for the first three expansion steps, yielding MAX-SAT weights of 853.74, 975.02, $1,099.34$ for URDF and 854.58, 937.74, $1,069.70$ for MAP, respectively, (see also Table 6.2). MAP does not scale up to larger CNFs than in these first three expansions, while MC-SAT cannot be run beyond the first expansion step anymore.

6.2 Lehigh University Benchmark (LUBM)

In addition to the YAGO knowledge base, we also carried out performance evaluations using the Lehigh University benchmark (LUBM) [20]. LUBM generates synthetic data sets that model the university domain, and it comes with 14 test queries. Three data sets were generated, LUBM(1), LUBM(5), LUBM(10) featuring scale factors of 1, 5 and 10 universities, consisting of 101,001, 624,766 and 1,273,448 base facts, respectively. As opposed to YAGO, LUBM data has no potential inconsistencies or mutually recursive inferencing rules. Furthermore, a large part of the inferencing functionalities supported by OWL-lite is not covered by LUBM. For example, LUBM does not use the *functional property* which could serve as basis for hard rules in URDF. Thus, the LUBM setting was defined as soft rules with uniform weights. The 14 benchmark queries were phrased as conjunctive queries. Of the 14 benchmark queries only queries 11, 12 and 13 require OWL reasoning.

The first reasoning query, LUBMQ_{11}, requires the system to reason using the *transistive property* of OWL. The query asks for all instances of *ResearchGroup* that are *subOrganizationOf* a given university. In the benchmark data, instances of *ResearchGroup* are sub-organizations of *Department* entities and departments are sub-organizations of university entities. The second reasoning query, LUBMQ_{12}, asks for all instances of *Chair* for all departments of a given university. The data however has no instances of department chairs. Instead the ontology defines that a *Professor* who is the *headOf* a given *Department* is also the *Chair* of that *Department*. Hence the system needs to infer about the relation equivalence between people being *Chair* and the relation *headOf* in order to answer this question. The last reasoning query, LUBMQ_{13}, asks for all instances of the relation *hasAlumnus* for a given university. The data has no explicit instances of this relation. Instead the ontology defines the *hasAlumnus* relation as the inverse of the *degreeFrom* relation. Furthermore, the data has no explicit instances of the *degreeFrom* relation. Instead, it is a composite relation whose instances can be inferred from three relations *undergraduateDegreeFrom*, *mastersDegreeFrom*, *doctoralDegreeFrom*.

Query processing results are summarized in Table 6.3 in comparison to the Jena Semantic Web toolkit[2] using PostgreSQL 8.3 as back-end. Performance for URDF is similar to that observed on YAGO, where MAX-SAT times are again much faster than SLD grounding times. URDF outperforms Jena even in the grounding step by a large margin, while Jena failed to respond to many queries at increasing scale factors (compare also to results of [20]).

[2] `http://jena.sourceforge.net/`

	LUBM(1)			
	$\lvert\mathcal{C}\rvert + \lvert\mathcal{S}\rvert$	URDF-SLD (ms)	URDF-MAX-SAT (ms)	Jena-OWL (ms)
LUBMQ_1	8	3	4	48,375
LUBMQ_2	7,526	2,028	35	7,375
LUBMQ_3	12	3	1	3,766
LUBMQ_4	490	109	1	474,797
LUBMQ_5	4,316	53	13	20,609
LUBMQ_6	46,740	543	85	313
LUBMQ_7	46,882	11,378	98	10,422
LUBMQ_8	34,326	3,775	113	304,625
LUBMQ_9	68,174	7,468	179	6,156
LUBMQ_{10}	8	1	13	3,734
LUBMQ_{11}	**2,718**	**281**	**4**	**40,235**
LUBMQ_{12}	**2,390**	**293**	**1**	**250**
LUBMQ_{13}	**10**	**3**	**1**	**22,610**
LUBMQ_{14}	11,832	187	38	250

	LUBM(5)			
	$\lvert\mathcal{C}\rvert + \lvert\mathcal{S}\rvert$	URDF-SLD (ms)	URDF-MAX-SAT (ms)	Jena-OWL (ms)
LUBMQ_1	8	9	1	308,751
LUBMQ_2	47,804	15,509	192	45,938
LUBMQ_3	12	3	1	27,234
LUBMQ_4	490	153	1	n/a
LUBMQ_5	4,316	62	7	8,527,910
LUBMQ_6	291,490	3,850	1,400	516
LUBMQ_7	291,632	91,897	929	68,203
LUBMQ_8	34,326	4,987	116	n/a
LUBMQ_9	425,036	62,462	2,229	310,468
LUBMQ_{10}	8	1	1	n/a
LUBMQ_{11}	**2,718**	**281**	**7**	**500,047**
LUBMQ_{12}	**2,390**	**387**	**7**	**n/a**
LUBMQ_{13}	**214**	**6**	**1**	**n/a**
LUBMQ_{14}	73,364	1,415	304	11,125

