Learning to Reason with a Scalable Probabilistic Logic

William Yang Wang

CMU-LTI-16-007

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:
Prof. William W. Cohen (Chair), Carnegie Mellon University
Prof. Tom M. Mitchell, Carnegie Mellon University
Prof. Christos Faloutsos, Carnegie Mellon University
Dr. Eric Horvitz, Microsoft Research

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Language and Information Technologies.

Keywords: Probabilistic logics, statistical relational learning, personalized PageRank.

For my parents.

Abstract

Learning to reason and understand the world's knowledge is a fundamental problem in Artificial Intelligence (AI). Traditional symbolic AI methods were popular in the 1980s, when first-order logic rules were mostly handwritten, and reasoning algorithms were built on top of them. In the 90s, more and more researchers became interested in statistical methods that deal with the uncertainty of the data, using probabilistic models.

While it is always hypothesized that both the symbolic and statistical approaches are necessary for building intelligent systems, in practice, bridging the two in a combined framework often brings intractability—most probabilistic first-order logics are simply not efficient enough for real-world sized tasks. For example, Markov Logics [83] integrate first-order logics with Markov random field theory, but when mapping the entities in a knowledge base (KB) to the propositional theory (i.e., grounding), the size of the network depends on the number of facts in the KB—i.e., $O(n^k)$ where k is the arity of the predicate, and n is the number of KB constants.

In this thesis, we design a new probabilistic logic programming paradigm to address various scalability issues in probabilistic logics. We propose a group of scalable methods for inference, learning, and inducing the structure of probabilistic logics. More specifically, we propose a scalable probabilistic logic called ProPPR [105] to combine the best of the symbolic and statistical worlds. ProPPR can be viewed as a probabilistic version of Prolog, and we associate a feature vector for each clause to learn weights from data. The learned weights are used to control search during inference. ProPPR's inference scheme is very special: instead of performing potentially intractable global inference, ProPPR uses a provably-correct approximate personalized PageRank to conduct local grounding, whose inference time is *independent of the size of the KB*. To test ProPPR for large, real-world relational learning problems, we show that it can be used as a recursive relational learning engine [108] for large scale KB inference tasks.

Another challenging problem in statistical relational learning is structure learning: learning logic programs from a KB. Prior approach can take up to a month to learn the theory for a small KB [50]. This thesis provides a scalable solution to finding inference rules: we create a higher-level of abstraction, and reduce the unknown structure learning problem to parameter learning in ProPPR. To refine the learning process, we also introduce an iterated structural gradient algorithm for pruning the search space. This thesis also connects structured sparsity in machine learning and predicate invention in inductive logic programming [107]. To improve structure learning and incorporate latent variable modeling, we have also designed a rather radical approach—learning low-dimensional continuous latent embeddings of logical formulas [103].

Acknowledgments

First and foremost, without the guidance of my advisor *William W. Cohen*, this thesis is not possible. I am deeply indebted to you for all you have done. Thank you for introducing me to probabilistic logics, being the best advisor in the world, and making my PhD such a valuable and pleasant experience.

Secondly, I thank my additional thesis committee members *Tom M. Mitchell, Christos Faloutsos, and Eric Horvitz* for insightful feedback and comments. It is my honor to work with such an all-star committee, and I have learned so many lessons from you. I am also very grateful to *Kathryn Rivard Mazaitis* for her fundamental contribution to ProPPR.

Throughout my graduate school, I am very thankful to all my friends, colleagues, and collaborators for useful discussions about my work and encouragement: Fatima Al-Raisi, Jun Araki, Waleed Ammar, Lidong Bing, Alan W. Black, Dan Bohus, Leonid Boytsov, Justine Cassell, Xinlei Chen, Yun-Nung Chen, Bhavana Dalvi Mishra, Dipanjan Das, Bhuwan Dhingra, Duo Ding, Jeremiah Dittmar, Christopher Dyer, Manaal Faruqui, Samantha Finkelstein, Robert Frederking, Kevin Gimpel, Malcolm Greaves, Zhenhao Hua, Ting-Hao Huang, Eduard Hovy, Hyeju Jang, Ece Kamar, Rose Kanjirathinkal, John Kominek, Lingpeng Kong, Walter Lasecki, Ni Lao, Sungjin Lee, Junyi Jessy Li, Xiang Li, Chu-Cheng Lin, Frank Lin, Edward Lin, Yibin Lin, Wang Ling, Zhengzhong Liu, Matthew Marge, Luis Marujo, Elijah Mayfield, Yashar Mehdad, Einat Minkov, Seungwhan Moon, Dana Movshovitz-Attias, Prasanna Kumar Muthukumar, Suresh Naidu, Dong Nyuyen, Amy Ogan, Dragomir Radev, Aasish Pappu, Joonsuk Park, Ellie Pavlick, Alexander Rudnicky, Mrinmaya Sachan, Yanchuan Sim, Amanda Stent, Ming Sun, Lu Wang, Samuel Thomson, Yulia Tsvetkov, Di Wang, Richard C. Wang, Miaomiao Wen, Derry Tanti Wijaya, Chenyan Xiong, Xiaohua Yan, Diyi Yang, Zhilin Yang, Zi Yang, Tae Yano, Shoou-I Yu, Dani Yogatama, Ran Zhao, Yuchen Zhang, Zeyu Zheng, and Guoqing Zheng.

Above all, I would like to thank my parents and other family members for your unconditional love and constant supports no matter where I go. I thank my fiancée Miao for always being there for me, and I am very lucky to have you by my side.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Challenges in Probabilistic Logics

Probabilistic logics are useful for many important tasks [33, 61, 79, 80]; in particular, such logics would seem to be well-suited for inference with the "noisy" facts that are extracted by automated systems from unstructured web data. While some positive results have been obtained for this problem [22], most probabilistic first-order logics are not efficient enough to be used for inference on the very large broad-coverage KBs that modern information extraction systems produce [18, 95]. One key problem is that queries are typically answered by "grounding" the query—i.e., mapping it to a propositional representation, and then performing propositional inference—and for many logics, the size of the "grounding" can be extremely large for large databases. For instance, in probabilistic Datalog [33], a query is converted to a structure called an "event expression", which summarizes all possible proofs for the query against a database; in ProbLog [27] and MarkoViews [45], similar structures are created, encoded more compactly with binary decision diagrams (BDDs); in probabilistic similarity logic (PSL) [16], an intentional probabilistic program, together with a database, is converted to constraints for a convex optimization problem; and in Markov Logic Networks (MLNs) [83], queries are converted to a propositional Markov network. In all of these cases, the result of this "grounding" process can be large.

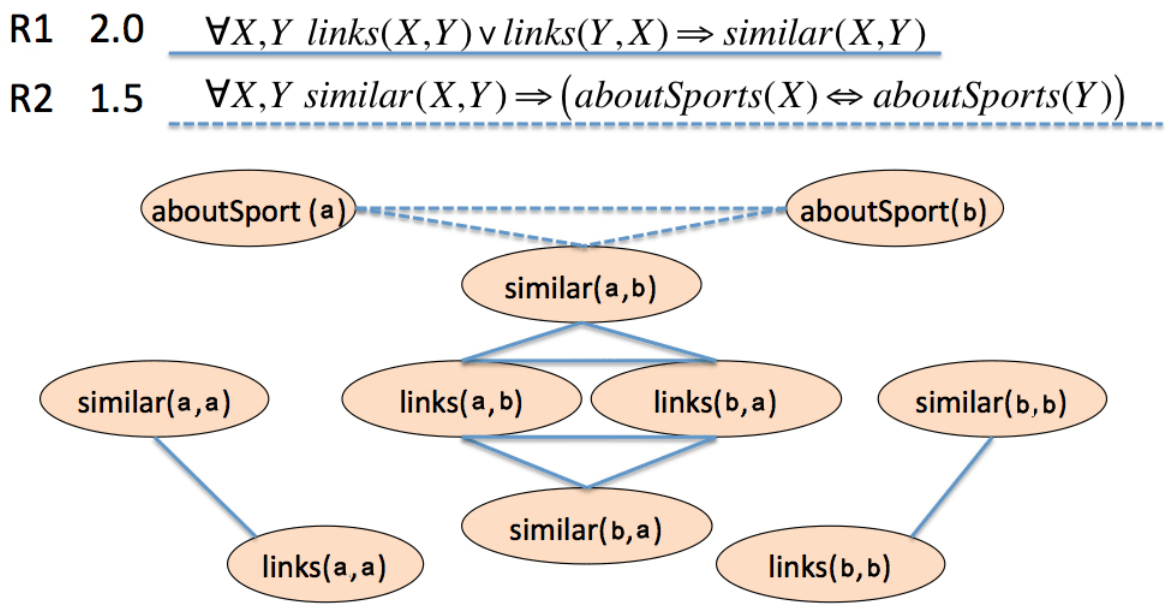


Figure 1.1: A Markov logic network program and its grounding relative to two constants a, b. (Dotted lines are clique potentials associated with rule R2, solid lines with rule R1.)

As a concrete illustration of the "grounding" process, Figure 1.1 shows a very simple MLN and its grounding over a universe of two web pages a and b. Here, the grounding is query-independent. In MLNs, the result of the grounding is a Markov network which contains one node for every atom in the Herbrand base of the program—i.e., the number of nodes is $O(n^k)$ where k is the maximal arity of a predicate and n the number of database constants. However, even a grounding size that is only linear in the number of facts in the database, $|DB|$, would be impractically large for inference on real-world problems. Superficially, it would seem that groundings must inherently be $o(|DB|)$ for some programs: in the example, for instance, the probability of *aboutSport(x)* must depend to some extent on the entire hyperlink graph, if it is fully connected. However, it also seems intuitive that if we are interested in inferring information about a specific page—say, the probability of *aboutSport(d1)*—then the parts of the network only distantly connected to *d1* are likely to have a small influence. This suggests that an *approximate* grounding strategy might be feasible, in which a query such as *aboutSport(d1)* would be

grounded by constructing a small subgraph of the full network, followed by inference on this small "locally grounded" subgraph. As another example, consider learning from a set of queries with their desired truth values. Learning might proceed by locally-grounding every query goal, allowing learning to also take less than $O(|DB|)$ time.

1.2 Thesis Statement

In this thesis, I consider a "locally grounded" probabilistic logic called ProPPR, whose inference time does not depend on the size of the knowledge base. I describe the core semantics of this new probabilistic logic, and its efficient inference and learning framework. I also show an efficient structure learning method that automatically induces logic programs from knowledge bases, as well as studies that considers structured regularization techniques for "soft predicate invention". I further discuss approaches to learning low-dimensional embeddings of logical formulas, and applications in joint information extraction and reasoning. More formally:

"Locally groundable probabilistic logics can be used to solve large, real-world relational learning problems."

1.3 Thesis Overview

The foundation of this thesis relies on a first-order probabilistic language which is well-suited to such approximate "local grounding". We describe an extension to *stochastic logic programs* (SLP) [26, 68] that is biased towards short derivations, and show that this is related to *personalized PageRank* (PPR) [20, 78] on a linearized version of the proof space. Based on the connection to PPR, we develop a provably-correct approximate inference scheme, and an associated proveably-correct approximate grounding scheme: specifically, we show that it is possible to prove a query, or to build a graph which contains the information necessary for weight-learning, in time $O(\frac{1}{\alpha\varepsilon})$, where α is a reset parameter associated with the bias towards short derivations,

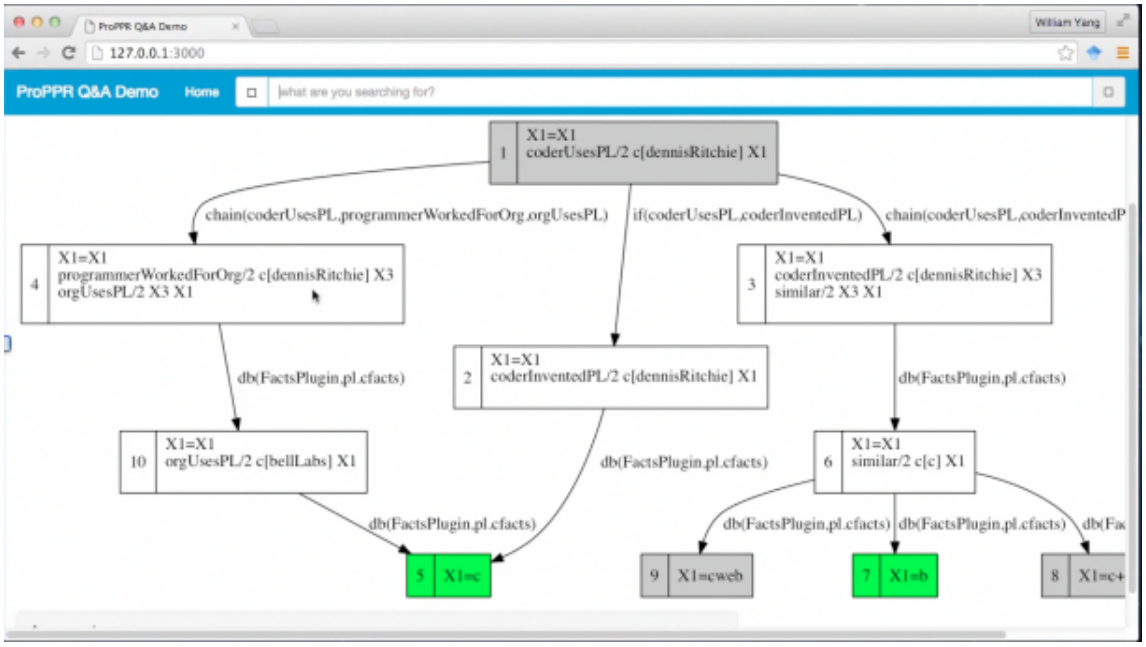


Figure 1.2: ProPPR's Web demo showing the grounding of answering queries about famous programmers and their programming languages, using an automatically learned program.

and ε is the worst-case approximation error across all intermediate stages of the proof. This means that both inference and learning can be approximated in time *independent of the size of the underlying database*—a surprising and important result, which leads to a very scalable inference algorithm. Figure 1.2 shows an example of the grounding for query answering in the domain of programming languages with an automatically learned ProPPR program, using the structure learning techniques that we are going to describe in Chapter 4.

The ability to locally ground queries has another important consequence: it is possible to *decompose* the problem of weight-learning to a number of moderate-size subtasks—in fact, tasks of size $O(\frac{1}{\alpha\varepsilon})$ or less—which are weakly coupled. Based on this we outline a parallelization scheme, which in our current implementation provides an order-of-magnitude speedup in learning time on a multi-processor machine. It is shown that ProPPR can be viewed as a recursive extension [108] to the popular path ranking algorithm (PRA) [53], and outperforms PRA on an inference task with one million facts from NELL [18], a large knowledge base with information

extracted imperfectly from the Web.

This thesis also provides a scalable solution to a core problem in statistical relational learning: finding inference rules via structure learning. For example, if we know that an athlete plays in a team, and a team plays in a league, we might want to learn an inference rule that asserts that

athletePlaysInLeague(A,L) :-

athletePlaysInTeam(A,T),teamInLeague(T,L).

To do this, we propose a second-order abductive theory [106], whose parameters correspond to plausible first-order inference rules. To improve the performance, an iterative structural gradient approach is proposed to incrementally refine the set of learned rules. In experiments, we show that the proposed approach only needs a few minutes to learn inference rules, and when using these learned rules, the predictive results improve traditional and statistical methods by a large margin. Furthermore, we also draw connections between structured sparsity in machine learning and predicate invention in inductive logic programming [107]. To improve structure learning, we have also investigated a rather radical ideal—learning low-dimensional continuous latent embeddings of logical formulas [103]. Unlike prior work on learning entity and relation embeddings, our approach focuses on the core of knowledge reasoning—learning a richer representation of the logical inference formulas.

Finally, we demonstrate that ProPPR can be used as a generic framework to combine the knowledge and text based methods [102]: a partially-populated KB and a set of relation mentions in context are used as input, and we jointly learn (1) how to extract new KB facts from the relation mentions, and (2) a set of logical rules that allow one to infer new KB facts. It is shown that the proposed joint model for IE and relational learning outperforms universal schemas [85], a state-of-the-art joint method, and that incorporating latent context further improves the results.

Chapter 2

The ProPPR Language

This chapter is organized as follows: first, we provide a broad review of the basic techniques and algorithms that we are going to use throughout this thesis. Then, we will describe the key component of the ProPPR language, including its connection to prior work, the semantics, inference, and learning procedures. We will also provide empirical results on the efficiency and effectiveness of learning. Finally, we close this chapter by relating our work with prior studies in statistical relational learning and probabilistic logics.

2.1 Background

In this section, we introduce the necessary background that our approach builds on: logic program inference as graph search, an approximate Personalized PageRank algorithm, and stochastic logic programs.

2.1.1 Logic Program Inference as Graph Search

To begin with, we first show how inference in logic programs can be formulated as search over a graph. We assume some familiarity with logic programming and will use notations from Lloyd [60]. Let *LP* be a program which contains a set of definite clauses $c_1, \ldots, c_n$, and

```
about(X,Z) :- handLabeled(X,Z)          # base.
about(X,Z) :- sim(X,Y),about(Y,Z)       # prop.
sim(X,Y) :- link(X,Y)                   # sim,link.
sim(X,Y) :-
   hasWord(X,W),hasWord(Y,W),
   linkedBy(X,Y,W)                      # sim,word.
linkedBy(X,Y,W) :- true                 # by(W).
```

Table 2.1: A simple program in ProPPR. See text for explanation.

consider a conjunctive query Q over the predicates appearing in LP. A traditional Prolog interpreter can be viewed as having the following actions. First, construct a "root vertex" v_0, which is a pair (Q,Q), and add it to an otherwise-empty graph $G'_{Q,LP}$. (For brevity, we drop the subscripts of G' where possible.) Then recursively add to G' new vertices and edges as follows: if u is a vertex of the form $(Q,(R_1,\ldots,R_k))$, and c is a clause in LP of the form $R' \leftarrow S'_1,\ldots,S'_\ell$, and R_1 and R' have a most general unifier $\theta = mgu(R_1,R')$, then add to G' a new edge $u \rightarrow v$ where $v = (Q\theta,(S'_1,\ldots,S'_\ell,R_2,\ldots,R_k)\theta)$.[1] Let us call $Q\theta$ the *transformed query* and $(S'_1,\ldots,S'_\ell,R_2,\ldots,R_k)\theta$ the *associated subgoal list*. Empty subgoal lists correspond to solutions, and if a subgoal list is empty, we will denote it by $\Box$.

The graph G' is often large or infinite so it is not constructed explicitly. Instead Prolog performs a depth-first search on G' to find the first *solution vertex* v—i.e., a vertex with an empty subgoal list—and if one is found, returns the transformed query from v as an answer to Q.

Table 2.1 and Figure 2.1 show a simple Prolog program and a proof graph for it. (Ignore for now the annotation after the hash marks, and edge labels on the graphs, which will be introduced below.) For conciseness, in Figure 2.1 only the subgoals $R_1,\ldots,R_k$ are shown in each node $u = (Q,(R_1,\ldots,R_k))$. Given the query $Q = about(a,Z)$, Prolog's depth-first search would return

[1] Here $Q\theta$ denotes the result of applying the substitution θ to Q; for instance, if $Q = about(a,Z)$ and $\theta = \{Z = fashion\}$, then $Q\theta$ is $about(a,fashion)$.