	LUBM(10)			
	$\lvert\mathcal{C}\rvert + \lvert\mathcal{S}\rvert$	URDF-SLD (ms)	URDF-MAX-SAT (ms)	Jena-OWL (ms)
LUBMQ_1	8	9	1	680,179
LUBMQ_2	96,510	31,727	498	109,220
LUBMQ_3	12	1	1	57,047
LUBMQ_4	490	165	1	n/a
LUBMQ_5	4,316	68	13	n/a
LUBMQ_6	597,390	8,184	4,091	n/a
LUBMQ_7	597,532	197,712	1,601	673,609
LUBMQ_8	34,326	5,121	116	n/a
LUBMQ_9	867,674	129,031	4,454	692,282
LUBMQ_{10}	8	1	1	n/a
LUBMQ_{11}	**2,718**	**271**	**13**	**512,250**
LUBMQ_{12}	**2,390**	**378**	**4**	**n/a**
LUBMQ_{13}	**334**	**12**	**1**	**n/a**
LUBMQ_{14}	151,094	2,884	538	283,313

Table 6.3: URDF grounded rule sizes ($|\mathcal{C}| + |\mathcal{S}|$) and runtimes in ms (SLD and MAX-SAT) vs. Jena runtimes in ms on LUBM.

7 Conclusions

We presented URDF, an efficient first-order reasoning framework for RDF data and SPARQL-like queries with soft and hard rules. URDF can *dynamically* resolve inconsistencies between hard constraints and the underlying knowledge base at query time. Our experiments confirm that our MAX-SAT approximation algorithm scales to many thousands of rules and facts, while empirically performing much better than the provided lower bound of 1/2 for the MAX-SAT approximation guarantee. We also saw that in many cases the grounding time for the Datalog-style soft inferencing rules is the actual delimiting factor for query response times. Our future work will thus investigate further speeding up the inferencing techniques we employ, for example, by using a combination of dynamic inference over highly transient parts of the knowledge base with precomputed resolution steps for the more static areas, as well as by investigating the parallelization of our inferencing strategies.

Appendix A APPENDIX

People usually do not live longer than 85 years.
C_1: *bornOnDate(a,b)* $\wedge$ *diedOnDate(a,c)* $\rightarrow$ *(c* $-$ *b* $\leq$ *85)* $_{[165847/192435]}$

People born before 1900 typically did not live longer than 65 years.
C_2: *bornOnDate(a,b)* $\wedge$ *yearBefore(b,1900)* $\wedge$ *diedOnDate(a,c)* $\rightarrow$ *(c* $-$ *b* $\leq$ *65)* $_{[48594/124999]}$

Everybody is born before his/her parent died.
C_3: *hasChild(a,b)* $\wedge$ *bornOnDate(b,c)* $\wedge$ *diedOnDate(a,d)* $\rightarrow$ *(c* $<$ *d)* $_{[2715/2793]}$

If somebody was born in a place and lived in that place, he/she also died in that place.
C_4: *bornIn(a,b)* $\wedge$ *livesIn(a,c)* $\rightarrow$ *diedIn(a,c)* $_{[77/588]}$

If somebody is a citizen of a country, he/she was also born in that country.
C_5: *isCitizenOf(a,b)* $\wedge$ *locatedIn(b,c)* $\rightarrow$ *bornIn(a,c)* $_{[4/51]}$

If somebody graduated from a university, he/she was born after the university was established.
C_6: *graduatedFrom(a,b)* $\wedge$ *establishedOnDate(b,c)* $\wedge$ *bornOnDate(a,d)* $\rightarrow$ *(c* $<$ *d)* $_{[4528/4791]}$

If somebody has an academic advisor who works at a university, then he/she graduates from that university.
C_7: *hasAcademicAdvisor(a,b)* $\wedge$ *worksAt(b,c)* $\rightarrow$ *graduatedFrom(a,c)* $_{[239/679]}$