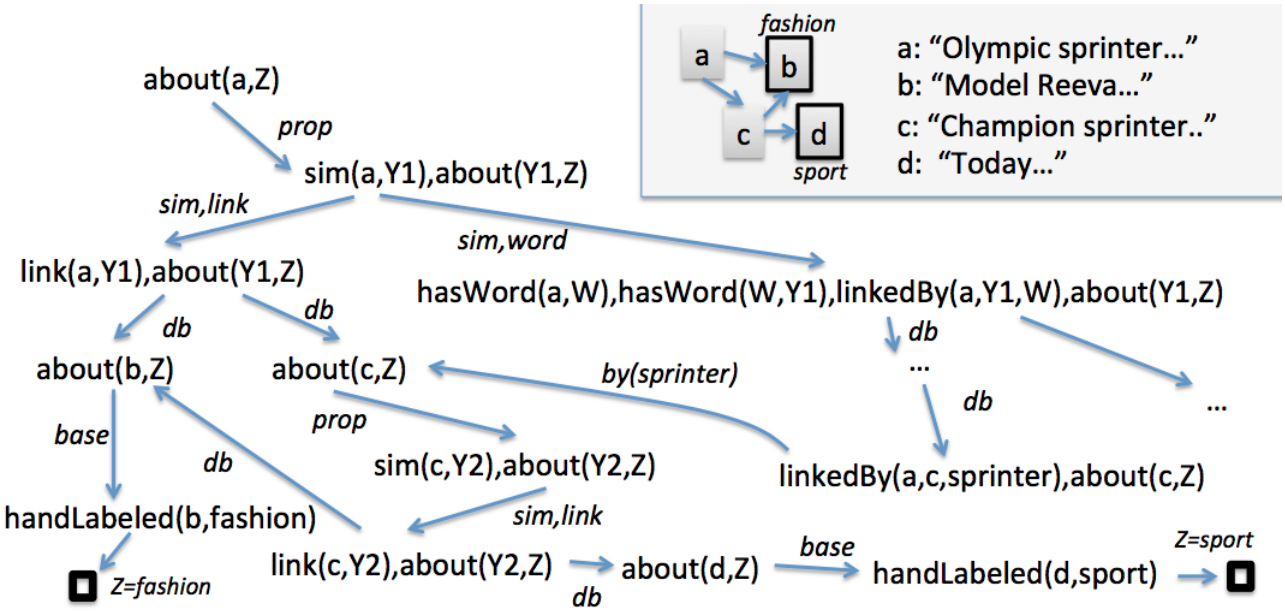


Figure 2.1: A partial proof graph for the query *about(a,Z)*. The upper right shows the link structure between documents a, b, c, and d, and some of the words in the documents. Restart links are not shown.

$Q =$ *about(a,fashion)*. Note that in this proof formulation, the nodes are *conjunctions* of literals, and the structure is, in general, a directed graph, rather than a tree. Also note that the proof is encoded as a graph, not a hypergraph, even if the predicates in the LP are not binary: the edges represent a step in the proof that reduces one conjunction to another, not a binary relation between entities.

2.1.2 Personalized PageRank

Personalized PageRank [78] is an algorithm for ranking nodes in large graphs. In this subsection, we introduce an approximate personalized PageRank method called PageRank-Nibble [2, 3]. The outline of this algorithm is shown in Table 2.2. This method has been used for the

Table 2.2: The PageRank-Nibble algorithm for computing the Personalized PageRank vector in a graph where $\Pr(v|u)$ is the transition probability for reaching v from u, α' is a lower-bound on $\Pr(v_0|u)$ for any node u to be added to the graph $\hat{G}$, and ε is the desired degree of approximation.

define PageRank-Nibble($v_0, \alpha', \varepsilon$):
    let $\mathbf{p} = \mathbf{r} = \mathbf{0}$, let $\mathbf{r}[v_0] = 1$, and let $\hat{G} = \emptyset$
    while $\exists u : \mathbf{r}(u)/|N(u)| > \varepsilon$ do:
        push(u)
    return $\mathbf{p}$
end

define push(u, α'):
    comment: this function modifies $\mathbf{p}$*,* $\mathbf{r}$*, and* $\hat{G}$
    $\mathbf{p}[u] = \mathbf{p}[u] + \alpha' \cdot \mathbf{r}[u]$
    $\mathbf{r}[u] = \mathbf{r}[u] \cdot (1 - \alpha')$
    for $v \in N(u)$:
        add the edge $(u, v, \phi_{u \to v})$ to $\hat{G}$
        if $v = v_0$ then
            $\mathbf{r}[v] = \mathbf{r}[v] + \Pr(v|u)\mathbf{r}[u]$
        else
            $\mathbf{r}[v] = \mathbf{r}[v] + (\Pr(v|u) - \alpha')\mathbf{r}[u]$
    endfor
end

problem of *local partitioning*: in local partitioning, the goal is to find a small, low-conductance[2] component $\hat{G}$ of a large graph G that contains a given node v. In the next section, we will show that this algorithm can also be used to implement a probabilistic Selective Linear Definite (SLD) resolution with bounded computation.

To approximate a "true" PageRank vector $\mathbf{ppr}(\alpha, s)$, we would like to use a pair of distribution $\mathbf{p}$ and $\mathbf{r}$ that satisfy:

$$\mathbf{p} + \mathbf{ppr}(\alpha, \mathbf{r}) = \mathbf{ppr}(\alpha, s)$$

Here, we say that $\mathbf{p}$ and $\mathbf{r}$ are two distributions with this property, where this *approximate PageRank vector* $\mathbf{p}$ approximates $\mathbf{ppr}(\alpha, s)$ with the *residual vector* $\mathbf{r}$.

More specifically, the PageRank-Nibble algorithm that we consider, maintains two vectors: $\mathbf{p}$, an approximation to the personalized PageRank vector[3] associated with node v_0, and $\mathbf{r}$, a vector of "residual errors" in $\mathbf{p}$. Initially, $\mathbf{p} = \emptyset$ and $\mathbf{r} = \{v_0\}$. The algorithm repeatedly picks a node u with a large residual error $\mathbf{r}[u]$, and reduces this error by distributing a fraction α' of it to $\mathbf{p}[u]$, and the remaining fraction back to $\mathbf{r}[u]$ and $\mathbf{r}[v_1], \ldots, \mathbf{r}[v_n]$, where the v_i's are the neighbors of u. The order in which nodes u are picked does not matter for the analysis. In our implementation, we follow Prolog's usual depth-first search as much as possible.

Specifically, after each push operation, the sum of approximate PageRank and the residual vectors is the "true" PageRank probability from the source node: $\mathbf{p} + \mathbf{r} = \mathbf{ppr}(v_0)$. From the algorithm, when PageRank-Nibble terminates at any point, then for any vertex u, the error $\mathbf{ppr}(v_0)[u] - \mathbf{p}[u]$ is $\mathbf{r}[u]$, which is bounded by $\varepsilon|N(u)|$: here, ε is the desired degree of approximation, which is the measure of a normalized difference—the "true" PageRank probability minus the current approximation, and then divided by the total number of neighbors of u. Hence, in any graph where the set of neighbors of u, namely $N(u)$, is bounded by a constant, a good ap-

[2]For small subgraphs G_S, *conductance* of G_S is the ratio of the weight of all edges exiting G_S to the weight of all edges incident on a node in G_S.

[3]In the probabilistic view of personalized PageRank, the i-th entry in the vector $\mathbf{p}$ is the probability of reaching the i-th node v_i from the start node v_0.

proximation will be returned by PageRank-Nibble. PageRank-Nibble incrementally exploits the graph, and the nodes with non-zero weights are the nodes actually touched by PageRank-Nibble. Consider the subset of G' defined by these nodes (the nodes visited by the PageRank-Nibble), clearly this subgraph is also of size $o(\frac{1}{\alpha'\varepsilon})$.

To summarize, the PageRank-Nibble algorithm is an approximation to Personalized PageRank. In the next section, we will show that this algorithm can also be used to implement a probabilistic Selective Linear Definite (SLD) resolution with bounded computation.

2.1.3 Stochastic Logic Programs

In past work on *stochastic logic programs* (SLPs) [26, 68], the randomized traversal of G' was defined by a probabilistic choice, at each node, of which clause to apply, based on a weight for each clause. Specifically, there is a weight for each clause $R_1 \leftarrow Q_1, \ldots, Q_k$ with head R_1, and these weights define a distribution on the children of node $((Q,(R_1,\ldots,R_k)))$; i.e., the clause weights defined a transition probability for the graphs G' defined above. This defines a probability distribution over vertices v, and hence a distribution over transformed queries (i.e. answers) $Q\theta$. The randomized procedure thus produces a distribution over possible answers, which can be tuned by learning to upweight desired correct answers and downweight others.

Note that each different $\Box$ node corresponds to a different proof for Q, and different proofs may be associated with different substitutions θ and hence different solutions to the query Q. In Figure 2.1, for instance, the leaf in the lower left corresponds to the solution $\theta = \{Z = \textit{fashion}\}$ of the query $Q = \textit{about(a,Z)}$, while the leaf in the lower right corresponds to $\{Z = \textit{sport}\}$. In the formalism of SLPs, one assigns a probabilistic score to every node in the proof graph, and then, to assign a probabilistic score to a particular solution Q', one simply sums the probabilities for all leaf nodes $(\Box_i, \theta_i)$ such that $Q\theta_i = Q'$.

2.2 The Programming with Personalized PageRank (PROPPR) Language

2.2.1 Extensions to the Semantics of SLPs

We will now describe our "locally groundable" first-order probabilistic language, which we call ProPPR. Inference for ProPPR is based on a personalized PageRank process over the proof constructed by Prolog's SLD resolution theorem-prover.

Feature-based transition probabilities

We propose two extensions. First, we will introduce a new way of computing clause weights, which allows for a potentially richer parameterization of the traversal process. Briefly, for each clause c in a proof, we associate with it a function $\Phi_c(\theta)$ that produces a feature vector ϕ from the binding to the variables in the head of c. The feature vector ϕ for a clause is defined by the code following "#" after c. For instance, applying the clause "*sim(X,Y):- link(X,Y) # sim, link.*" always yields a vector ϕ that has unit weight on (the dimensions corresponding to) the two ground atoms *sim* and *link*, and zero weight elsewhere; likewise, applying the clause "*linkedBy(X,Y,W):- true # by(W).*" to the goal *linkedBy(a,c,sprinter)* yields a vector ϕ that has unit weight on the atom *by(sprinter)*.

To do this, the annotations after the hashmarks in the program in Table 2.1 are used to generate the features on the edges in Figure 2.1. For example, the *#sim,link* annotation in Table 2.1 is instantiated by the edge in the top-left of Figure 2.1, and *#by(W)* generates a feature template, which can be activated as, for example, the *by(sprinter)* edge in the middle of Figure 2.1. More generally, we associate with each edge $u \rightarrow v$ in the graph a *feature vector* $\phi_{u\rightarrow v}$. This feature vector is produced indirectly, by associating with every clause $c \in LP$ a function $\Phi_c(\theta)$, which produces the vector ϕ associated with an application of c using mgu θ. As an example, if the last clause of the program in Table 2.1 was applied to $(Q, linkedBy(a,c,sprinter), about(c,Z))$ with

mgu $\theta = \{X = a, Y = c, W = sprinter\}$ then $\Phi_c(\theta)$ would be $\{by(sprinter)\}$, if we use a set to denote a sparse vector with 0/1 weights.

In ProPPR, it is possible to have one unique feature for each clause: in fact, this is the default setting in ProPPR. However, allowing user-defined feature vectors gives us more flexibility in guiding the learning system. This allows us to decouple "feature engineering" from writing logic programs. Note that another key difference between SLPs and ProPPR is that the former associates a fixed weight/probability with each clause, whereas in ProPPR, when a feature template is involved, the weight of this clause can depend on the partial instantiation of the clause. For example, the clause "*sim(X,Y) :- hasWord(X,W),hasWord(Y,W),linkedBy(X,Y,W) # sim,word.*" associates two weights with the feature vector for this clause. In SLP, there would be only one weight. Additionally ProPPR allows clauses like "*linkedBy(X,Y,W) :- true # by(W).*" which have feature templates that allow the weight of a clause to depend on the constants used in the proof—e.g., in Table 2.1, depending on the actual word, the second $sim(X,Y)$ could get different weights: if $W =$ "*champion*" the second $sim(X,Y)$ clause gets a weight of 0.9, while for $W =$ "*the*", the weight could be 0.2. This would not be possible in SLPs, except by including a separate clause for each instantiation of W.[4] Note that *db* is a special feature indicating that a database predicate was used.

This feature vector is computed during theorem-proving, and used to annotate the edge $u \rightarrow v$ in G' created by applying c with mgu θ. Finally, an edge $u \rightarrow v$ will be traversed with probability $\Pr(v|u) \propto f(\mathbf{w}, \phi_{u \rightarrow v})$ where $\mathbf{w}$ is a parameter vector and where $f(\mathbf{w}, \phi)$ is a weighting function. (Here we use $f(\mathbf{w}, \phi) = \exp(\mathbf{w} \cdot \phi)$, but any differentiable function would be possible.) This weighting function now determines the probability of a transition, in theorem-proving, from u to v: specifically, $\Pr_{\mathbf{w}}(v|u) \propto f(\mathbf{w}, \phi_{u \rightarrow v})$. Weights in $\mathbf{w}$ default to 1.0, and learning consists of tuning these weights.

[4] We thank an anonymous reviewer of [108] for pointing out this example.

Restart links and solution self-links

Our second and more fundamental extension is to add additional edges in G'. Specifically we add edges (a) from every solution vertex to itself; and (b) also add an edge from every vertex to the start vertex v_0. We will call this augmented graph $G_{Q,LP}$ below (or just G if the subscripts are clear from context). The first type of link, the self-loops, serves to upweight solution nodes, and are a heuristic — all analytic results would hold without them. The second type of link, the restart links, make SLP's graph traversal a *personalized PageRank* (PPR) procedure[20, 78], sometimes known as *random-walk-with-restart* [99]. Self-loop links and restart links are annotated by additional feature vector functions $\Phi_{\text{restart}}(R)$ and $\Phi_{\text{selfloop}}(R)$, which are applied to the leftmost literal R of the subgoal list for u to annotate the edge $u \rightarrow v_0$.

The restart links, which link back to the start vertex, bias the traversal of the proof graph to upweight the results of *short proofs*. To see this, note that if the restart probability $P(v_0|u) = \alpha$ for every node u, then the probability of reaching any node at depth d is bounded by $(1-\alpha)^d$. Later we will show that theoretically these restart edges are important for constructing the approximated local groundings in time independent of the database size.

Summary of the extended proof graph

To summarize, if u is a node of the search graph, $u = (Q\theta, (R_1, \ldots, R_k))$, then the transitions from u, and their respective probabilities, are defined as follows, where Z is an appropriate normalizing constant:

- If $v = (Q\theta\sigma, (S'_1, \ldots, S'_\ell, R2, \ldots, R_k)\theta\sigma)$ is a state derived by applying the clause c (with mgu σ), then
$$\Pr_{\mathbf{w}}(v|u) = \frac{1}{Z} f(\mathbf{w}, \Phi_c(\theta\sigma))$$
- If $v = (Q, \square)$ is a solution, then
$$\Pr_{\mathbf{w}}(v|u) = \frac{1}{Z} f(\mathbf{w}, \Phi_{\text{selfloop}}(R_1\theta))$$

- If $v = v_0 = (Q, Q)$ is the initial state in G, then

$$\Pr_{\mathbf{w}}(v|u) = \frac{1}{Z} f(\mathbf{w}, \Phi_{\text{restart}}(R_1\theta))$$

- If v is any other node, then $\Pr(v|u) = 0$.

Finally we must specify the functions Φ_c and Φ_{restart}. For clauses in *LP*, the feature-vector producing function $\Phi_c(\theta)$ for a clause is specified by annotating c as follows: every clause $c = (R \leftarrow S_1, \ldots, S_k)$ is annotated with a conjunction of "feature literals" $F_1, \ldots, F_\ell$, which are written at the end of the clause after the special marker "#". The function $\Phi_c(\theta)$ then returns a vector $\phi = \{F_1\theta, \ldots, F_\ell\theta\}$, where every $F_i\theta$ must be ground: it cannot contain any free variables.

The requirement[5] that edge features $F_i\theta$ are ground is the reason for introducing the apparently unnecessary predicate *linkedBy(X,Y,W)* into the program of Table 2.1: adding the feature literal *by(W)* to the second clause for *sim* would result in a non-ground feature *by(W)*, since W is a free variable when c is used. Notice also that the weight on the *by(W)* features are meaningful, even though there is only one clause in the definition of *linkedBy*, as the weight for applying this clause competes with the weight assigned to the restart edges.

It would be cumbersome to annotate every database fact, and difficult to learn weights for so many features. Thus, if c is the unit clause that corresponds to a database fact, then $\Phi_c(\theta)$ returns a default value $\phi = \{db\}$[6]. In practice, developers may consider starting with a single, default feature for each clause, which is essentially using the identifies of clauses during learning. Note that for text features, it is essential to use a "feature template" to instantiate all possible words in a DB for learning[7].

[5]The requirement that the feature literals returned by $\Phi_c(\theta)$ must be ground in θ is not strictly necessary for correctness. However, in developing ProPPR programs we noted that non-ground features were usually not what the programmer intended.

[6]If a non-database clause c has no annotation, then the default vector is $\phi = \{id(c)\}$, where *id(c)* is an identifier for the clause c.

[7]For example, in Section 7.2.2, by using feature templates to model the lexical features for information extraction, and we show significant improvements in the experimental section of 7.2.4.

The function $\Phi_{\text{restart}}(R)$ depends on the functor and arity of R. If R is defined by clauses in *LP*, then $\Phi_{\text{restart}}(R)$ returns a unit vector $\phi = \{defRestart\}$. If R is a database predicate (e.g., *hasWord(doc1,W)*) then we follow a slightly different procedure, which is designed to ensure that the restart link has a reasonably large weight even with unit feature weights: we compute n, the number of possible bindings for R, and set $\phi[defRestart] = n \cdot \frac{\alpha}{1-\alpha}$, where α is a global parameter. This means that with unit weights, after normalization, the probability of following the restart link will be α.

Putting this all together with the standard power-iteration approach to computing personalized PageRank over a graph [78], we arrive at the following inference algorithm for answering a query Q, using a weight vector $\mathbf{w}$. We let $\mathbf{W}$ be a matrix such that $\mathbf{W}[u,v] = \Pr_{\mathbf{w}}(v|u)$, and in our discussion, we use $\mathbf{ppr}(v_0)$ to denote the personalized PageRank vector for v_0. Let $N_{u+0}(u)$ be the neighbors of u with non-zero weight.

1. Let $v_0 = (Q,Q)$ be the start node of the search graph. Let G be a graph containing just v_0. Let $\mathbf{v}^0 = \{v_0\}$.

2. For $t = 1,\ldots,T$ (i.e., until convergence):

   For each u with non-zero weight in $\mathbf{v}^{t-1}$, and each $v \in N_{u+0}(u)$, add $(u,v,\phi_{u\to v})$ to G with weight $\Pr_{\mathbf{w}}(v|u)$, and set $\mathbf{v}^t = \mathbf{W} \cdot \mathbf{v}^{t-1}$

3. At this point $\mathbf{v}^T \approx \mathbf{ppr}(v_0)$. Let S be the set of nodes $(Q\theta, \square)$ that have empty subgoal lists and non-zero weight in $\mathbf{v}^T$, and let $Z = \sum_{u \in S} \mathbf{v}^T[u]$. The final probability for the literal $L = Q\theta$ is found by extracting these solution nodes S, and renormalizing:

$$\Pr_{\mathbf{w}}(L) \equiv \frac{1}{Z}\mathbf{v}^T[(L,\square)]$$

where $\mathbf{v}^T[(L,\square)]$ is the unnormalized probability of L reaching this particular solution node. For example, given the query $Q =$ *about(a,Z)* and the program of Table 2.1, this procedure would assign a non-zero probability to the literals *about(a,sport)* and *about(a,fashion)*, concurrently building the graph of Figure 2.1.

As a further illustration of the sorts of ProPPR programs that are possible, some small sample programs are shown in Table 2.3. Clauses c_1 and c_2 are, together, a bag-of-words classifier: each proof of *predictedClass(D,Y)* adds some evidence for document D having class Y, with the weight of this evidence depending on the weight given to c_2's use in establishing *related(w,y)*, where w is a specific word in D and y is a possible class label. In turn, c_2's weight depends on the weight assigned to the $r(w,y)$ feature by $\mathbf{w}$, relative to the weight of the restart link.[8] Adding c_3 and c_4 to this program implements label propagation[9], and adding c_5 and c_6 implements a sequential classifier. These examples show that ProPPR allows many useful heuristics to be encoded as programs.

Discussion

Thus far, we have introduced a language quite similar to SLPs. The power-iteration PPR computation outlined above corresponds to a depth-bounded breadth-first search procedure, and the main extension of ProPPR, relative to SLPs, is the ability to label a clause application with a feature vector, instead of the clause's identifier. Below, however, we will discuss a much faster approximate grounding strategy, which leads to a novel proof strategy, and a parallelizable weight-learning method.

2.2.2 Locally Grounding a Query

Note that the procedure for computing PPR on the proof graph both performs inference by computing a distribution over literals $Q\theta$ and "grounds" the query, by constructing a graph G. ProPPR inference for this query can be re-done efficiently[10], by running an ordinary PPR process

[8]The existence of the restart link thus has another important role in this program, as it avoids a sort of "label bias problem" [52] in which local decisions are difficult to adjust.

[9]Note that we use the *inDoc* predicate for efficiency purposes; in ProPPR, like Prolog, predicates are indexed by their first argument.

[10]This is useful for faster weight learning, which will be explained in the next subsection.

Table 2.3: Some more sample ProPPR programs. $LP = \{c_1, c_2\}$ is a bag-of-words classifier (see text). $LP = \{c_1, c_2, c_3, c_4\}$ is a recursive label-propagation scheme, in which predicted labels for one document are assigned to similar documents, with similarity being an (untrained) cosine distance-like measure. $LP = \{c_1, c_2, c_5, c_6\}$ is a sequential classifier for document sequences.

c_1: predictedClass(Doc,Y) :-
    possibleClass(Y),
    hasWord(Doc,W),
    related(W,Y) # c1.

c_2: related(W,Y) :- true,
  # relatedFeature(W,Y)

Database predicates:
hasWord(D,W): doc D contains word W
inDoc(W,D): doc D contains word W
previous(D1,D2): doc D2 precedes D1
possibleClass(Y): Y is a class label

c_3: predictedClass(Doc,Y) :-
    similar(Doc,OtherDoc),
    predictedClass(OtherDoc,Y) # c3.

c_4 : similar(Doc1,Doc2) :-
    hasWord(Doc1,W),
    inDoc(W,Doc2) # c4.

c_5 : predictedClass(Doc,Y) :-
    previous(Doc,OtherDoc),
    predictedClass(OtherDoc,OtherY),
    transition(OtherY,Y) # c5.

c_6: transition(Y1,Y2) :- true,
  # transitionFeature(Y1,Y2)

Table 2.4: The PageRank-Nibble-Prove algorithm for inference in ProPPR. α' is a lower-bound on $\Pr(v_0|u)$ for any node u to be added to the graph $\hat{G}$, and ε is the desired degree of approximation of the **ppr** vector. The nodes touched by PageRank-Nibble also define a subset $\hat{G}$ of the proof graph G.

define PageRank-Nibble-Prove(Q):

let $\mathbf{v}$ =PageRank-Nibble($(Q,Q),\alpha',\varepsilon$)

let $S = \{u : \mathbf{p}[u] > \varepsilon \text{ and } u = (Q\theta, \square)\}$

let $Z = \sum_{u \in S} \mathbf{p}[u]$

define $\Pr_{\mathbf{w}}(L) \equiv \frac{1}{Z}\mathbf{v}[(L,\square)]$

end

on G. Unfortunately, the grounding G can be very large: it does not always include the entire database, but if T is the number of iterations until convergence for the sample program of Table 2.1 on the query $Q = about(d,Y)$, G will include a node for every page within T hyperlinks of d. To address this problem, we can use a proof procedure based on PageRank-Nibble, the algorithm described in Subsection 2.1.2.

The PageRank-Nibble-Prove algorithm is shown in Table 2.4, which calls the approximate Personalized PageRank algorithm in Table 2.2. Relative to PageRank-Nibble, the main differences are the use of a lower-bound on α rather than a fixed restart weight and the construction of the graph $\hat{G}$. Putting them together, we have the following efficiency bound:

Theorem 1 (Andersen, Chung, Lang) *Let u_i be the i-th node pushed by PageRank-Nibble-Prove. Then, for the sum of all neighbors of u_i, we have $\sum_i |N(u_i)| < \frac{1}{\alpha'\varepsilon}$.*

This can be proved by noting that initially $||\mathbf{r}||_1 = 1$, and also that $||\mathbf{r}||_1$ decreases by at least $\alpha'\varepsilon|N(u_i)|$ on the i-th push. As a direct consequence we have the following:

Corollary 1 *The number of edges in the graph $\hat{G}$ produced by PageRank-Nibble-Prove is no more than $\frac{1}{\alpha'\varepsilon}$.*

Note that we expand the proof tree gradually, and generate a partial proof tree $\hat{G_Q}$ during this process. Importantly, the bound holds *independent of the size of the full database of facts*. The bound also holds regardless of the size or loopiness of the full proof graph, so this inference procedure will work for recursive logic programs[11]. Note that the bound depends on $1/\alpha'$, so if $\alpha' = 0$, then this approximate PageRank procedure may not terminate.

We should emphasize a limitation of this analysis: this approximation result holds for the individual nodes in the proof tree, not the answers $Q\theta$ to a query Q. Following SLPs, the probability of an answer $Q\theta$ is the sum of the weights of all solution nodes that are associated with θ, so if an answer is associated with n solutions, the error for its probability estimate with PageRank-Nibble-Prove may be as large as $n\varepsilon$.

To summarize, we have outlined an efficient approximate proof procedure, which is closely related to personalized PageRank. As a side-effect of inference for a query Q, this procedure will create a ground graph $\hat{G}_Q$ on which personalized PageRank can be run directly, without any relatively expensive manipulation of first-order theorem-proving constructs such as clauses or logical variables. Since ProPPR may find multiple answers to a query, we may sum up the *ppr* scores for proofs of each candidate answer, and re-normalize the candidate answer scores to derive final scores for comparison. Again, we have two main reasons to make this design philosophy of preferring shorter proofs: 1) we would like to perform efficient inference with bounded depth; 2) we prefer answers supported by direct evidence, rather than long chains of indirect evidence.

[11]For undirected graphs, it can also be shown [2, 3] that the subgraph $\hat{G}$ is in some sense a "useful" subset of the full proof space: for an appropriate setting of ε, if there is a low-conductance subgraph G_* of the full graph that contains v_0, then G_* will be contained in $\hat{G}$: thus if there is a subgraph G_* containing v_0 that approximates the full graph well, PageRank-Nibble will find (a supergraph of) G_*.

As we will see, this "locally grounded"[12] graph will be very useful in learning weights $\mathbf{w}$ to assign to the features of a ProPPR program.

2.2.3 Learning for ProPPR

As noted above, inference for a query Q in ProPPR is based on a personalized PageRank process over the graph associated with the SLD proof of a query goal G. More specifically, the edges $u \rightarrow v$ of the graph G are annotated with feature vectors $\phi_{u \rightarrow v}$, and from these feature vectors, weights are computed using a parameter vector $\mathbf{w}$, and finally normalized to form a probability distribution over the neighbors of u. The "grounded" version of inference is thus a personalized PageRank process over a graph with feature-vector annotated edges.

In prior work, Backstrom and Leskovec [6] outlined a family of supervised learning procedures for this sort of annotated graph. In the simpler case of their learning procedure, an example is a triple (v_0, u, y) where v_0 is a query node, u is a node in the personalized PageRank vector $\mathbf{p}_{v_0}$ for v_0, y is a target value, and a loss $\ell(v_0, u, y)$ is incurred if $\mathbf{p}_{v_0}[u] \neq y$. In the more complex case of "learning to rank", an example is a triple (v_0, u_+, u_-) where v_0 is a query node, u_+ and u_- are nodes in the personalized PageRank vector $\mathbf{p}_{v_0}$ for v_0, and a loss is incurred unless $\mathbf{p}_{v_0}[u_+] \geq \mathbf{p}_{v_0}[u_-]$. The core of Backstrom and Leskovec's result is a method for computing the gradient of the loss on an example, given a differentiable feature-weighting function $f(\mathbf{w}, \phi)$ and a differentiable loss function ℓ. The gradient computation is broadly similar to the power-iteration method for computation of the personalized PageRank vector for v_0. Given the gradient, a number of optimization methods can be used to compute a local optimum.

Instead of directly using the above learning approach for ProPPR, we decompose the pairwise ranking loss into a standard positive-negative log loss function. The training data D is a set of triples $\{(Q^1, P^1, N^1), \ldots, (Q^m, P^m, N^m)\}$ where each Q^k is a query, $P^k = \langle Q\theta^1_+, \ldots, Q\theta^I_+ \rangle$ is a list

[12]Note that "local grounding" means constructing a proof graph, but the nodes in this graph need not to be ground terms in the logic programming sense.

of correct answers, and N^k is a list $\langle Q\theta^1_-, \ldots, Q\theta^J_- \rangle$ of incorrect answers. We use a log loss with L_2 regularization of the parameter weights. Hence the final function to be minimized is

$$-\left(\sum_{k=1}^{I} \log \mathbf{p}_{v_0}[u^k_+] + \sum_{k=1}^{J} \log(1 - \mathbf{p}_{v_0}[u^k_-]) \right) + \mu ||\mathbf{w}||_2^2$$

To optimize this loss, we use stochastic gradient descent (SGD), rather than the quasi-Newton method of Backstrom and Leskovic. Weights are initialized to $1.0 + \delta$, where δ is randomly drawn from $[0, 0.01]$. We set the learning rate β of SGD to be $\beta = \frac{\eta}{\text{epoch}^2}$ where epoch is the current epoch in SGD, and η, the initial learning rate, defaults to 1.0.

We implemented SGD because it is fast and has been adapted to parallel learning tasks [76, 120]. Local grounding means that learning for ProPPR is quite well-suited to parallelization. The step of locally grounding each Q^i is "embarrassingly" parallel, as every grounding can be done independently. To parallelize the weight-learning stage, we use multiple threads, each of which computes the gradient over a single grounding $\hat{G}_{Q^i}$, and all of which access a single shared parameter vector $\mathbf{w}$. The shared parameter vector is a potential bottleneck [119]; while it is not a severe one on moderate-size problems, contention for the parameters does become increasingly important with many threads.

2.3 Results

We now present results using ProPPR on two statistical relational learning tasks. Our first sample task is an entity resolution task previously studied as a test case for MLNs [90]. The program we use in the experiments is shown in Table 2.5: it is approximately the same as the MLN(B+T) approach from Singla and Domingos.[13] To evaluate accuracy, we use the Cora dataset, a collection of 1295 bibliography citations that refer to 132 distinct papers. We set the regularization coefficient μ to 0.001 and the number of epochs to 5.

[13]The principle difference is that we do not include tests on the absence of words in a field in our clauses, and we drop the non-horn clauses from their program.

Our second task is a bag-of-words classification task, which was previously studied as a test case for both ProbLog [39] and MLNs [61]. In this experiment, we use the following ProPPR program:

```
class(X,Y) :- has(X,W), isLabel(Y), related(W,Y).
related(W,Y) :- true # w(W,Y).
```

which is a bag-of-words classifier that is approximately[14] the same as the ones used in prior work [39, 61]. The dataset we use is the WebKb dataset, which includes a set of web pages from four computer science departments (Cornell, Wisconsin, Washington, and Texas). Each web page has one or multiple labels: *course, department, faculty, person, research project, staff, and student*. The task is to classify the given URL into the above categories. This dataset has a total of 4165 web pages. Using our ProPPR program, we learn a separate weight for each word for each label.

2.3.1 Efficiency

For these smaller problems, we can evaluate the cost of the PageRank-Nibble-Prove inference/grounding technique relative to a power-iteration based prover. Table 2.6 shows the time required for inference with uniform weights for a set of 52 randomly chosen entity-resolution tasks from the Cora dataset, using a Python implementation of the theorem-prover. We report the time in seconds for all 52 tasks, as well as the mean average precision (MAP) of the scoring for each query. It is clear that PageRank-Nibble-Prove offers a substantial speedup on these problems with little loss in accuracy: on these problems, the same level of accuracy is achieved in less than a tenth of the time.

While the speedup in inference time is desirable, the more important advantages of the local grounding approach are that (1) grounding time, and hence inference, need not grow with the database size and (2) learning can be performed in parallel, by using multiple threads for parallel computations of gradients in SGD. Figure 2.2 illustrates the first of these points: the scalability

[14]Note that we do not use the negation rule and the link rule from Lowd and Domingos.

Table 2.5: ProPPR program used for entity resolution.

```
samebib(BC1,BC2) :-
    author(BC1,A1),sameauthor(A1,A2),authorinverse(A2,BC2)                # author.
samebib(BC1,BC2) :-
    title(BC1,A1),sametitle(A1,A2),titleinverse(A2,BC2)                   # title.
samebib(BC1,BC2) :-
    venue(BC1,A1),samevenue(A1,A2),venueinverse(A2,BC2)                   # venue.
samebib(BC1,BC2) :-
    samebib(BC1,BC3),samebib(BC3,BC2)                                     # tcbib.
sameauthor(A1,A2) :-
    haswordauthor(A1,W),haswordauthorinverse(W,A2),keyauthorword(W)       # authorword.
sameauthor(A1,A2) :-
    sameauthor(A1,A3),sameauthor(A3,A2)                                   # tcauthor.
sametitle(A1,A2) :-
    haswordtitle(A1,W),haswordtitleinverse(W,A2),keytitleword(W)          # titleword.
sametitle(A1,A2) :-
    sametitle(A1,A3),sametitle(A3,A2)                                     # tctitle.
samevenue(A1,A2) :-
    haswordvenue(A1,W),haswordvenueinverse(W,A2),keyvenueword(W)          # venueword.
samevenue(A1,A2) :-
    samevenue(A1,A3),samevenue(A3,A2)                                     # tcvenue.
keyauthorword(W) :- true                                                  # authorWord(W).
keytitleword(W) :- true                                                   # titleWord(W).
keyvenueword(W) :- true                                                   # venueWord(W).
```

Table 2.6: Performance of the approximate PageRank-Nibble-Prove method on the Cora dataset, compared to the grounding by running personalized PageRank to convergence (power iteration). In all cases $\alpha = 0.1$.

ε	MAP	Time(sec)
0.0001	0.30	28
0.00005	0.40	39
0.00002	0.53	75
0.00001	0.54	116
0.000005	0.54	216
power iteration	0.54	819

of the PageRank-Nibble-Prove method as the database size increases. For comparison, we also show the inference time for MLNs with three inference methods: Gibbs refers to Gibbs sampling, Lifted BP is the lifted belief propagation method, and MAP is the maximum a posteriori inference approach. In each case the performance task is inference over 16 test queries.

Note that ProPPR's runtime is constant, independent of the database size: it takes essentially the same time for $2^8 = 256$ entities as for $2^4 = 16$. In contrast, lifted belief propagation is up to 1000 times slower on the larger database.

Figure 2.3 explores the speedup in learning due to multi-threading. The weight-learning is using a Java implementation of the algorithm which runs over ground graphs. For Cora, the speedup is nearly optimal, even with 16 threads running concurrently. For WebKB, while learning time averages about 950 seconds with a single thread, it can be reduced to about 120 seconds if 16 threads are used. For comparison, Lowd and Domingos report that around 10,000 seconds were needed to obtain the best results for MLNs.

Table 2.7: AUC results on Cora citation-matching.

	Cites	Authors	Venues	Titles
MLN(Fig 2.1)	0.513	0.532	0.602	0.544
MLN(S&D)	0.520	0.573	0.627	0.629
ProPPR(**w**=1)	0.680	0.836	0.860	**0.908**
ProPPR	**0.800**	**0.840**	**0.869**	0.900

Table 2.8: AUC results on the WebKb classification task. ProbLog results are from [39], and MLN results are from [61]. Co.: Cornell. Wi.: Wisconsin. Wa.: Washington. Te.: Texas.

	Co.	Wi.	Wa.	Te.	Avg.
ProbLog	–	–	–	–	0.606
MLN (VP)	–	–	–	–	0.605
MLN (CD)	–	–	–	–	0.604
MLN (CG)	–	–	–	–	0.730
ProPPR(**w**=1)	0.501	0.495	0.501	0.505	0.500
ProPPR	**0.785**	**0.779**	**0.795**	**0.828**	**0.797**

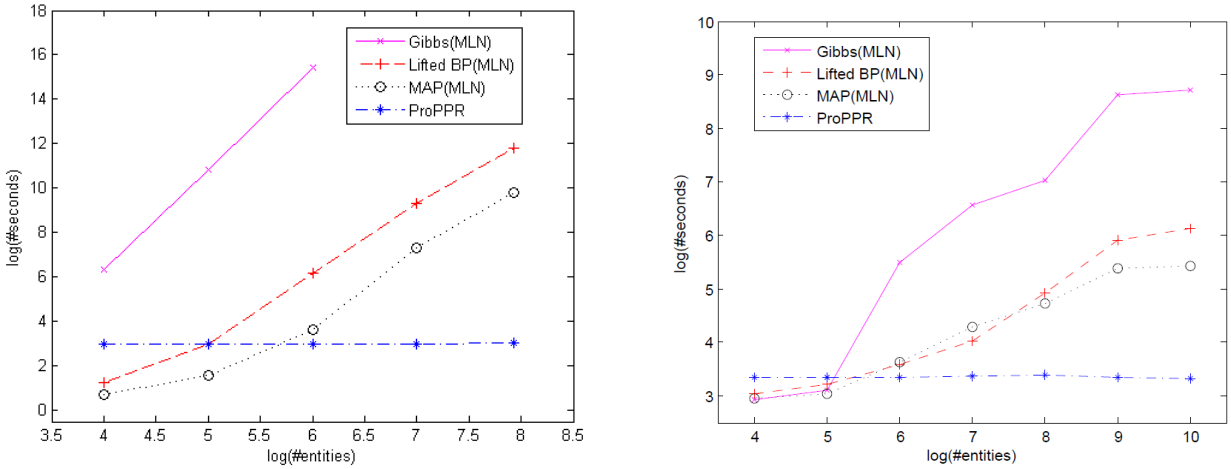


Figure 2.2: Run-time for inference when using ProPPR (with a single thread) as a function of the number of entities in the database. The base of the log is 2. Left, the Cora dataset; right, the WebKB dataset. These results correspond to ProPPR's theoretical guarantee: the local grounding approach constructs query-based subgraphs with bounded depth.

2.3.2 Effectiveness of Learning

We finally consider the effectiveness of weight learning. For Cora, we train on the first four sections of the Cora dataset, and report results on the fifth. Table 2.7 shows AUC on the test set used by Singla and Domingos for several methods. The line for MLN(Fig 2.1) shows results obtained by an MLN version of the program of Figure 2.1. The line MLN(S&D) shows analogous results for the best-performing MLN from [90]. Compared to these methods, ProPPR does quite well even before training (with unit feature weights, $\mathbf{w}$=1); the improvement here is likely due to the ProPPR's bias towards short proofs, and the tendency of the PPR method to put more weight on shared words that are rare, and hence have lower fanout in the graph walk. Training ProPPR improves performance on three of the four tasks, and gives the most improvement on citation-matching, the most complex task.

The results in Table 2.7 all use the same data and evaluation procedure, and the MLNs were trained with the state-of-the-art Alchemy system using the recommended commands for this data,

which is distributed with Alchemy.[15] However, we should note that the MLN results reproduced here are not identical to previous-reported ones [90]. Singla and Domingos used a number of complex heuristics that are difficult to reproduce—e.g., one of these was combining MLNs with a heuristic, TFIDF-based matching procedure based on canopies [63]. While the trained ProPPR model outperforms the reproduced MLN model in all prediction tasks, it outperforms the reported results from Singla and Domingos only on *venue*, and does less well than the reported results on *citation* and *author*.[16]

On the Webkb dataset, we use the usual cross-validation method [39, 61]: in each fold, for the four universities, we train on three, and report result on the fourth. In Table 2.8, we show the detailed AUC results of each fold, as well as the averaged results. If we do not perform weight learning, the averaged result is equivalent to a random baseline. As reported by Gutmann et al. the ProbLog approach obtains an AUC of 0.606 on the dataset [39], and as reported by Lowd and Domingos, the results for voted perceptron algorithm (MLN VP, AUC ≈ 0.605) and the contrastive divergence algorithm (MLN CD, AUC ≈ 0.604) are in same range as ProbLog [61]. ProPPR obtains an AUC of 0.797, which outperforms the prior results reported by ProbLog and MLN.

2.4 Related Work

Although we have chosen here to compare mainly to MLNs [83, 90], ProPPR represents a rather different philosophy toward language design: rather than beginning with a highly-expressive but intractable logical core, we begin with a limited logical inference scheme and add to it a minimal set of extensions that allow probabilistic reasoning, while maintaining stable, efficient inference and learning. More specifically, MLNs aim at constructing the possible "world" of the knowledge, and view the ground formulas as Markov Random Fields (MRFs). To learn weights

[15] http://alchemy.cs.washington.edu

[16] Performance on *title* matching is not reported by Singla and Domingos.

for each clause, it is now a inference and learning problem on MRFs. Unfortunately, inference and learning on MRFs are intractable. In contrast, ProPPR does not aim at building a factor graph of ground atoms: instead, ProPPR performs local grounding and constructs many subgraphs with bounded depth for each query. With this property, ProPPR is ideal for solving large, real-world relational learning and reasoning problems. On the other hand, in order to be efficient, ProPPR only considers Horn clauses, whereas MLNs are more expressive: MLNs have negations in the semantic of its first-order logic. For smaller problems and tasks involving the frequent usage of negations, the readers might also want to consider MLNs as an alternative.