If two people have the same academic advisor, they also graduate from the same university.
C_8: *hasAcademicAdvisor(a,c)* $\wedge$ *hasAcademicAdvisor(b,c)* $\wedge$ *graduatedFrom(a,d)* $\wedge$ *(a* $\neq$ *b)* $\rightarrow$ *graduatedFrom(b,d)* $_{[1072/2130]}$

If two people are married, they also live in the same place.
C_9: *marriedTo(a,b)* $\wedge$ *livesIn(a,c)* $\rightarrow$ *livesIn(b,c)* $_{[37/843]}$

If two people are married and one was born in some place, then the other person was also born in that place.
C_{10}: *marriedTo(a,b)* $\wedge$ *bornIn(a,c)* $\rightarrow$ *bornIn(b,c)* $_{[59/589]}$

If two people are married and were both born in the same place, then one of them also lives in that place.
C_{11}: *marriedTo(a,b)* $\wedge$ *bornIn(a,c)* $\wedge$ *bornIn(b,c)* $\rightarrow$ *livesIn(a,c)* $_{[0/59]}$

If two different people have a child in common, then they are married.
C_{12}: *hasChild(a,c)* $\wedge$ *hasChild(b,c)* $\wedge$ *(a* $\neq$ *b)* $\rightarrow$ *marriedTo(a,b)* $_{[820/2172]}$

People are not married to their children.
C_{13}: *hasChild(a,b)* $\wedge$ *marriedTo(a,c)* $\rightarrow$ *(b* $\neq$ *c)* $_{[1242/1251]}$

If two people are married, live in the same place and have a child in common, then the child was born in the same place.
C_{14}: *marriedTo(a,b)* $\wedge$ *livesIn(a,c)* $\wedge$ *livesIn(b,c)hasChild(a,d)* $\wedge$ *hasChild(b,d)* $\rightarrow$ *bornIn(d,c)* $_{[1/4]}$

If somebody is director of a movie, he/she is unlikely to be an actor in the same movie.
C_{15}: *directorOfMovie(a,b)* $\wedge$ *actedInMovie(c,b)* $\rightarrow$ *(a* $\neq$ *c)* $_{[16304/16626]}$

If two different people acted together in the same movie, then they might be married.
C_{16}: *actedInMovie(a,c)* $\wedge$ *actedInMovie(b,c)* $\wedge$ *(a* $\neq$ *b)* $\rightarrow$ *marriedTo(a,b)* $_{[17/10346]}$

Table A.1: Set of 16 hand-crafted soft rules used in the experiments.

Single-fact queries:

Where is Al Gore born?
Q_1: *bornIn(Al_Gore, x)*

Who is Woody Allen married to?
Q_2: *marriedTo(Woody_Allen, x)*

Chain queries:

Who acted in the movie Total Recall and was born in Thal, Austria?
Q_3: *actedInMovie(x, Total_Recall)* $\wedge$ *bornIn(x, Thal_Austria)*

Who acted together with Arnold Schwarzenegger in the same movie, and where were they born?
Q_4: *actedInMovie(Arnold_Schwarzenegger, y)* $\wedge$ *actedInMovie(x, y)* $\wedge$ *bornIn(x,z)*

Who was born in Oxford, where did he or she graduate from, and who was his/her academic advisor who himself had graduated from the University of Cambridge?
Q_5: *bornIn(x, Oxford)* $\wedge$ *graduatedFrom(x,y)* $\wedge$ *hasAcademicAdvisor(x,z)* $\wedge$ *graduatedFrom(z, University_of_Cambridge)*

Ring query:

Who was born in Paris and is citizen of a country where Paris is not located in?
Q_6: *bornIn(x, Paris)* $\wedge$ *isCitizenOf(x, y)* $\wedge$ *isLocatedIn(Paris, z)* $\wedge$ $(y \neq z)$

Star queries:

Who acted in the movie Totall Recall, where was he/she born, where does he/she live, when was he/she born, and who is he/she married to?
Q_7: *actedInMovie(x, Totall_Recall)* $\wedge$ *bornIn(x, y)* $\wedge$ *livesIn(x, z)* $\wedge$ *bornOnDate(x, a)* $\wedge$ *marriedTo(x, b)*

Who has won the Nobel prize in physics, was born in Ulm, was born before 1900, and originally was a patent examiner?
Q_8: *hasWonPrize(x,Nobel_Prize_in_Physics)* $\wedge$ *bornIn(x, Ulm)* $\wedge$ *bornOnDate(x,y)* $\wedge$ $(y < 1900)$ $\wedge$ *worksAs(x, Patent_Examiner)*

Clique queries:

Who is Emma Thompson married to, and in which movies did she act in which also here spouse acted?
Q_9: *marriedTo(Emma_Thompson,x)* $\wedge$ *actedInMovie(Emma_Thompson, y)* $\wedge$ *actedInMovie(x,y)*

Which movies were directed by Martin Scorsese, and which two different people acted in this movie?