To summarize, while ProPPR is less expressive than MLNs (for instance, it is limited to definite clause theories) it is also much more efficient. This philosophy is similar to that illustrated by probabilistic similarity logic (PSL) [16]; however, unlike ProPPR, PSL does not include a "local" grounding procedure. This thesis also aligns with the lifted personalized PageRank [1] algorithm, which can be easily incorporated as an alternative inference algorithm in our language.

Technically, ProPPR is most similar to stochastic logic programs (SLPs) [26]. The key innovation is the integration of a restart into the random-walk process, which, as we have seen, leads to very different computational properties.

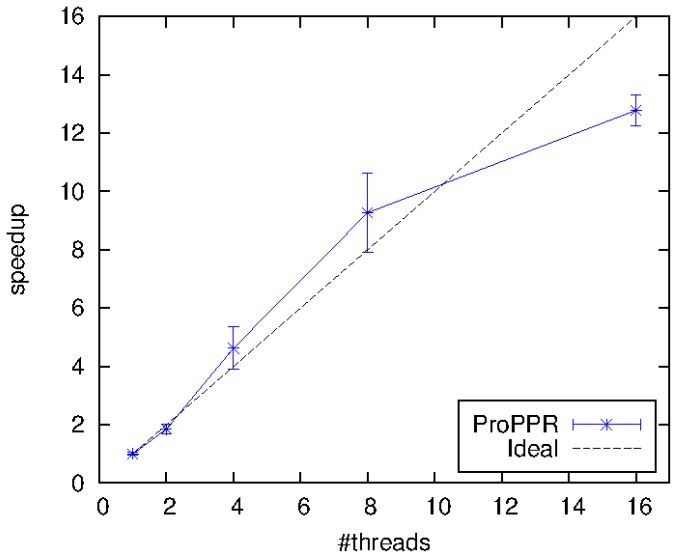
ProbLog [27], like ProPPR, also supports approximate inference, in a number of different variants. An extension to ProbLog also exists which uses decision theoretic analysis to determine when approximations are acceptable [100].

There has also been some prior work on reducing the cost of grounding probabilistic logics: notably, Shavlik et al [87] describe a preprocessing algorithm called FROG that uses various heuristics to greatly reduce grounding size and inference cost, and Niu et al. [75] describe a more efficient bottom-up grounding procedure that uses an RDBMS. Other methods that reduce grounding cost and memory usage include "lifted" inference methods (e.g., [92]) and "lazy" inference methods (e.g., [91]); in fact, the LazySAT inference scheme for Markov networks is broadly similar algorithmically to PageRank-Nibble-Prove, in that it incrementally extends

a network in the course of theorem-proving. However, there is no theoretical analysis of the complexity of these methods, and experiments with FROG and LazySAT suggest that they still lead to groundings that grow with DB size, albeit more slowly.

As noted above, ProPPR is also closely related to the PRA, learning algorithm for link prediction [53]. Like ProPPR, PRA uses random walk processes to define a distribution, rather than some other forms of logical inference, such as belief propagation. In this respect PRA and ProPPR appear to be unique among probabilistic learning methods; however, this distinction may not be as great as it first appears, as it is known there are close connections between personalized PageRank and traditional probabilistic inference schemes.[17] PRA is less expressive than ProPPR, as noted above.

[17]For instance, it is known that personalized PageRank can be used to approximate belief propagation on certain graphs [23].



	Co.	Wi.	Wa.	Te.	Avg.
1	1190.4	504.0	1085.9	1036.4	954.2
2	594.9	274.5	565.7	572.5	501.9
4	380.6	141.8	404.2	396.6	330.8
8	249.4	94.5	170.2	231.5	186.4
16	137.8	69.6	129.6	141.4	119.6

Figure 2.3: Performance of the parallel SGD method on the WebKB dataset. The x axis is the number of threads on a multicore machine, and the y axis is the speedup factor over a single-threaded implementation. Co.: test on the Cornell subset. Wi.: test on the Wisconsin subset. Wa.: test on the Washington subset. Te.: test on the Texas subset.

Chapter 3

Scalable Knowledge Base Inference with ProPPR

In this section, we focus on improving the state-of-the-art in learning weights for inference rules for a noisy KB using ProPPR. The road map is as follows:

- first, we introduce the challenges of inference in a noisy KB in the next section;
- second, in Section 3.2, we describe the background of PRA [54], which is a state-of-the-art algorithm that learns non-recursive theories of a particular type;
- finally, we then show in Section 3.3 how one can use ProPPR to form a recursive theory of PRA's learned rules to improve this learning scheme.

A detailed version of this chapter can be found in [108].

3.1 Challenges of Inference in a Noisy KB

A number of recent efforts in industry [89] and academia [18, 42, 95] have focused on automatically constructing large knowledge bases (KBs). Because automatically-constructed KBs are typically imperfect and incomplete, inference in such KBs is non-trivial.

We situate our study in the context of the NELL (Never Ending Language Learning) research project, which is an effort to develop a never-ending learning system that operates 24 hours per day, for years, to continuously improve its ability to extract structured facts from the web [18]. NELL is given as input an ontology that denotes hundreds of categories (e.g., person, beverage, athlete, sport) and two-place typed relations among these categories (e.g., athletePlaysSport(Athlete, Sport)), which it must learn to extract from the web. NELL is also provided a set of 10 to 20 positive seed examples of each such category and relation, along with a downloaded collection of 500 million web pages from the ClueWeb2009 corpus [17] as unlabeled data, and access to 100,000 queries each day to Google's search engine. NELL uses a multi-strategy semi-supervised multi-view learning method to iteratively grow the set of extracted "beliefs".

This task is challenging for two reasons. First, the extensional knowledge inference is not only incomplete, but also noisy, since it is extracted imperfectly from the web. For example, a football team might be wrongly recognized as two separate entities, one with connections to its team members, and the other with a connection to its home stadium. Second, the sizes of inference problems are large relative to those in many other probabilistic inference tasks. Given the very large broad-coverage KBs that modern information extraction systems produce [18, 95], even a grounding of size that is only linear in the number of facts in the database, $|DB|$, would be impractically large for inference on real-world problems.

Past work on first-order reasoning has sought to address the first problem by learning "soft" inference procedures, which are more reliable than "hard" inference rules, and to address the second problem by learning restricted inference procedures. In the next sub-section, we will recap a recent development in solving these problems, and draw a connection to the ProPPR language.

3.2 Related Work: Inference using the Path Ranking Algorithm (PRA)

Lao et al. [54] use the path ranking algorithm (PRA) to learn an "inference" procedure based on a weighted combination of "paths" through the KB graph. PRA is a relational learning system which generates and appropriately weights rules, which accurately infer new facts from the existing facts in the noisy knowledge base. As an illustration, PRA might learn rules such as those in Table 3.1, which correspond closely to Horn clauses, as shown in the table.

PRA only considers rules which correspond to "paths", or chains of binary, function-free predicates. Like ProPPR, PRA will weight some solutions to these paths more heavily than others: specifically, weights of the solutions to a PRA "path" are based on random-walk probabilities in the corresponding graph. For instance, the last clause of Table 3.1, which corresponds to the "path"

$$T \xrightarrow{\text{teamHasAthlete}} A \xrightarrow{\text{athletePlaysSport}} S$$

can be understood as follows:

1. Given a team T, construct a uniform distribution $\mathscr{A}$ of athletes such that $A \in \mathscr{A}$ is a athlete playing for team T.
2. Given $\mathscr{A}$, construct a distribution of sports $\mathscr{S}$ such that $S \in \mathscr{S}$ is played by A.

This final distribution $\mathscr{S}$ is the result: thus the path gives a weighted distribution over possible sports played by a team.

The paths produced by PRA, together with their weighting scheme, corresponds precisely to ProPPR clauses. More generally, the output of PRA corresponds roughly to a ProPPR program in a particular form—namely, the form

$$b(S,T) \leftarrow r_{1,1}(S,X_1), r_{1,2}(X_1,X_2), \ldots, r_{1,k_1}(X_{k_1-1},T).$$
$$b(S,T) \leftarrow r_{2,1}(S,X_1), r_{2,2}(X_1,X_2), \ldots, r_{2,k_2}(X_{k_2-1},T).$$
$$\vdots$$

where b is the binary predicate being learned, and the $r_{i,j}$'s are other predicates defined in the database. In Table 3.1, we emphasize that the $r_{i,j}$'s are already extensionally defined by prefixing them with the string "fact". PRA generates a very large number of such rules, and then combines them using a sparse linear weighting scheme, where the weighted solutions associated with a single "path clause" are combined with learned parameter weights to produce a final ranking over entity pairs. More formally, following the notation of [53], we define a *relation path RE* as a sequence of relations $r_1, ..., r_\ell$. For any relation path $RE = r_1, ..., r_\ell$, and seed node s, a *path constrained random walk* defines a distribution h as $h_{s,RE}(e) = 1$ if $e = s$, and $h_{s,RE}(e) = 0$ otherwise. If RE is not empty, then $RE' = r_1, ..., r_{\ell-1}$, such that:

$$h_{s,RE}(e) = \sum_{e' \in RE'} h_{s,RE'}(e') \cdot P(e|e'; r_\ell) \tag{3.1}$$

where the term $P(e|e'; r_\ell)$ is the probability of reaching node e from node e' with a one-step random walk with edge type r_ℓ; that is, it is $\frac{1}{k}$, where $k = |\{e' : r_\ell(e, e')\}|$, i.e., the number of entities e' related to e via the relation r_ℓ.

Assume we have a set of paths $RE_1, ..., RE_n$. The PRA algorithm treats each entity-pair $h_{s,RE}(e)$ as a *path feature* for node e, and ranks entities using a linear weighting scheme:

$$w_1 h_{s,RE_1}(e) + w_2 h_{s,RE_2}(e) + ... + w_n h_{s,RE_n}(e) \tag{3.2}$$

where w_i is the weight for the path RE_i. PRA then learns the weights $\mathbf{w}$ by performing using elastic net-like regularized maximum likelihood estimation of the following objective function:

$$\sum_i j_i(w) - \mu_1 \|\mathbf{w}\|_1 - \mu_2 \|\mathbf{w}\|_2^2 \tag{3.3}$$

Here μ_1 and μ_2 are regularization coefficients for elastic net regularization, and the loss function $j_i(w)$ is the per-instance objective function. The regularization on $\|\mathbf{w}\|_1$ tends to drive weights to zero, which allows PRA to produce a sparse classifier with relatively small number of path clauses. More details on PRA can be found elsewhere [53].

Table 3.1: Example PRA rules learned from NELL, written as Prolog clauses.

PRA Paths for inferring **athletePlaysSport**:
athletePlaysSport(A,S) :- factAthletePlaysForTeam(A,T),factTeamPlaysSport(T,S).
PRA Paths for inferring **teamPlaysSport**:
teamPlaysSport(T,S) :-
factMemberOfConference(T,C),factConferenceHasMember(C,T'),factTeamPlaysSport(T',S).
teamPlaysSport(T,S) :-
factTeamHasAthlete(T,A),factAthletePlaysSport(A,S).

3.3 From Non-Recursive to Recursive Theories: Joint Inference for Multiple Relations

One important limitation of PRA is that it learns only programs in the limited form given above. In particular, PRA can not learn or even execute recursive programs, or programs with predicates of arity more than two. PRA also must learn each predicate definition completely independently.

To see why this is a limitation consider the program in Table 3.1, which could be learned by PRA by invoking it twice, once for the predicate *athletePlaysSport* and once for *teamPlaysSport*. We call this formulation the *non-recursive formulation* for a theory. An alternative would be to define two mutually recursive predicates, as in Table 3.2. We call this the *recursive formulation*. Learning weights for theories written using the recursive formulation is a *joint* learning task, since several predicates are considered together. In the next section, we ask the question: can joint learning, via weight-learning of mutually recursive programs of this sort, improve performance for a learned inference scheme for a KB?

Table 3.2: Example recursive Prolog rules constructed from PRA paths.

Rules for inferring **athletePlaysSport**:
athletePlaysSport(A,S) :- factAthletePlaysSport(A,S).
athletePlaysSport(A,S) :- athletePlaysForTeam(A,T),teamPlaysSport(T,S).
Rules for inferring **teamPlaysSport**:
teamPlaysSport(T,S) :- factTeamPlaysSport(T,S).
teamPlaysSport(T,S) :- memberOfConference(T,C),conferenceHasMember(C,T'),teamPlaysSport(T',S).
teamPlaysSport(T,S) :- teamHasAthlete(T,A),athletePlaysSport(A,S).

3.4 Experiments in KB Inference

To understand the locally groundable first-order logic in depth, we investigate ProPPR on the difficult problem of drawing reliable inferences from imperfectly extracted knowledge. In this experiment, we create training data by using NELL's KB as of iteration 713, and test, using as positive examples new facts learned by NELL in later iterations. Negative examples are created by sampling beliefs from relations that are mutually exclusive with the target relation. Throughout this section, we set the number of SGD optimization epochs to 10. Since PRA has already applied the elastic net regularizer when learning the weights of different rules, and we are working with multiple subsets with various sizes of input, μ was set to 0 in ProPPR's SGD learning in this section.

For experimental purposes, we construct a number of varying-sized versions of the KB using the following procedure. First, we construct a "knowledge graph", where the nodes are entities and the edges are the binary predicates from NELL. Then, we pick a seed entity s, and find the M entities that are ranked highest using a simple untyped random walk with restart over the full knowledge graph from seed s. Finally, we project the KB to just these M entities: i.e., we select

all entities in this set, and all unary and binary relationships from the original KB that concern only these M entities.

This process leads to a well-connected knowledge base of bounded size, and by picking different seeds s, we can create multiple different knowledge bases to experiment on. In the experiments below, we used the seeds "Google", "The Beatles", and "Baseball" obtaining KBs focused on technology, music, and sports, respectively.

In this section, we mainly look at three types of rules:

- *KB non-recursive*: the simple non-recursive KB rules that do not contain PRA paths (e.g. teamPlaysSport(T,S) :- factTeamPlaysSport(T,S).);
- *PRA non-recursive*: the non-recursive PRA rules (e.g. rules in Table 3.1);
- *PRA recursive*: the recursive formulation of PRA rules (e.g. rules in Table 3.2).

When we conducted these experiments, ProPPR's structure learning components were not available, so we leveraged the pre-learned rules from PRA to test the scalability of our inference engine. To do this, we construct a program by taking the top-weighted k rules produced PRA for each relation, for some value of k, and then syntactically transforming them into ProPPR programs, using either the recursive or non-recursive formulation, as described in Table 3.1 and in Table 3.2 respectively. Again, note that the recursive formulation allows us to do joint inference on all the learned PRA rules for all relations at once.

3.4.1 Varying The Size of The Graph

To explore the scalability of the system on large tasks, we evaluated the performance of ProPPR on NELL KB subsets that have $M = 100,000$ and $M = 1,000,000$ entities. In these experiments, we considered only the top-weighted PRA rule for each defined predicate. On the 100K subsets, we have 234, 180, and 237 non-recursive KB rules, and 534, 430, and 540 non-recursive/recursive PRA rules in the Google, Beatles, and Baseball KBs,

Methods	Google	Beatles	Baseball
ProPPR 100K KB non-recursive	0.699	0.679	0.694
ProPPR 100K PRA non-recursive	0.942	0.881	0.943
ProPPR 100K PRA recursive	**0.950**	**0.884**	**0.952**
ProPPR 1M KB non-recursive	0.701	0.701	0.700
ProPPR 1M PRA non-recursive	0.945	0.944	0.945
ProPPR 1M PRA recursive	**0.955**	**0.955**	**0.955**

Table 3.3: Comparing the learning algorithm's AUC among non-recursive KB, non-recursive PRA, and recursive formulation of ProPPR on NELL 100K and 1M datasets.

respectively.[1] On the 1M subsets, we have 257, 253, and 255 non-recursive KB rules, and 569, 563, and 567 non-recursive/recursive PRA rules for the three KBs. We set $\varepsilon = 0.01$ and $\alpha = 0.1$. For example, here queries are in the form of "headquarterInCity(Google,?)", where as positive/negative examples are "+headquarterInCity(Google,MountainView)", and "-headquarterInCity(Google,Pittsburgh)".

First we examine the AUC[2] of non-recursive KB rules, non-recursive PRA and recursive PRA ProPPR theories, after weight-learning, on the 100K and 1M subsets. From Table 3.3, we see that the recursive formulation performs better in all subsets. Performance on the 1M KBs are similar, because it turns out the largest KBs largely overlap. (This version of the NELL KB has only a a little more than one million entities involved in binary relations.) When examining the learned weights of the recursive program, we notice that the top-ranked rules are the recursive PRA rules, as we expected.

In the second experiment, we consider the training time for ProPPR, and in particular, how multithreaded SGD training affects the training time. Table 3.4 shows the runtime for the multi-threaded SGD on the NELL 100K and 1M datasets. Learning takes less than two minute for all

[1]Note that because of the additional recursive rules, the number of rules in non-recursive/recursive case is much larger than using non-recursive only.

[2]Throughout this section, all the AUCs are areas under the ROC curve.

100K

#Threads	Google	Beatles	Baseball
1	54.9	20.0	51.4
2	29.4	12.1	26.6
4	19.1	7.4	16.8
8	12.1	6.3	13.0
16	9.6	5.3	9.2

1M

#Threads	Google	Beatles	Baseball
1	116.4	87.3	111.7
2	52.6	54.0	59.4
4	31.0	33.0	31.3
8	19.0	21.4	19.1
16	15.0	17.8	15.7

Table 3.4: Runtime (seconds) for parallel SGD while training the recursive formulation of ProPPR on NELL 100K and 1M datasets.

the data sets, even on a single processor, and multithreading reduces this to less than 20 seconds. Hence, although we have not observed perfect speedup (probably due to parameter-vector contention) it is clear that SGD is fast, and that parallel SGD can significantly reduce the training time for ProPPR.

3.4.2 Comparing ProPPR and MLNs

Next we quantitatively compare ProPPR's inference time, learning time, and performance with MLNs, using the Alchemy toolkit.[3] We train with a KB with $M = 1000$ entities[4], and test with a KB with $M = 10,000$. The number of non-recursive KB rules is 95, 10, and 56 respectively, and the corresponding number of non-recursive/recursive PRA rules are 230, 29, and 148. The number of training queries are 466, 520, and 130, and the number of testing queries are 3143, 2552, and 4906. We set $\varepsilon = 0.01$ and $\alpha = 0.1$. Again, we only take the top-1 PRA paths to construct ProPPR programs in this section.

In the first experiment, we investigate whether inference in ProPPR is sensitive to the size of graph. Using MLNs and ProPPR non-recursive KB programs trained on the 1K training subsets, we evaluate the inference time on the 10K testing subsets by varying the number of entities in the database used at evaluation time. Specifically, we use a fixed number of test queries, and increase the total number of entities in the KB by a factor of X, for various values of X. In Figure 3.1, we see that ProPPR's runtime is independent of the size of the KB. In contrast, when comparing to MC-SAT, the default and most efficient inference method in MLN, we observe that inference time slows significantly when the database size grows.

In the second experiment, we compare ProPPR's SGD training method with MLNs most efficient discriminative learning methods: voted perceptron and conjugate gradient [61]. To do this, we fixed the number of iterations of discriminative training in MLN to 10, and also fixed the number of SGD passes in ProPPR to 10. In Table 3.5, we show the runtime of various approaches on the three NELL subdomains. When running on the non-recursive KB theory, ProPPR has averages 1-2 seconds runtime across all domains, whereas training MLNs takes hours. When training on the non-recursive/recursive PRA theories, ProPPR is still efficient.[5]

We now examine the accuracy of ProPPR, in particular, the recursive formulation, and com-

[3] http://alchemy.cs.washington.edu/.

[4] We were unable to train MLNs with more than 1,000 entities.

[5] We were unable to train MLNs with non-recursive or recursive PRA rules.

Table 3.5: Comparing the learning algorithm's runtime between ProPPR and MLNs on the NELL 1K subsets .

Method	Google	Beatles	Baseball
ProPPR SGD KB non-recursive	2.6	2.3	1.5
MLN Conjugate Gradident	8604.3	1177.4	5172.9
MLN Voted Perceptron	8581.4	967.3	4194.5
ProPPR SGD PRA non-recursive	2.6	3.4	1.7
ProPPR SGD PRA recursive	4.7	3.5	2.1

Table 3.6: Comparing the learning algorithm's AUC between recursive formulation of ProPPR and MLNs.

Methods	Google	Beatles	Baseball
ProPPR SGD KB non-recursive	0.568	0.510	0.652
MLN Conjugate Gradident	0.716	0.544	0.645
MLN Voted Perceptron	0.826	0.573	0.672
ProPPR SGD PRA non-recursive	0.894	**0.922**	0.930
ProPPR SGD PRA recursive	**0.899**	0.899	**0.935**

pare with MLN's popular discriminative learning methods: voted perceptron and conjugate gradient. In Table 3.6, we see that MLNs outperform ProPPR's using the non-recursive formulation. However, ProPPR's recursive formulation outperforms all other methods, and shows the benefits of joint inference with recursive theories.

We should emphasize that the use of AUC means that we are evaluating only the *ranking* of the possible answers to a query; in other words, we are not measuring the quality of the actual probability scores produced by ProPPR, only the relative scores for a particular query. ProPPR's random-walk scores tend to be very small for all potential answers, and are not well-suited to estimating probabilities in its current implementation.

Table 3.7: AUCs for using top-k PRA paths for recursive formulation of ProPPR on NELL 100K and 1M datasets.

Methods	Google	Beatles	Baseball
ProPPR 100K top-1 recursive	0.950	0.884	**0.952**
ProPPR 100K top-2 recursive	0.954	0.916	0.950
ProPPR 100K top-3 recursive	**0.959**	**0.953**	**0.952**
ProPPR 1M top-1 recursive	0.955	0.955	0.955
ProPPR 1M top-2 recursive	0.961	0.960	0.960
ProPPR 1M top-3 recursive	**0.964**	**0.964**	**0.964**

3.4.3 Varying The Size of The Theory

So far, we have observed improved performance using the recursive theories of ProPPR, constructed from top $k = 1$ PRA paths for each relation. Here we consider further increasing the size of the ProPPR program by including more PRA rules in the theory. In particular, we also extract the top-2 and top-3 PRA paths, limiting ourselves to rules with positive weights. On the 100K datasets, this increased the number of clauses in the recursive theories to 759, 624, and 765 in the Google, Beatles, and Baseball subdomains in the top-2 condition, and to 972, 806, and 983 in the top-3 condition. On the 1M datasets, we have now 801, 794, and 799 clauses in the top-2 case, and 1026, 1018, and 1024 in the top-3 setup. From Table 3.7, we observe that using more PRA paths improves performance on all three subdomains.

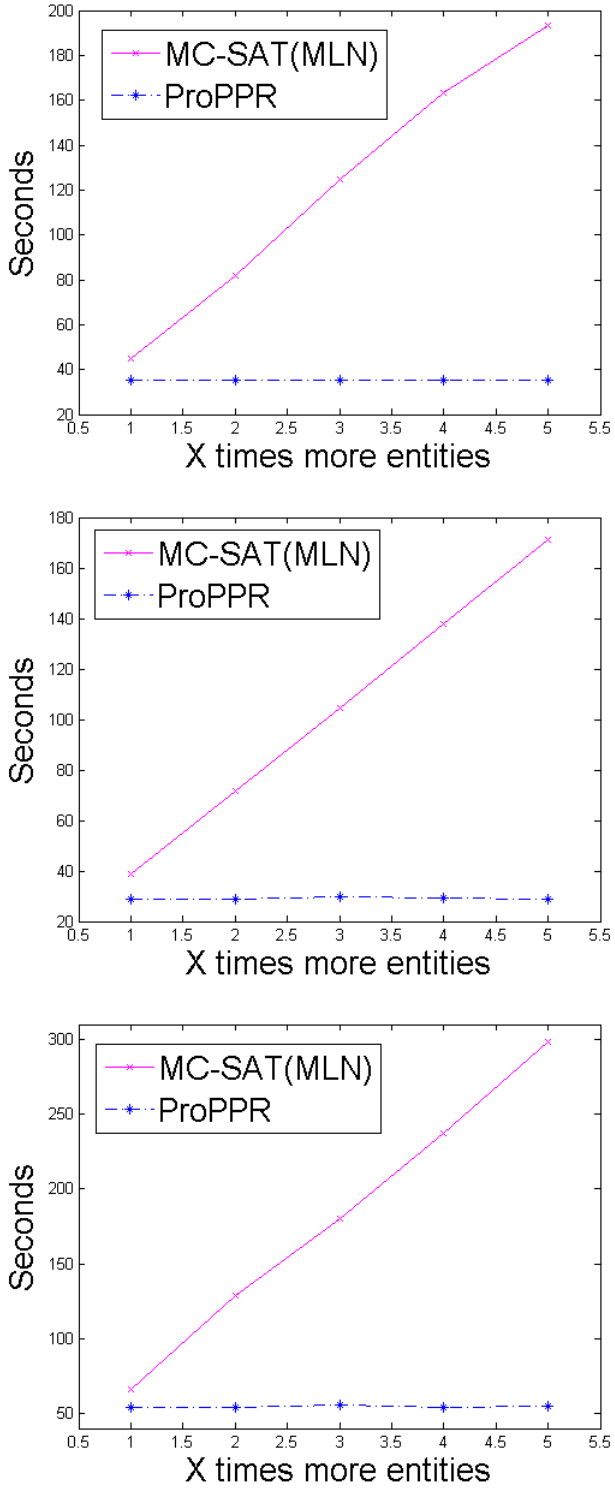


Figure 3.1: Run-time for non-recursive KB inference on NELL 10K subsets the using ProPPR (with a single thread) as a function of increasing the total entities by X times in the database. Total test queries are fixed in each subdomain. Top, the Google 10K dataset; middle, the Beatles 10K dataset; bottom, the Baseball 10K dataset.

Chapter 4

Structure Learning

In this chapter, we describe ProPPR's own scalable structure learning framework: how to efficiently learn first-order logic clauses from data. An early version of this work can be found in [106].

In the previous chapter, we show it is possible to use rules learned by PRA for large scale inference problems using ProPPR. However, this is not ideal, because PRA and ProPPR are two very different relational learning paradigms. Here, we are interested in designing a "native" structure learning method within ProPPR. One major challenge for structure learning in probabilistic logics is the efficiency issue. More practically, many of the existing structure learning methods for learning probabilistic first-order logics are not efficient enough to be used for large, real-world datasets: when the total number of predicates and entities in a database become large, the costs of searching through all possible candidates to construct first-order clauses are also growing rapidly. For example, some existing techniques for learning Markov Logic Networks (MLNs) [83] take days to run [48, 50], even though the input datasets include only a few dozen predicates and a few thousand grounded atoms.

However, there are no existing methods for learning ProPPR theories: previous experiments have used theories learned assuming radically different semantics for the clauses, or hand-written theories.

We present here a structure-learning method for ProPPR based on a recently-learned approach to learning the structure of conventional logic programs, in which logic programs are generated by using a second-order *abductive* program to "prove" that every observed positive example is covered. In abductive reasoning, assumptions are made, as necessary, to complete a proof: in this setting, the assumptions made by the second-order program concern the existence of elements of the first-order program being generated. This approach, which is embodied in a system called Metagol [70], turns out to be both elegant and powerful, providing a conceptually clear framework for such important tasks as predicate invention, and learning recursive programs.

We adapt a Metagol-like approach to learning ProPPR rules. In particular, we present an "abductive" stochastic second-order program for ProPPR, in which every assumption corresponds directly to a useful clause in a ProPPR program, and where further, *every learnable parameter corresponds directly to a first-order clause*. Structure learning is performed by computing the *gradient* of these features on training data, and constructing a small first-order stochastic program, consisting of those clauses that are potentially useful according to the gradient information. Parameters for this first-order program can then be learned in the usual way. We show that on small problems, this approach provides more accurate theories on the task of *knowledge base completion*, where the goal is to learn an interrelated set of rules to infer missing facts in an incomplete knowledge base. The method is also scalable enough to perform knowledge base completion for realistic knowledge bases containing tens of thousands of facts. We then propose an iterative variant of the gradient-guided structure learning approach that incrementally refines the hypothesized space of plausible clauses.

4.1 Structure learning is difficult for KB completion

To illustrate and motivate the problem of structure learning for ProPPR, we will use a classic problem introduced by Hinton in 1986 [41]. In this problem we have two families, each with

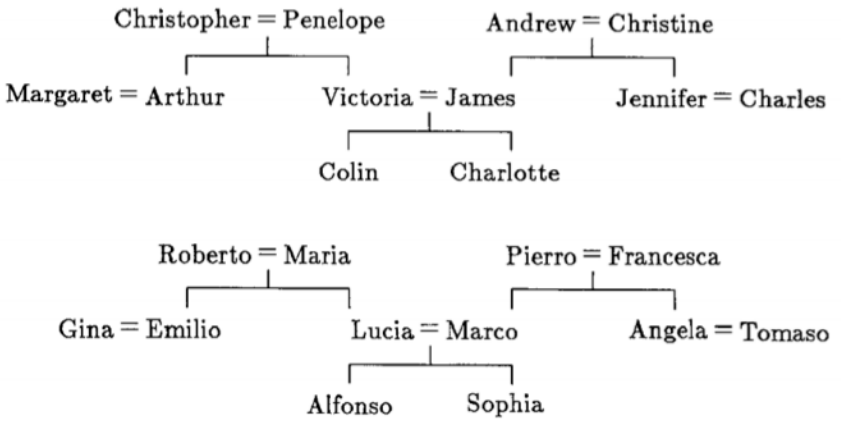


Figure 4.1: The families dataset, from [41].

twelve individuals, and twelve binary relations between these individuals: husband, wife, father, mother, son, daughter, brother, sister, uncle, aunt, nephew, and niece. From this data, we can also define 104 "queries", such as *uncle(charlotte,Y)*, each of which has some positive (correct) answers (e.g.*uncle(charlotte,james)*), and some incorrect answers. In our experiments the universe of potential answers (which we use ProPPR to rank) consists of all person pairs that are related by one of the twelve known predicates. We measure mean average precision (MAP)[1] over all the T_e test queries.

In past experiments, high accuracies have been obtained by holding back a small number of test queries and training on the rest. We confirmed these results with two systems: Quinlan's FOIL [81] and Alchemy with structure-learning [48]. We designated one family as test and one as train, and we then performed 12 experiments where we held out the queries from a single relation: in other words, for relation R, database consisted of facts defining the other 11 relations for both the train and test family; the training data consisted of the queries for R from the train family; and the test data was the queries for R from the test family. FOIL obtained precision

[1]Note that AP not only measures precision at rank k, but also measures recall: it uses the denominator to penalize the cases where positive solutions are missing. MAP has shown to be very robust and stable in many tasks [62], and it is widely used in relation learning tasks (e.g. [85] [53]).

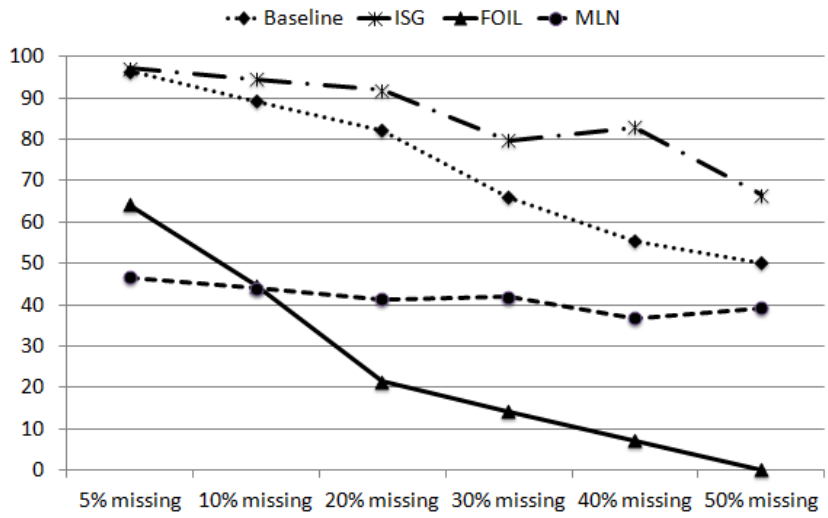


Figure 4.2: Completing an incomplete DB of family relations. X-axis: the percentage of background facts missing. Y-axis: the MAP result.

of 100% for all 12 relations, and Alchemy obtained precision of 100% on 11 of the 12. This is not surprising, since all of the predicates have succinct definitions in terms of the others: for instance, *wife(X,Y)* $\Leftrightarrow$ *husband(Y,X)* (in this data).

However, the prior systems do not perform well if they need to learn interrelated concepts. We held out six pairs of relations (wife/husband, sister/brother, aunt/uncle, niece/nephew, and daughter/son) and repeated the same experiments. Alchemy's mean average precision (MAP) on the six problems drops to 27%, and FOIL's drops to zero. The problem for both systems stems from the use of pseudo-likelihood[2] to estimate the semantics of the partially-learned program from *examples*, rather than actual inference using the learned program. As a typical result, for the relation pair aunt/uncle, FOIL learns the rules *uncle(X,Y) :- husband(X,Z), aunt(Z,Y)* and *aunt(X,Y) :- wife(X,Z), uncle(Z,Y)*, which are circular.

Another illustration of the weaknesses of existing algorithms is provided by the following set of experiments. We trained with a DB containing all but k% of the facts for the training family, and all of the training-family queries as training data: thus, we are asking the system to learn

[2]Or in FOIL's case, an approach broadly similar to pseudo-likelihood.

	Assumption	ProPPR clause
(a)	P(X,Y) :- R(X,Y)	interp(P,X,Y) :- interp0(R,X,Y),abduce_if(P,R).
(b)	P(X,Y) :- R(Y,X)	interp(P,X,Y) :- interp0(R,Y,X),abduce_ifInv(P,R).
(c)	P(X,Y) :- R1(X,Z), R2(Z,Y)	interp(P,X,Y) :- interp0(R1,X,Z),interp0(R2,Z,Y), abduce_chain(P,R1,R2).
(d)		abduce_if(P,R) :- true # f_if(P,R).
(e)		abduce_ifInv(P,R) :- true # f_ifInv(P,R).
(f)		abduce_chain(P,R1,R2) :- true # f_chain(P,R1,R2).
(g)		interp0(P,X,Y) :- rel(R,X,Y).

Table 4.1: The abductive ProPPR program, in the right-most column, along with labels for each rule, and the second-order rules to which they correspond.

rules which can *complete* an incomplete database. As test data, we used a database with all but k% of the facts for the test family, and again, all test-family queries as the test set. The results are shown in Figure 4.2: neither FOIL nor Alchemy's MLN method[3] outperform the simple baseline of predicting exactly the facts in the incomplete database.[4]

4.2 Iterated Structural Gradients for Structure Learning

Motivated by this, we introduce a new technique which we call the *iterated structural gradient* method for structure-learning in ProPPR. In particular, as noted in the introduction, we implement a Metagol-like method for introducing structure. We start with an "abductive" stochastic second-order program, in which every assumption corresponds directly to a useful clause in a first-order program, and where further, every learnable parameter corresponds directly to a first-order clause. Table 4.1 shows the theory that we use: this theory assumes that the first-order DB contains only binary facts, which are encoded for the second-order theory as triples of the form *rel(r,x,y)*: e.g., the fact *father(james,colin)* is represented as *rel(father,james,colin)*. Thus, rule (g) is a second-order version of the baseline algorithm of Figure 4.2: to interpret the predicate $P(X,Y)$, rule (g) simply checks for the fact *rel(P,X,Y)*.

Rules (a-c) can be viewed as a more powerful interpreter for the binary predicate P, which also makes assumptions about the presence of rules in the first-order theory. For instance, in an application of rule (b), the goal of "interpreting" (with the predicate *interp*) the a predicate *uncle(arthur,Y)* is reduced to the goal of interpreting (with the lower-level predicate *interp0*) some predicate *nephew(Y,arthur)*, assuming that the first-order theory contains a clause *uncle(X,Y) :- nephew(Y,X)*.

The way the assumption mechanism is implemented is quite simple. Associated with every interpretive action—i.e., clauses (a-c)—is an extra goal, such as *abduce_ifInv* for clause (b).

[3] Alchemy's performance is quite sensitive to the precise set of missing facts, so we average over ten runs in the figure.

[4] Note that we have also experimented with a more recent "Learning with Structural Motifs (LSM)" variant [50] for learning MLN, but the results were much worse than Alchemy: we only observe a MAP of 10.7 on the missing 5% setting. This is because LSM is designed to learn long clauses (with more than 5 predicates) using recurring short patterns, whereas in our task, our goal is to learn short clauses with a maximum of 3 predicates in a clause. Another issue is that LSM has more than 20 hyperparameters to tune, which makes the structure learning process sensitive to the choice of the datasets and hyperparameters.

These abductive goals are defined to always succeed, but whenever the proof step which lets them succeed is applied, the corresponding edge in the proof graph is labeled with an appropriate feature (e.g., *f_ifInv(uncle,nephew)*) which records that this assumption was made. Table 4.2 gives an example proof in the theory, showing how the abductive feature *f_ifInv(uncle,nephew)* might be used. ProPPR's natural bias towards short proofs (in the second-order theory) guides it toward near-minimal sets of assumptions regarding the first-order theory. Furthermore, *the gradient of the abductive features indicates the utility of the corresponding first-order clauses.*

Goal List	Rule + Features
in(uncle,arthur,Y)	
↓	(b)
in0(R,Y,arthur),ab_ifInv(uncle,R)	
↓	(g)
rel(R,Y,arthur),ab_ifInv(uncle,R)	
↓	DB_1
rel(nephew,colin,arthur), ab_ifInv(uncle,nephew)	
↓	DB_2
ab_ifInv(uncle,nephew)	
↓	(e) + *f_ifInv(uncle,nephew)*
□	

Table 4.2: A slightly-abbreviated sample proof using the second-order theory. The second column is the rule used at that point in the derivation, along with the features generated by that clause application, if any. (For clarity, we list the process of binding *rel(R,Y,arthur)* to the head of the unit clause *rel(nephew,colin,arthur) :-* , and then removing it, as two steps, DB_1 and DB_2.)

Structure learning is performed by computing the *gradient* of these features on training data, and then producing a small first-order stochastic program, consisting of those clauses that are

1. For $t = 1, 2, \ldots$:

   (a) Perform $t - 1$ epochs of parameter-learning on the theory of Table 7.1.

   (b) Compute the gradient of the loss, and for each feature with a negative gradient, add the corresponding clause to the theory:

   - for *f_if(p,q)*, add

     interp0(p,X,Y) :- interp0(q,X,Y).

   - for *f_ifInv(p,q)*, add

     interp0(p,X,Y) :- interp0(q,Y,X).

   - for *f_chain(p,q,s)*, add

     interp0(p,X,Y) :- interp0(q,X,Z),

     interp0(s,Z,Y).

   (c) Stop when no new rules are added.

2. Discard all rules but the added ones, and retrain the parameters for multiple epochs.

Table 4.3: The Iterated Structural Gradient (ISG) Algorithm

potentially useful according to the gradient information. Parameters for this first-order program can then be learned in the usual way.

We thus adopt the learning algorithm of Table 5.3, which we call the *iterated structural gradient* (ISG) method. As an extended example, we consider the operation of ISG on the problem of learning aunt/uncle together. In the first iteration, the following rules are proposed (not in order, and abbreviating *interp0* as *in0*):

in0(aunt,X,Y) :- in0(sister,X,Z), in0(father,Z,Y).

in0(uncle,X,Y) :- in0(brother,X,Z), in0(mother,Z,Y).

in0(aunt,X,Y) :- in0(nephew,Y,X).

in0(aunt,X,Y) :- in0(niece,Y,X).

in0(uncle,X,Y) :- in0(nephew,Y,X).