Q_{10}: *directorOfMovie(Martin_Scorsese,x)* $\wedge$ *actedInMovie(y,x)* $\wedge$ *actedInMovie(z,x)* $\wedge$ $(y \neq z)$ $\wedge$ $(y \neq$ *Martin_Scorsese)* $\wedge$

$(z \neq$ *Martin_Scorsese)*

Table A.2: 10 real-world queries of different shapes used as baseline for the experiments.

Bibliography

[1] RDF Primer (W3C Rec.2004-02-10) & RDF Schema. `http://www.w3.org/TR/rdf-primer/` `http://www.w3.org/TR/rdf-schema/`.

[2] D. J. Abadi, A. M. 0002, S. Madden, and K. J. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, pages 411–422, 2007.

[3] N. Alon and J. Spencer. *The Probabilistic Method*. John Wiley, 1992.

[4] G. Antoniou and F. van Harmelen. *A Semantic Web Primer (Cooperative Information Systems)*. The MIT Press, April 2004.

[5] L. Antova, C. Koch, and D. Olteanu. MayBMS: Managing incomplete information with probabilistic world-set decompositions. In *ICDE*, pages 1479–1480, 2007.

[6] K. R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *J. ACM*, 29(3):841–862, 1982.

[7] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A Nucleus for a Web of Open Data. In *ISWC*, pages 722–735, 2007.

[8] P. Baumgartner and F. M. Suchanek. Automated reasoning support for first-order ontologies. In *PPSWR*, volume 4187 of *LNAI*. Springer, 2006.

[9] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[10] H. C. Bravo and R. Ramakrishnan. Optimizing MPF queries: decision support and probabilistic inference. In *SIGMOD*, pages 701–712, 2007.

[11] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[12] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[13] N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.

[14] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in KnowItAll. In *WWW*, pages 100–110, 2004.

[15] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[16] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[17] L. Getoor and B. Taskar. *An Introduction to Statistical Relational Learning*. MIT Press, 2007.

[18] W. Gilks, S. Richardson, and D. J. S. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996.

[19] M. X. Goemans and D. P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM J. Discrete Math.*, 7(4):656–666, 1994.

[20] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, October 2005.

[21] B. Jaumard and B. Simeone. On the complexity of the maximum satisfiability problem for Horn formulas. *Information Processing Letters*, 26(1):1 – 4, 1987.

[22] A. Jha, V. Rastogi, and D. Suciu. Query evaluation with soft-key constraints. In *PODS*, pages 119–128, 2008.

[23] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.

[24] T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. *PVLDB*, 1(1):647–659, 2008.

[25] I. Niles and A. Pease. Towards a standard upper ontology. In *FOIS*, pages 2–9, New York, NY, USA, 2001. ACM.

[26] H. Nottelmann and N. Fuhr. Adding probabilities and rules to OWL lite subsets based on probabilistic Datalog. *Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 14(1):17–41, 2006.

[27] H. Poon, P. Domingos, and M. Sumner. A general method for reducing the complexity of relational inference and its application to MCMC. In *AAAI*, pages 1075–1080, 2008.

[28] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2), 2006.

[29] S. Russel and P. Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, 2002.

[30] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, pages 596–605, 2007.

[31] P. Singla and P. Domingos. Memory-efficient inference in relational domains. In *AAAI*, 2006.

[32] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A core of semantic knowledge - unifying WordNet and Wikipedia. In *WWW*, pages 697–706. ACM, 2007.

[33] T. Tran, C. Sutton, R. Cocci, Y. Nie, Y. Diao, and P. Shenoy. Probabilistic inference over RFID streams in mobile environments. In *ICDE*, pages 1096–1107, 2009.

[34] D. S. Weld, F. Wu, E. Adar, S. Amershi, J. Fogarty, R. Hoffmann, K. Patel, and M. Skinner. Intelligence in Wikipedia. In *AAAI*, pages 1609–1614, 2008.

[35] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.