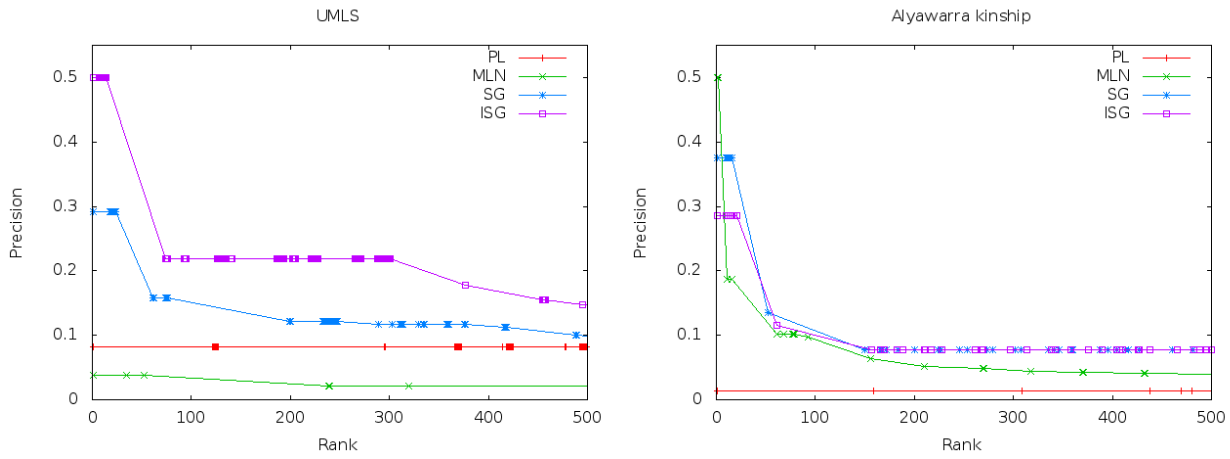


Figure 4.3: Performance on the UMLS and Alyawarra kinship datasets.

in0(uncle,X,Y) :- in0(niece,Y,X).

The first two of these are correct rules, and the remaining ones are over-general, as they confuse aunts and uncles. In the second iteration, ISG proposes the rules:

in0(aunt,X,Y) :- in0(wife,X,Z), in0(uncle,Z,Y).

in0(uncle,X,Y) :- in0(husband,X,Z), in0(aunt,Z,Y).

in0(aunt,X,Y) :- in0(wife,X,Z), in0(aunt,Z,Y).

in0(uncle,X,Y) :- in0(husband,X,Z), in0(uncle,Z,Y).

in0(aunt,X,Y) :- in0(uncle,X,Y).

in0(uncle,X,Y) :- in0(aunt,X,Y).

in0(aunt,X,Y) :- in0(aunt,X,Y).

in0(uncle,X,Y) :- in0(uncle,X,Y)

The first two of these are correct, while the remaining rules are, to various degrees, overgeneral and/or redundant. However, after parameter-learning, the learned theory performs perfectly on the test set.

Figure 4.2 shows the performance of ISG on the database-completion task, and compares it to

	FOIL	PL	MLN	SG	ISG
father+mother	0.0	23.32	42.53	70.05	100.0
husband+wife	0.0	4.73	3.20	39.63	79.4
daughter+son	0.0	11.49	22.74	70.05	100.0
sister+brother	0.0	3.29	10.37	62.18	78.85
uncle+aunt	0.0	10.41	53.35	79.41	100.0
niece+nephew	0.0	6.49	28.54	72.25	80.09
average	0.0	9.96	26.79	65.60	89.70

Table 4.4: Average precision performance for learning two mutually-related relations at once.

FOIL, MLNs with structure learning, and the KB-only baseline. Table 4.4 shows results for the leave-two-relation out experiments discussed above. We also introduce two additional baselines for comparison: one is to perform the main loop only once, which we call the *structural gradient* (SG) method, and a final baseline is parameter-learning for the second-order theory, which we label PL in the table. Note that ISG performs quite well on the task of learning two interrelated predicates, even though it is quite difficult for the other systems (e.g. FOIL and MLNs). In addition to this, the ISG method usually converges quickly: empirically it typically converges within only 5 iterations.

Our method is superior because instead of using pseudolikelihood, we take a holistic point of view: we use a second-order abductive logic to construct the hypothesis space, and relax the structure learning problem to first-order parameter learning using supervised personalized PageRank with log likelihood. It is not surprising that the proposed ISG method has a good performance: the ISG method incrementally adds newly-learned gradient-guided plausible inference rules to the structure learning rule set, which helps to refine the overall multi-epoch structure learning process. Additional structure learning results can be found in our prior work [106].

	KB seed	
	Google	*baseball*
top 1k entities		
#train/test queries	100	100
#DB facts	853	890
#rules learned	20	15
train time (sec)	29	19
top 10k entities		
#rain/test queries	1000	1000
#DB facts	10630	11972
#rules learned	401	392
train time (sec)	62	65
top 100k entities		
#train/test queries	5000	5000
#DB facts	12902	9746
#rules learned	1094	939
train time (sec)	354	276

Table 4.5: Summary of the KBs used in experiments on completing subsets of NELL's KB.

4.3 Results: Learning inference rules for the NELL knowledge base

Finally, as a larger-scale and more realistic task, we explore using ProPPR's structure learning method to induce inference rules for the NELL knowledge base.

Following Chapter 3, we used a number of varying-sized versions of the NELL knowledge base (KB): specifically we took KBs containing 1,000, 10,000 and 100,000 entities, centered around two NELL concepts, "Google" and "baseball". We took M NELL queries from these KBs

to use for training queries, and a disjoint M to use for testing queries. All facts not associated with these queries were used as the database. We used $M = 5000$ for the 100k-entity KBs, $M = 1000$ for the 10k-entity KBs, and $M = 100$ for the 1k-entity KBs[5]. Note that the baseline method of predicting using the database would, by construction, achieve an average precision of 0% on these test sets.

The performance of ISG on these tasks is shown in Figure 5.1 and Table 5.4. (Runtime is using 20 threads for learning on a conventional desktop machine.) Even though the data is noisy, ISG learns large and useful theories—theories such that the high-confidence predictions do indeed correspond, in most cases, with facts actually in the NELL knowledge base.

4.4 Related Work for Structure Learning

Our overall approach builds on Metagol [70], but differs in many respects: most notably, unlike Metagol, our system learns probabilistic programs in ProPPR, rather than "hard" Prolog programs. ProPPR's use of the (approximate) PageRank-Nibble-based proof method leads to many other differences: for instance, to learn recursive programs, Metagol requires a well-founded ordering of the Herbrand base to prevent infinite loops, while ProPPR does not; also, Metagol uses iterative deepening search to find a minimal set of abductions for each positive example, an NP-hard problem in the worst case, while ProPPR's proof methods instead find a large set of approximately minimal abductions. Metagol has been recently extended to learn a certain class of probabilistic programs [71], but has not been used to learn recursive theories of the size considered in this thesis.

Experimentally, our experimental comparisons focus mainly on the widely-adopted MLN formalism—in particular the approach described by Kok and Domingos [48] which uses a beam search, coupled with pseudo-likelihood based parameter estimation, to learn MLNs. As noted

[5]On these problems, we were not able to successfully run MLN's structure-learning, even with only 1,000 entities.

above, use of pseudo-likelihood, while efficient, causes problems in learning multiple related predicates, the specific task addressed here, and other more recently-proposed MLN structure-learning schemes (e.g., [43, 50, 64]) do not appear to address this issue. In our experiments, we have also investigated the performance of LSM [50] on our datasets, but it fails to outperform the existing pseudo-likelihood based MLN structure learning algorithm in Alchemy. We hypothesize that the reason may be that LSM is a variant for learning long clauses for MLNs by examining short recurring patterns (aka Structural Motifs), thus when motifs are too short, the benefits of LSM may not retain. Another issue we encounter with LSM is the complexity of the setup: it includes more than 25 hyperparameters, making the process of adapting LSM to new problems very difficult.

The Alchemy implementation of structure-learning for MLNs that we used in our experiments is well-documented and stable, but is based on an arguably suboptimal inference substrate. Faster inference schemes (e.g., [75, 87, 91, 92] could possibly support structure-learning approaches that do not rely on the pseudo-likelihood approximation. Although these faster MLN inference methods have not yet been incorporated into structure-learning systems, integration of these lines of work is a plausible alternative to the approach we have described and experimentally tested here. We note, however, there is no theoretical analysis of the complexity of these methods, and experiments with both FROG [87] and LazySAT [92] suggest that unlike ProPPR they still lead to a groundings that grow with DB size, albeit more slowly; we also caution that heuristic inference speedups based on hand-coded MLNs need not necessarily transfer well to MLNs generated automatically.

Our work also builds heavily on Lao et al's Path Ranking Algorithm (PRA) [53, 54], which supports structure learning; however, PRA can learn only a very limited type of program (roughly, disjunctions of non-recursive chains of binary predicates). The underlying ProPPR logic used here can be viewed as combining ideas from PRA with stochastic logic programs (SLPs) [26]. Relative to SLPs, ProPPR adds a restart to the random-walk process, and the addi-

tion of a more flexible scheme for featurizing the logic. The featurization scheme was heavily used in the structure-learning proposed in this thesis: in fact, the ability to attach arbitrary feature sets, computed on-the-fly at proof time, appears to be unique to ProPPR, among first-order logics. To the best of our knowledge, SLPs have not been coupled with structure-learning methods.

ProbLog [27] is an alternative probabilistic logic programming formalism, which grounds probabilistic program by converting the space of possible proofs to a binary decision diagram (BDD), which can be very large, in the worst case, for recursive programs. There has been some prior work on learning BDD-based probabilistic programs, however; in particular, they have been used as the substrate for structure learning in Bellodi and Riguzzi's systems SLIPCASE [10] and SLIPCOVER [11]. In these systems, beam search is used to explore a space of probabilistic logic programs, and candidate programs are scored by running a small number of iterations of EM. In past experiments, SLIPCASE has been run with either a small depth bound (e.g., three), or else limited to non-recursive theories, so it is not clear that it will perform well on the larger mutually-recursive programs considered here. Some limited comparison can be seen in prior work [105], which evaluates ProPPR on the WebKB dataset: ProPPR obtained an AUC of 0.80 here with a simple fixed theory, compared to 0.61 for ProbLog with the same fixed theory, or 0.76 for SLIPCOVER's learned theory. However, experimental comparisons with SLIPCASE and similar systems remains a topic for future work. The BDD-based approach seems especially promising in conjunction with approximate reasoning methods [100], but to our knowledge, these have not been integrated with structure-learning approaches.

Chapter 5

Soft Predicate Invention via Structured Sparsity

In this chapter, we revisit a classic task in statistical relational learning—predicate invention, where we present a novel interpretation by connecting predicate invention to structured sparsity regularization techniques in machine learning. This chapter is based on a conference paper recently accepted to IJCAI 2015.

In relational learning, *predicate invention* (PI) is a method in which new predicates are introduced into a logical theory. Most PI techniques simplify a logical theory by allowing a group of closely-related rules to be combined somehow, using the invented predicate.

PI may be viewed from a sparse structure learning perspective: early PI algorithms from the inductive logic programming community often leverage similar patterns from first-order logic representations, and then invent new predicates to compress the first-order formulas to form compact theories. For example, in learning logical rules for the domain of family relations, one might have learned the following clauses:

$$daughter(X,Z),\ father(Z,Y) \Rightarrow sister(X,Y)$$

$$daughter(X,Z),\ mother(Z,Y) \Rightarrow sister(X,Y)$$

A PI system like CHAMP [47] would create a new predicate by combining these similar rules: e.g., it might invent a predicate "*parent*", along with an appropriate definition (as the disjunction of *father* and *mother*), and then compress the above clauses into[1]:

$$daughter(X,Z),\ parent(Z,Y) \Rightarrow sister(X,Y)$$

The difficulty with applying such PI invention methods is that they are somewhat prone to errors when data is noisy. Past approaches to PI have avoided these problems by some combination of clean data and computationally-intensive search[2] over structure space [46, 49].

Intuitively, PI is motivated by the principle that parsimonious explanations of the data are likely generalize well. A similar bias towards parsimonious theories is made by methods such as Lasso [98], which "sparsify" a model by pushing the weights for some features to zero. Sparsity-encouraging regularization methods are a common tool in analyzing complex, high-dimensional datasets, and have been useful in domains including text classification [30] and vision [77, 115]. Sparsity-encouraging regularization methods are often viewed as "softer" substitutes for feature selection.

It is natural to conjecture that sparse regularization might make PI more robust, by effectively removing "noisy" invented predicates. Notice, however, that an invented predicate P couples the performance of all rules that use P: in the example above, for instance, the performance of the *parent* rules are coupled to the performance of the *sister* rule that calls it. This suggests that *structured sparsity* methods such as the group Lasso [32, 118] might be more useful for PI.

Further reflection suggests another connection between structured sparsity and PI. Consider

[1] We use the predicate symbol "parent" for clarity–a real invented predicate would have a meaningless name, like *invented16*.

[2] We note that structure search is especially expensive in probabilistic logics, where inference is generally non-trivial.

the set of rules that would be simplified by an invented predicate. PI compresses a theory by replacing this set with a smaller one, thus reducing the number of parameters to learn. A structured sparsity regularizer that regularizes together the parameters for this set also reduces the number of parameters to learn, in a very analogous way. We call this approach *soft PI*. Soft PI does not explicitly creating new symbols to compress the existing theory: instead, soft PI relies on modifying the learner, via a regularizer, to exploit the same commonalities.

In this thesis, we explore the connections between PI and sparsity-encouraging regularizers. More specifically, we use the iterated structural gradient (ISG) approach [106] to identify potentially useful rules. We then apply CHAMP-like heuristics to identify groups of clauses that could be compressed with invented predicates. We compare "hard PI" methods, in which the invented predicates are actually introduced, with "soft PI", in which we impose a group Lasso penalty regularization term to learn parameters for a final set of clauses, with structured sparsity. We compare these approaches with non-structured sparse and non-sparse regularizers, as well as an alternative structured sparsity regularizer, namely a sparse graph Laplacian regularization exploits pair-wise relationships between rules.

Our approach is built on top of ProPPR [105] The methods we explore here are highly scalable: when using a parallel stochastic gradient descent learner with lazy proximal structured sparsity updates, learning takes only a few minutes to process 20,000 examples against a 10,000-tuple database. We also scale the group Lasso approach to a version of the NELL [18] KB with 100K facts, while achieving good performances on the "long tail" of inferences in the KB.

5.1 Hard Predicate Invention

In previous work [106] involving the structural gradient method, the space of rules $\mathscr{R}$ includes rules over a fixed set if predicate symbols that are known in advance. In some cases it is useful to invent new predicates and define them. For instance, in learning a definition of *aunt* using the pre-defined predicates *mother*, *father*, a system might include the rules:

aunt(X,Y) :- sister(X,Z),mother(Z,Y).

aunt(X,Y) :- sister(X,Z),father(Z,Y).

A potentially more compact definition for *aunt* might be found by inventing the new predicate *invented1* and defining it as the disjunction of *mother* and *father*. Constructing and defining new predicate symbols in this way is called predicate invention (PI).

Existing PI approaches involve creating new predicates based on similarities [114] and differences [69] between learned rules. However, many PI systems are not robust enough to handle noisy data, as when incorrect predicates are invented, errors may eaily cascade. In this thesis we evaluated several variations of a CHAMP-style analysis [47] to invent predicates based on sets of similar rules. We consider pairs of rules to be similar if they have the following format.

- R1 is "p(X,Y) :- q(X,Z),r(Z,Y)" and R2 is "p(X,Y) :- q(X,Z),s(Z,Y)", i.e., they are length-two chains that differ only in the last predicate of the RHS, or
- R1 is "p(X,Y) :- q(X,Y)" and R2 is "p(X,Y) :- s(X,Y)", i.e., they are length-one chains that differ only in the last predicate of the RHS, or
- R1 is "p(X,Y) :- q(Y,X)" and R2 is "p(X,Y) :- s(Y,X)", which is the inverse relation case.

In our experiments on learning family relations (detailed settings and quantitative results will be shown in Section 5.3.), the results that "hard PI" produced are as follows:

nephew(X,Y) :- invented1(Y,X).

invented1(X,Y) :- uncle(X,Y).

invented1(X,Y) :- aunt(X,Y).

uncle,X,Y) :- invented2(Y,X).

invented2(X,Y) :- nephew(X,Y).

invented2(X,Y) :- niece(X,Y).

aunt(X,Y) :- sister(X,Z),invented1(Z,Y).

sister(X,Y) :- niece(X,Z),invented1(Z,Y).

...

brother(X,Y) :- invented1(X,Y).

uncle(X,Y) :- uncle(X,Z),invented1(Z,Y).

Although the majority of the compressed rules produced by hard PI are intuitively meaningful, the last two rules are not.

5.2 Soft Predicate Invention via Structured Sparsity

From the example in the previous section, we see that a drawback of hard predicate invention is that, given noisy inputs, incorrect clauses may be generated. In this section, we present a structured sparsity based learning approach for soft predicate invention: instead of creating new symbols, our approach groups similar concepts together, and exploit regularization-based structured sparsity technique to induce soft predicates.

A key issue that we observe from the problem is that, ProPPR's default regularization term $R(\mathbf{w}) = \mu||\mathbf{w}||_2^2$, which is the Ridge estimator [56], will not produce sparse estimates. The noisy estimates may not be ideal: the incorrect first-order logic program may lead to more errors in the downstream applications. Nishino et al. [74] is among the first to study a projected gradient

approach for learning sparse parameters in relational parameter learning, but the problems of learning sparse structures and predicate invention are not discussed. To solve this issue, we consider the following Lasso [98] formulation, where objective function is:

$$\min\Big(-\ell+\mu||\mathbf{w}||_1\Big)$$

Unlike the Ridge estimator, the above Lasso penalty will now produce sparse estimates, even though the objective function is now non-differentiable. To optimize the above function, instead of using a subgradient approach, we use a proximal operator: each weight component w is shrunk towards 0 by a shrinkage value σ,

$$signum(w)\cdot max(0,|w|-\sigma)$$

where in our lazy L_1 regularization update is

$$\sigma=\delta\sqrt{2}\mu\beta.$$

here, δ is the total number of accumulated regularization update. Note that the update cost for regularization is high, especially when the features are growing. To mitigate this issue, we use the following Lazy L_1 update algorithm in Table 5.1, inspired by a recent work on lazy stochastic gradient descent optimization [19]. The main idea is that, the regularization updates for all features in each example are slow and unnecessary, and we can cache the regularization updates of relevant features, then update the accumulated regularization changes.

One challenge associated by PI is that the reuse of invented predicates makes them hard to remove by simple element-wise regularizers. In addition to this, even though the above Lasso update will produce sparse logic program, it does not explore the structural similarity among each logic clauses via soft PI. To better incorporate the dependencies among the features in each logic clause, we introduce a pair-wise graph Laplacian penalty [8] for $R(\mathbf{w})$:

$$\min\Big(-\ell+\zeta\sum_{(p,q)}||w_p-w_q||A_{(p,q)}\Big)$$

Here, ζ is the regularization coefficient that controls the strength of the structured penalty, and $A_{(p,q)}$ is an adjacency matrix that indicates the pair-wise similarity among logic clauses, and the basic idea is to push similar logic clauses to have similar weights after learning. For example, consider the following clauses:

sister(X,Y) :- daughter(X,Z), father(Z,Y).

sister(X,Y) :- daughter(X,Z), mother(Z,Y).

Since they share the same goal and the same first predicate on the right hand side (RHS), it make sense for them to have similar weights. To construct the sparse A matrix , we use a CHAMP-style analysis [47]: for all pairs of clauses that share the same goal and have $|RHS| = 1$ cases, we connect them in the adjacency graph. For all pairs of clauses that share the same goal and have $|RHS| = 2$ cases, if the first predicates on the RHS are the same, we connect the two clauses together in A. Note that this approach not only considers the syntactic similarity among clauses, but also the semantics: for a pair of clauses to be similar, they have to share the same proof goal. For instance, instead of inventing a hard predicate rule, we instead assign a weight value of 1 for this pair of rules in the affinity matrix A.

We then define a degree matrix D to be the total number of connections for each entry in A, and a graph Laplacian $L = D - A$, which transforms the Laplacian regularization as:

$$\min\Big(-\ell + \zeta \mathbf{w}^T L \mathbf{w}\Big)$$

Now since the L is symmetric and positive semi-definite, we can view this $R(\mathbf{w})$ term as a quadratic penalty. To incorporate sparsity, we may also consider this alternative sparse Laplacian regularization formula:

$$\min\Big(-\ell + \mu||\mathbf{w}||_1 + \zeta \mathbf{w}^T L \mathbf{w}\Big)$$

A problem with Laplacian regularization is that while it pushes similar pairs of clauses to have similar weights, it will not typically remove groups in incorrect related clauses by pushing their

weights to zero. To solve this problem, we introduce a sparse group Lasso [32, 118] formulation:

$$\min\Big(-\ell+\mu||\mathbf{w}||_1+\zeta\sum_{l=1}^{g}||\mathbf{w}_l||_2\Big)$$

where g is the total number of feature groups. This is now a structured sparsity learning problem, where we can introduce group sparsity. To generate the groups, we utilize the same A matrix in the Laplacian regularization: each row in A corresponds to a feature group, and different groups may have overlapping features. The benefit of using sparse group Lasso is that it will drive the weights for the entire group of features to zero if the group is not useful or noisy, which appears to be critical for soft PI. Again, we take the first-order derivatives of the group Lasso term, and use the same proximal operator algorithm to solve the sparse group Lasso optimization.

5.3 Experiments

In this section, we evaluate the effectiveness of the proposed approach on two datasets: a new, large family relation dataset[3], which features the Kings and Queens of Europe, including Great Britain's royal family up to 1992; as well as the NELL subsets that include up to 100K grounded facts extracted from the Web. In particular, we focus on the task of structure learning for *knowledge base completion* [25, 106], where the goal is to learn first-order logic program to reconstruct the KB, given only partially complete background database.

5.3.1 Learning Family Relations

We introduce a new dataset for research in SRL[4]. The original dataset was created in 1992 by Denis R. Reid, including 3010 individuals and 1422 families of European royalty. We parsed the genealogical data to extract six pairs of inter-related family relations: {*uncle & aunt, sister*

[3]This is motivated by Hinton's classic kinship dataset, which includes only two families, each with twelve individuals, and twelve binary relations between these individuals.

[4]http://www.cs.cmu.edu/~yww/data/family_data.zip

& brother, daughter & son, father & mother, husband & wife, niece & nephew}. (Learning such pairs of relations has proven to be quite difficult for structure-learning systems in past work [106].) We use a temporal split to separate the train and test subsets. The training set includes 21,423 facts, while the test set contains 8,894 facts. In our KB completion experiment, we randomly delete 50% of the background facts for both the training and test sets respectively, and we use soft PI to complete the missing facts, and evaluate the effectiveness of our approach using Mean Average Precision (MAP).

Table 5.2 and Table 5.3 show the MAP results for KB completion on the royal family dataset, using the non-iterated and iterated structural gradient variants [106] respectively. We see that for both cases, hard PI performs poorly, due to the error cascades.[5] Traditional element-wise non-sparse and sparse methods, namely, the Ridge and Lasso estimators, did help improving the test performance for both settings. The pair-wise graph Laplacian regularization does not have a strong evidence that it improves the performance. This is not surprising, because by forcing similar logic clauses to learn similar weights, there is no guarantee that the final predictive accuracy will be improved. We observe that for soft PI, the proposed sparse overlapping group Lasso method do have strong gain on this task: on both settings, it outperforms all the competitive baselines by a large margin. In general, we also see the advantage of soft PI for inference with probabilistic logic programs— they tend to lead to better predictive performances than hard PI or no PI solutions.

5.3.2 Completing the NELL KB

Finally, as a larger-scale and more realistic task, we explore learning inference rules for the NELL knowledge base. NELL is given as input an ontology that defines hundreds of categories (e.g., person, beverage, athlete, sport) and two-place typed relations among these categories (e.g., *athletePlaysSport(Athlete, Sport)*), which it must learn to extract from the web. We use the

[5]Hard PI here is the best of several PI techniques we experimented with.

datasets from Chapter 4, and compare with their ridge method. The summary of the dataset is shown in Table 5.4.

Inference on NELL's learned KB is challenging for two reasons. First, the learned KB is not only incomplete, but also noisy, since it is extracted imperfectly from the web. For example, a football team might be wrongly recognized as two separate entities, one with connections to its team members, and the other with a connection to its home stadium. Second, the inference problems are large.

The performance of soft PI via sparse group Lasso on these tasks is shown in Figure 5.1. Even though the data is noisy, we see that the overlapping sparse group Lasso learns large and useful theories—theories such that the high-confidence predictions do indeed correspond, in most cases, with facts actually in the NELL knowledge base. Comparing to the Ridge estimator, our proposed method is better at modeling the long-tail distribution of facts on all of the "Google" and "Baseball" subsets in the NELL KB.

1. Let $k = 0$, and let H and $\mathbf{w}$ be empty hashtables of features from each logic clause. H will record the value of k last time $\mathbf{w}[j]$ was updated.
2. For each pass $t = 1, \ldots, T$
   - For each example $\mathbf{x}_i, y_i$:
   - Let $k = k + 1$
     - For each non-zero feature of $\mathbf{x_i}$ with index j and value x_j:
       - If j is not in $\mathbf{w}$, set $\mathbf{w}[j] = 0$.
       - If j is not in H, set $A[j] = 0$.
       - Simulate the "regularization" updates that would have been performed for the $\delta = k - A[j]$ examples since the last time a non-zero x_j was encountered by setting
       $$\mathbf{w}[j] = signum(\mathbf{w}[j]) \cdot max(0, |\mathbf{w}[j]| - \sigma)$$
       - Set $\mathbf{w}[j] = \mathbf{w}[j] + \beta(y - p)x_i^j$
       - Set $A[j] = k$
3. For each parameter $\mathbf{w}_1, \ldots, \mathbf{w}_d$, set
$$\mathbf{w}[j] = signum(\mathbf{w}[j]) \cdot max(0, |\mathbf{w}[j]| - \sigma)$$
4. Output the parameters $\mathbf{w}_1, \ldots, \mathbf{w}_d$.

Table 5.1: The Lazy Proximal Lasso Algorithm for Learning Sparse Structures.

Methods	ReLU	#c	tanh	#c
Hard Predicate Invention	70.47	80	71.60	80
No Predicate Invention	76.03	101	78.02	101
+ Ridge	76.72	101	78.17	101
+ Lasso	77.25	94	78.76	94
Soft Predicate Invention				
+ Laplacian	75.41	101	77.55	101
+ Group Lasso	77.47	78	77.73	79
+ Sparse Laplacian	77.30	94	78.75	94
+ Sparse Group Lasso	**81.14**	61	**81.55**	59

Table 5.2: The MAP results from non-iterated structural gradients for KB completion on the royal family dataset. #c: the number of logic clauses with non-zero weights.

Methods	ReLU	#c	tanh	#c
Hard Predicate Invention	76.41	85	71.98	98
No Predicate Invention	78.49	121	80.49	121
+ Ridge	79.12	121	81.22	121
+ Lasso	79.58	111	81.63	111
+ Soft Predicate Invention				
+ Laplacian	78.31	121	81.19	121
+ Group Lasso	79.30	94	80.67	96
+ Sparse Laplacian	79.28	111	81.33	111
+ Sparse Group Lasso	**81.03**	73	**82.68**	70

Table 5.3: The MAP results from iterated structural gradients for KB completion on the royal family dataset. #c: the number of logic clauses with non-zero weights.

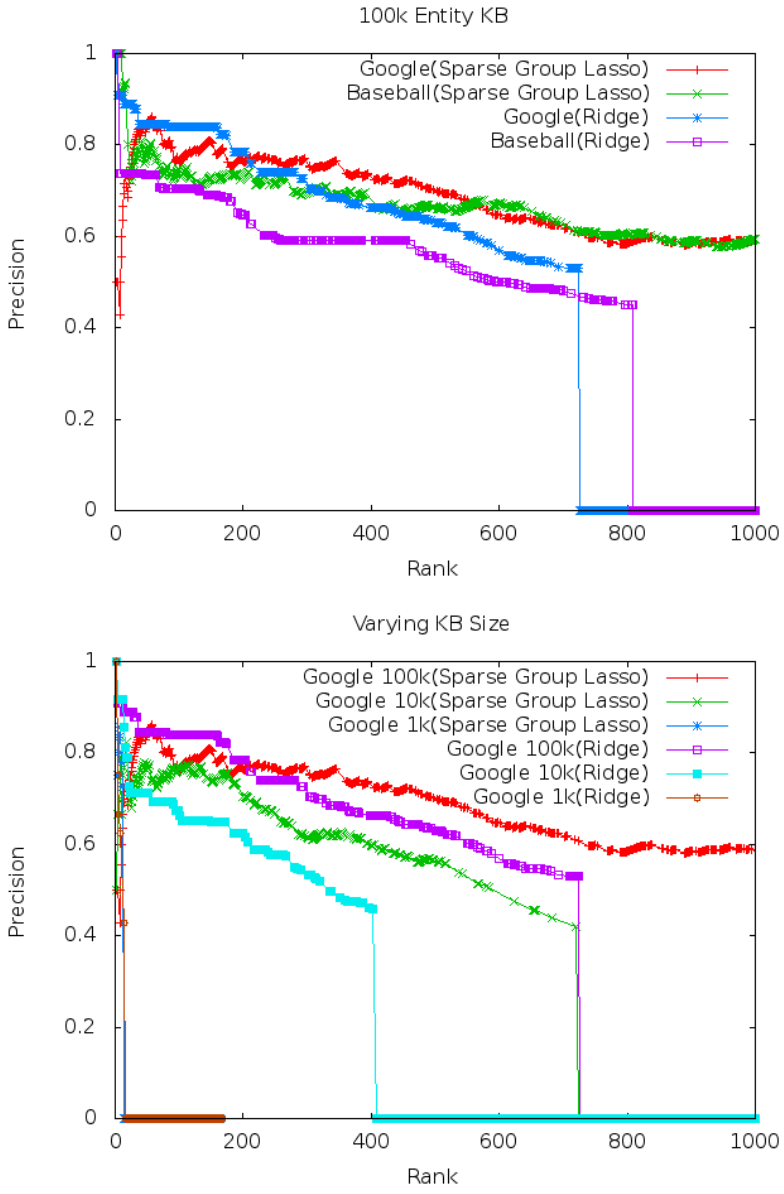


Figure 5.1: Performance on completing subsets of the NELL KB. Top, interpolated precision vs rank for two KBs with 100k entities; bottom; comparison on three KBs of different sizes based on the seed "Google".

	KB seed	
	Google	*baseball*
top 1k entities		
#train/test queries	100	100
#DB facts	853	890
top 10k entities		
#train/test queries	1000	1000
#DB facts	10630	11972
top 100k entities		
#train/test queries	5000	5000
#DB facts	12902	9746

Table 5.4: Summary of the KBs used in experiments on completing subsets of NELL's KB. Note that we also use a temporal split to create the train/test split for this experiment.

Chapter 6

Matrix Factorization for Parameter Learning

6.1 Introduction

In recent years, learning low-dimensional embeddings becomes very successful in representing language, entities, and relations. For example, Mikolov et al. [65] have developed Word2Vec: a tool that allows one to learn the multidimensional embeddings of words and phrases from large corpora. When working with relational data, Bordes et al. [14] find that learning continuous embedded vectors for relations together with head and tail entities improves knowledge base completion [13]. In addition to effectiveness, their approaches are also efficient enough to deal with very large knowledge bases.

In this chapter, we explore the possibility of representation learning as parameter learning for ProPPR. To build on the existing results of structure learning in Chapter 4, we use representation learning to learn low-dimensional first-order logic embeddings from scratch. More specifically, we are interested in learning latent, distributional representations of parameters. In the context of structure learning, we are essentially learning logical formulas to enhance logic-based knowledge base completion for large datasets. To start with, we first use structural gradient

approach of Chapter 4 to identify a set of plausible formulas from knowledge bases. Then, we use ProPPR [105] to produce a grounded proof graph using these logical formulas, training examples, and the background knowledge base. In ProPPR's proof graph, the activated first-order logic formulas become the edges, and the nodes are intermediate states of the proof. To perform representation learning on the formulas, we map the proof graphs to a binary matrix, and use a scalable low-rank approximation based framework to learn the embeddings of the examples and formulas. Finally, we transform these learned embeddings to the parameter space, and enable first-order logic inference with formula embeddings. In experiments, we evaluate the proposed method on a Freebase 15K dataset and a WordNet dataset with hundreds of thousands of facts, and demonstrate the advantage of reasoning with first-order logic embeddings. Our main contributions are two-fold:

- We propose a practical algorithm of learning first-order logic embeddings for enhancing statistical relational learning;
- We demonstrate that the proposed approach can be scaled to reasoning tasks on two large knowledge bases, outperforming various strong and state-of-the-art baselines.

6.2 Related Work

Our work is closely related to recent studies of learning knowledge graph embeddings. RESCAL [72] is among the first to consider tensor decomposition for the task of learning entity and relation embeddings. They use a rank-3 tensor to represent relations, head, and tail entities, and deploy a least squares training method. A number of energy based approaches for learning relational embeddings have also been proposed [13, 15, 44], showing strong performances. Recently, Bordes et al. [14] introduce a scalable method for translating embeddings in knowledge base completion, and together with its variants [59, 109], they achieve the state-of-the-art results in this task. With the revival of neural network methods, Shi and Zhu [88] have investigated

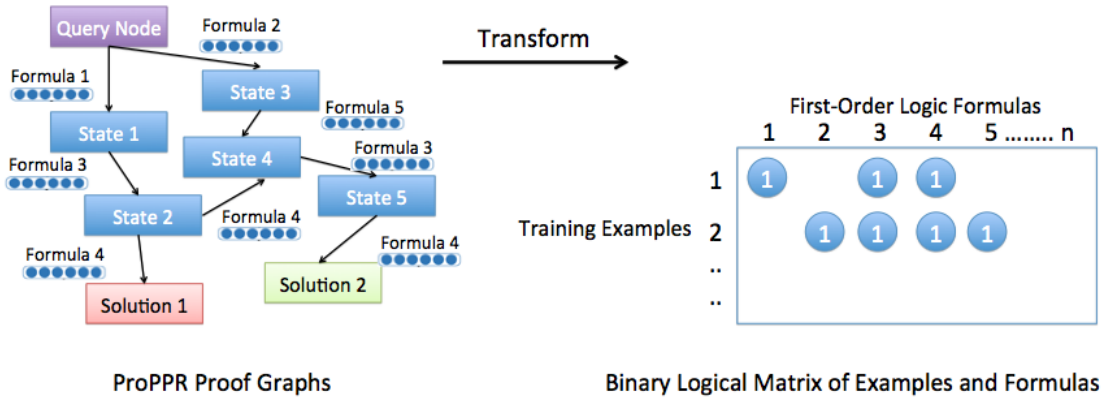


Figure 6.1: The Matrix Factorization Framework for Learning First-Order Logic Embeddings.

a deep Convolutional Neural Networks approach for learning relational embeddings. Although these approaches have considered learning embeddings for entities and relations, none of the above have actually investigated the possibility of learning first-order logic embeddings.

There has been recent studies [58, 101] utilizing logic-like inference paths as constraints for learning entity and relation embeddings. Guu et al. [40] leverage embedding learning techniques and path constraints for path query answering. Wei et al. [110] learns knowledge base embeddings, and then use MLNs as a postpass for ranking the instances. Our work also aligns with an attempt of using entailment based first-order constraints [86] for universal schema based knowledge base completion [85]. To the best of our knowledge, we are among the first formal study to investigate the problem of learning low-dimensional first-order logic embeddings from scratch, while scaling formula embeddings based probabilistic logic reasoning to large knowledge bases.

6.3 Our Approach

We now describe the technical details of our approach. First, we explain how we formulate the representation learning task. Then, we introduce a scalable approach for learning low-dimensional embeddings of first-order logic formulas.

6.3.1 Problem Formulation

In this work, we propose a novel matrix factorization approach to learning first-order logic embeddings. Figure 6.1 shows an overview of the framework.

More specifically, we first use ProPPR's structural gradient method [106] to compute a set of candidate formulas from knowledge bases. We then use this set of formulas, background knowledge base, and training examples to produce standard ProPPR proof graphs (see the left part of Figure 6.1 for an example). In ProPPR's grounding mechanism, we start from a query node (training example), and incrementally apply each first-order logic formula to an edge as the proof process proceeds. The process will stop once it has reached a solution node, and now those activated formulas will be on the edges of this proof graph and the nodes will become the intermediate states of the proof.

We formulate this first-order logic formula embedding learning task as a matrix completion problem. Given a collection of proof graphs, we assume that there are m total examples, which are the rows in the our matrix. As for the columns, each column represents a first-order logic formula. The total number of columns in the input matrix is n. Our matrix F now encodes training examples, and the activated first-order logic formulas for each example. Here we use i to index the i-th example and j to index the j-th formula. The formulas take only binary values—either they are used in the proof graph for this example or not.

6.3.2 Low-Rank Approximation

Since the Netflix competition [9], collaborative filtering techniques with latent factor models have had huge success in recommender systems. These latent factors, often in the form of low-rank embeddings, capture not only explicit information but also implicit dependencies from the input data. Following prior work [51], we are interested in learning two low-rank matrices $\mathbf{P} \in \mathbf{R}^{k \times m}$ and $\mathbf{Q} \in \mathbf{R}^{k \times n}$. The intuition is that $\mathbf{P}$ is the embedding of all examples, and $\mathbf{Q}$ is the embedding of first-order logic formulas.

Here k is the number of latent dimensions, and we would like to approximate $F_{(i,j)} \simeq \vec{p}_i{}^T \vec{q}_j$, where $\vec{p}_i$ is the latent embedding vector for the i-th example and $\vec{q}_j$ is the latent embedding vector for the j-th column. We seek to approximate the matrix F by these two low-rank matrices $\mathbf{P}$ and $\mathbf{Q}$. We can then formulate the optimization problem for this task:

$$\min_{\mathbf{P},\mathbf{Q}} \sum_{(i,j)\in F} (F_{(i,j)} - \vec{p}_i{}^T \vec{q}_j)^2 + \lambda_P ||\vec{p}_i||^2 + \lambda_Q ||\vec{q}_j||^2$$

here, λ_P and λ_Q are regularization coefficients to prevent the model from overfitting. To solve this optimization problem efficiently, a popular approach is stochastic gradient descent (SGD) [51]. In contrast to traditional methods that require time-consuming gradient computation, SGD takes only a small number of random samples to compute the gradient. SGD is also natural to online algorithms in real-time streaming applications, where instead of retraining the model with all the data, parameters might be updated incrementally when new data comes in. Once we have selected a random sample $F_{(i,j)}$, we can simplify the objective function:

$$(F_{(i,j)} - \vec{p}_i{}^T \vec{q}_j)^2 + \lambda_P (\vec{p}_i{}^T \vec{p}_i) + \lambda_Q (\vec{q}_j{}^T \vec{q}_j)$$

Now, we can calculate the sub-gradients of the two latent vectors $\vec{p}_i$ and $\vec{q}_j$ to derive the following variable update rules:

$$\vec{p}_i \leftarrow \vec{p}_i + \beta(\ell_{(i,j)} q_j - \lambda_P \vec{p}_i) \tag{6.1}$$

$$\vec{q}_j \leftarrow \vec{q}_j + \beta(\ell_{(i,j)} p_i - \lambda_Q \vec{q}_j) \tag{6.2}$$

Here, β is the learning rate, whereas $\ell_{(i,j)}$ is the loss function that estimates how well the model approximates the ground truth:

$$\ell(i,j) = F_{(i,j)} - \vec{p}_i{}^T \vec{q}_j$$

The low-rank approximation here is accomplished by reconstructing the F matrix with two low-rank matrices $\mathbf{P}$ and $\mathbf{Q}$, and we use the row and column regularizers to prevent the model from overfitting to the training data.

In addition to standard loss functions such as logarithm loss, hinge loss, and squared hinge loss, we have also experimented with a pairwise loss function that resembles Bayesian Personalized Ranking [82].

The idea is that we assume that all positive, activated formulas should be ranked above all missing formulas (i.e., unused formulas) in each example row *ROW*. And the objective function could be written as:

$$\begin{aligned}\min_{\mathbf{P},\mathbf{Q}} \sum_{(i,j)\in ROW} \sum_{(i,w)\notin ROW_i} \log(1+\exp(\vec{p}_i{}^T(\vec{q}_w-\vec{q}_j))) + \mu_p||\vec{p}_i||_1 \\ +\mu_q(||\vec{q}_j||_1+||\vec{q}_w||_1) + \frac{\lambda_p}{2}||\vec{p}_i||_2^2 + \frac{\lambda_q}{2}(||\vec{q}_j||_2^2+||\vec{q}_w||_2^2)\end{aligned} \tag{6.3}$$

where ROW_i is the set of positive formulas in the i-th row of *ROW*. If we switch the notation of i as column index, j and w as row indexes, then this method will be optimizing column-oriented BPR instead of row-oriented BPR. In this work, we follow a recently proposed optimization approach called fast parallel stochastic gradient descent (FPSG) [21] to learn the latent first-order logic embeddings.

6.3.3 Learning First-Order Logic Embeddings

We outline our matrix factorization based method in Algorithm 1. Since this is a supervised learning approach, we assume the dataset includes a collection of triples as examples E.

During training, we first apply the gradient-based structure learning method [106] to all training triples E^{tr} to derive a set of plausible first-order formulas S. Given this subset of formulas, we then traverse all training examples to produce grounded ProPPR proof graphs G. For each training example i, we construct a row F_i in a matrix F. The columns correspond to the formulas learned by structural gradient, and F_i is non-zero if this particular formula is used during grounding. Figure 6.1 illustrates the transformation process. Then, we perform stochastic gradient descent training to learn the hidden low-rank embeddings of examples and formulas P and Q using the update rules outlined earlier. After learning the embeddings for the formulas, we average all the dimensions of the learned embedding of each formula to derive a parameter $\mathbf{w}_j$.

Datasets	#Relations	#Entities	#Train	#Valid	#Test
WordNet	18	40,943	141,442	5,000	5,000
FB15K	1,345	14,951	483,142	50,000	59,071

Table 6.1: Statistics of the two publicly available datasets used in the knowledge base completion experiments.

During testing, we still ground the testing examples using a set of candidate formulas S, but now the transition of the proof process will be guided by the $\mathbf{w}_j$s.

This new matrix factorization based parameter learning method is different from ProPPR's standard gradient based parameter learning method in Chapter 2. The major difference is that standard gradient based method does not model latent interactions among various formulas: we assume that each path is independent of each other, which might not always be true in practice. In contrast, the matrix factorization based method projects examples and formulas into a low-rank latent space, and consider the interactions among various examples and formulas during learning.

6.4 Experiments

In this section, we first introduce datasets used in this knowledge base completion study and the evaluation protocol. Then, we discuss the baselines. Finally, we show empirical results on these two datasets, including analyses on robustness and the effects of different loss functions for learning first-order logic embeddings.

6.4.1 Datasets and Evaluation Setup

We choose two popular datasets for the task of knowledge base completion. The statistics of the datasets are shown in Table 6.1. The task is to predict the missing head or tail entities in a relation fact triple: given the relation and an entity, we rank the candidate entities. Following

prior work [14], we use the Hits@10 measure, which indicates the portion of correct answers falling in the top-10 ranks. For example, if all testing queries have correct answers returned in the top-10 positions, then the system will have a perfect Hits@10 score of 100. We do not filter any triples in the experiments. Unless specified, the latent dimension of first-order logic embeddings is set to 8. ProPPR's reset parameter α is set to 0.1, and the approximation error parameter ε is set to 1×10^{-3}.

6.4.2 Baselines

To demonstrate the effectiveness of our method, we compare with a number of competitive and state-of-the-art baselines—**Unstructured** [15]: a TransE [14] baseline that considers the data as mono-relational and sets all translations to 0; **RESCAL** [72]: a collective matrix factorization model that factorizes a rank-3 tensor; **SE** [13], **SME** [15], and **LFM** [44]: energy-based structured embedding models of knowledge bases; **TransE** [14]: a popular multi-relational model that considers relationships as translations in the embedding space; **ConvNets** [88]: a recent study on convolutional neural network based concept learning model. **TransH** [109]: an improved TransE-style model that considers reflexive, 1-to-*N*, *N*-to-1, and *N*-to-*N* relations; **TransR** [59]: a state-of-the-art model that builds entity and relation embeddings in separate entity space and relation spaces; **PTransE** [58]: a state-of-the-art TransE-style method that uses path-based constraints to learn KB embeddings[1].

6.4.3 Comparing with Various Baselines

We show the comparison of our approaches with various methods on the FB15K dataset in Table 6.2. We see that TransE and its variants outperform energy based models such as SE, SME, and LFM. The ConvNets model also achieves reasonable results. The best results from prior work on this dataset is from PTransE, a TransE-like model with relational path as constraints

[1]Note that PTransE and ConvNets do include results for the WordNet dataset.

Methods	Hits@10
Unstructured [15]	4.5
RESCAL [72]	28.4
SE [13]	28.8
SME [15]	31.3
LFM [44]	26.0
TransE [14]	34.9
ConvNets [88]	37.7
TransH [109]	45.7
TransR [59]	48.4
PTransE [58]	51.8
ProPPR	59.0
ProPPR+MF (k=8)	**61.0**

Table 6.2: Comparing our approach with various baselines on the FB15K dataset.

during training. We see that the structural gradient based approach from ProPPR obtains a strong performance, leveraging the effectiveness of symbolic reasoning and probabilistic modeling. Finally, leading the scoreboard is our first-order logic embedding enhanced ProPPR model with a Hits@10 score at 61.0.

On the WordNet dataset, we show the comparison of performances in Table 6.3. We observe similar trends of results from prior work: TransE and its variants have dominated the score sheet on this WordNet dataset. ProPPR's structure learning produces a strong result of 83.0, showing ProPPR's advantage on modeling hypernym and hyponym relations. When considering first-order formula embeddings, we observe a large improvement of Hits@10 score to 94.1. Comparing to the prior experimental results from the FB15K dataset, we believe that the large-margin improvement observed from the WordNet experiment is due to the nature of these two datasets. FB15K has more than 1K relations, but it has only about 15K entities. We observe a

Methods	Hits@10
Unstructured [15]	35.5
RESCAL [72]	37.2
SE [13]	68.5
SME [15]	65.1
LFM [44]	71.4
TransE [14]	75.4
TransH [109]	75.4
TransR [59]	79.8
ProPPR	83.0
ProPPR+MF (k=8)	**94.1**

Table 6.3: Comparing our approach with various baselines on the WordNet dataset.

large number of first-order formula candidates after structure learning, while the training examples for learning first-order logic embeddings are relatively sparse. In contrast, the WordNet data set has only over a dozen relations, but the number of total entities is about three times larger, allowing more training examples.

6.4.4 Varying the Latent Dimensions

In this experiment, we vary the latent dimension k for learning first-order logic embeddings. The results are shown in Figure 6.2. We see that the latent dimension has an effect on the performance of the WordNet dataset, and that $k = 4$ yields the best overall performance.

6.4.5 Varying the Loss Functions

In this subsection, we investigate the effects of choosing various loss functions for learning first-order logic embeddings. We see that the general loss functions, such as the log loss, hinge loss,

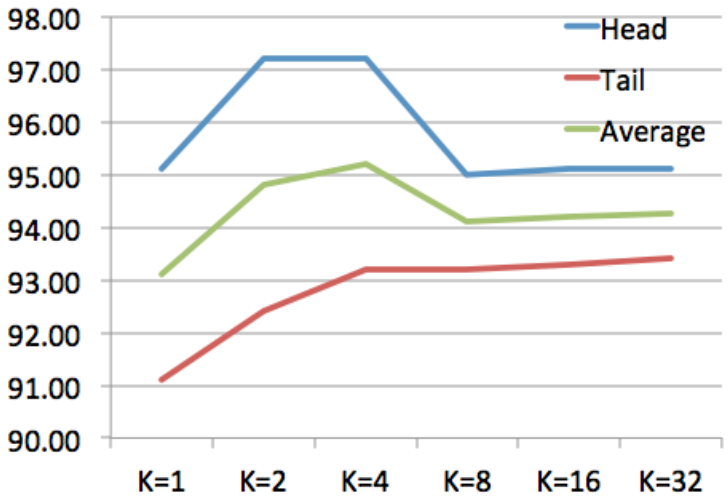


Figure 6.2: The Hits@10 scores of varying #dimensions for retrieving head and tail entities on the WordNet dataset.

and the squared hinge loss do not make much difference on the results. Interesting, when using row-oriented pairwise log loss, we obtain the best performance. This pairwise loss function resembles Bayesian Personalized Ranking [82], which assumes that all positive first-order logic formulas should be ranked above all missing formulas in an example. This is a useful assumption for the pairwise loss function, which allows us to model the non-existent "negative" formulas. Interestingly, column-oriented pairwise log loss obtains the lowest performances, because it is inappropriate to compare formulas across totally different examples.

Figure 6.3: The Hits@10 scores of different loss functions for learning embeddings to retrieve head and tail entities on the WordNet dataset.

Algorithm 1 A Matrix Factorization Based Algorithm for Learning First-Order Logic Embeddings

Input: training examples E in the form of relation triples.
procedure TRAINING(E^{tr})
    $S \leftarrow$ StructureLearning(E^{tr})
    for each training example E_i^{tr} in E^{tr} **do**
        $G_i \leftarrow$ Grounding(E_i^{tr},S)
        $F_i \leftarrow$ GraphToMatrix(G_i)
        for each epoch t **do**
            for each cell i, j in F_i **do**
                $\vec{p}_i{}^{(t)} \leftarrow \vec{p}_i{}^{(t)} + \delta(\ell_{(i,j)} q_j^{(t)} - \lambda_P \vec{p}_i{}^{(t)})$
                $\vec{q}_j{}^{(t)} \leftarrow \vec{q}_j{}^{(t)} + \delta(\ell_{(i,j)} p_i^{(t)} - \lambda_Q \vec{q}_j{}^{(t)})$
            end for
        end for
    end for
    for each formula embedding Q_j in **Q do**
        $\mathbf{w}_j \leftarrow mean(Q_j)$
    end for
end procedure
procedure TESTING(E^{te})
    for each test example E_i^{te} in E^{te} **do**
        $\vec{A} \leftarrow$ QueryAnswering(E_i^{te},S,W)
    end for
end procedure

Chapter 7

Joint Learning and Inference for Information Extraction and Reasoning

7.1 Introduction

In this chapter, we introduce our work on joint extraction and reasoning. In particular, we will discuss the work of joint information extraction and reasoning of the relational KB data, as well as our work on exploiting various latent representations for joint learning and inference of embeddings and first-order logic. An earlier version of this work appears elsewhere [102].

Joint Information Extraction and Reasoning

Information extraction (IE) is often an early stage in a pipeline that contains non-trivial downstream tasks, such as parsing [29], question answering [67], machine translation [5], or other applications [57, 104]. Knowledge bases (KBs) populated by IE techniques have also been used as an input to systems that learn rules allowing further inferences to be drawn from the KB [54], a task sometimes called KB completion [94, 106, 111]. Pipelines of this sort frequently suffer

from error cascades, which reduces performance of the full system[1].

In this chapter, we address this issue, and propose a joint model system for IE and KB completion in a statistical relational learning (SRL) setting [37, 97]. In particular, we outline a system which takes as input a partially-populated KB and a set of relation mentions in context, and jointly learns (1) how to extract new KB facts from the relation mentions, and (2) a set of logical rules that allow one to infer new KB facts. Evaluation of the KB facts inferred by the joint system shows that the joint model outperforms its individual components. We present a joint model for IE and relational learning in a statistical relational learning setting which outperforms universal schemas [85], a state-of-the-art joint method. We will also discuss the possibility of integrating latent matrix factorization techniques and ProPPR's logical inference in the future.

Joint Reasoning Using Low-Dimensional Embeddings

In the abstract of this thesis, we have mentioned that reasoning with symbolic systems, such as first-order logics, is a primary research theme in the AI community in the 1980s. Interestingly, neural network based learning methods revived in the 1980s as well, even though researchers back then did not figure out how to scale up training on large datasets.

First-order logic based methods are often deterministic, reliable, and perform well using a handful crisp inference rules. However, an obvious drawback of these traditional symbolic systems is the scalability issue: it is expensive to create and update the hand-written rules, and they often suffer from the coverage issues when new data comes in.

On the other hand, neural network based methods are reviving again in the past decade, due to the availability of massive computing resources, as well as better learning and inference methods. Even though there have been many promising results reported for simple classification tasks, it is still unclear how well we can use neural models to perform reasoning tasks in statistical relational learning [37]. Another notable issue with the pure neural network based methods is

[1]For example, KBP slot filling is known for its complex pipeline, and the best overall F1 scores [4, 113] for recent competitions are within the range of 30-40.

the explainability: the learned hidden layers are often hard to explain, and it is often difficult to perform error analysis, and understand the science behind neural methods.

In this thesis, we propose a model that allows joint learning and inference of information extraction and reasoning clauses. Motivated by this, we propose several approaches that integrate latent representations in ProPPR. First, we introduce a novel extension of the joint IE and reasoning model called Latent Context Invention (LCI), which associates latent states with context features for the IE component of the model. We show that LCI further improves performance, leading to a substantial improvement over prior state-of-the-art methods for joint relation-learning and IE.

7.2 Joint IE and Reasoning

7.2.1 Dataset Generation

The KBs and text used in experiments of this chapter were derived from Wikipedia. Briefly, we choose a set of closely-related pages from a hand-selected Wikipedia list. These pages define a set of entities M, and a set of commonly-used Infobox relations $\mathscr{R}$ between these entities define a KB. The relation mentions are hyperlinks between the pages, and the features of these relation mentions are words that appear nearby these links. This information is encoded in a single relation *link(X,Y,W)*, which indicates that there is hyperlink between Wikipedia pages X to Y which is near the context word W. The Infobox relation triples are stored in another relation, *rel(R,X,Y)*. [2]

Figure 7.1 shows an example. We first find the "*European royal families*" to find a list of

[2]In more detail, the extraction process was as follows. (1) We used a DBpedia dump of categories and hyperlink structure to find pages in a category; sometimes, this included crawling a supercategory page to find categories and then entities. (2) We used the DBpedia hyperlink graph to find the target entity pages, downloaded the most recent (2014) version of each of these pages, and collected relevant hyperlinks and anchor text, together with 80 characters of context to either side.

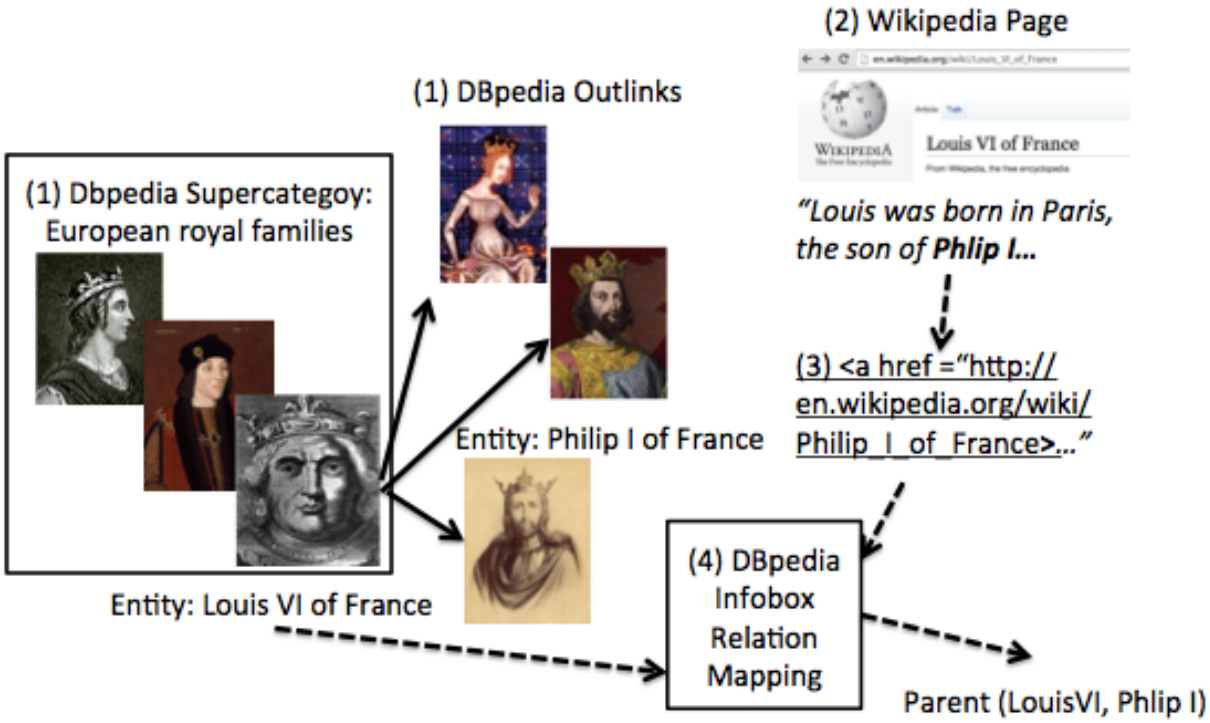


Figure 7.1: The data generation example as described in section 7.2.1.

entities $\mathscr{E}$. This list contains the page "*Louis VI of France*", the *source entity*, which contains an outlink to the *target entity page* "*Philip I of France*". On the source page, we can find the following text: "*Louis was born in Paris, the son of* ***Philip I*** *and his first wife, Bertha of Holland.*" From Infobox data, also may know of a relationship between the source and target entities: in this case, the target entity is the *parent* of the source entity.

7.2.2 Theory for Joint IE and SL

The structure learning templates we used are identical to those used in prior chapter (Chapter4), and are summarized by the clauses (a-c) in Table 7.1. In the templates in the left-hand side of the table, P, R, $R1$ and $R2$ are variables in the template, which will be bound to specific relations found to be useful in prediction. (The interpreter rules on the right-hand side are the

implementation of these templates in ProPPR)

The second block of the table contains the templates used for IE. For example, template (d) recall that the predicate *link* indicates a hyperlink from Wikipedia page X to Y, which includes the context word W between two entities X and Y. The abductive predicate *abduce_indicates* activates a feature template, in which we learn the degree of association of a context word and a relation from the training data. These rules essentially act as a trainable classifier which classifies entity pairs based on the hyperlinks that contain them, and classifies the hyperlinks according to the relation they reflect, based on context-word features.

Notice that the learner will attempt to tune word associations to match the gold *rel* facts used as training data, and that doing this does *not* require assigning labels to individual links, as would be done in a traditional distant supervision setting: instead these labels are essentially left latent in this model. Similar to "deep learning" approaches, the latent assignments are provided not by EM, but by hill-climbing search in parameter space.

A natural extension to this model is to add a bilexical version of this classifier in clause (e), where we learn a feature which conjoins word $W1$, word $W2$, and relation R.

Combining the clauses from (a) to (e), we derive a hybrid theory for joint SL and IE: the structure learning section involves a second-order probabilistic logic theory, where it searches the relational KB to form plausible first-order relational inference clauses. The information extraction section from (d) to (e) exploits the distributional similarity of contextual words for each relation, and extracts relation triples from the text, using distant supervision and latent labels for relation mentions (which in our case are hyperlinks). Training this theory as a whole trains it to perform joint reasoning to facts for multiple relations, based on relations that are known (from the partial KB) or inferred from the IE part of the theory. Both parameters for the IE portion of the theory and inference rules between KB relations are learned.[3]

[3]In in addition to finding rules which instantiate the templates, weights on these rules are also learned.

7.2.3 Datasets

Using the data generation process that we described in section 7.2.1, we extract two datasets from the supercategories of "*European royal families*" and "*American people of English descent*, and third geographic dataset using three lists: "*List of countries by population*", "*List of largest cities and second largest cities by country*" and "*List of national capitals by population*".

For the *royal* dataset, we have 2,258 pages with 67,483 source-context-target mentions, and we use 40,000 for training, and 27,483 for testing. There are 15 relations. In the *American* dataset, we have 679 pages with 11,726 mentions, and we use 7,000 for training, and 4,726 for testing. This dataset includes 30 relations. As for the *Geo* dataset, there are 497 pages with 43,475 mentions, and we use 30,000 for training, and 13,375 for testing. There are 10 relations. The detailed relations are described in Table 7.2.

7.2.4 Experiments

To evaluate these methods, we use the setting of Knowledge Base completion [94, 106, 111]. We randomly remove a fixed percentage of facts in a training knowledge base, train the learner from the partial KB, and use the learned model to predict facts in the test KB. KB completion is well-studied task in SRL, where multiple relations are often needed to fill in missing facts, and thus reconstruct the incomplete KB. We use mean average precision (MAP) as the evaluation metric.

Baselines

To understand the performance of our joint model, we compare with three prior methods. **Structure Learning (SL)** includes the second-order relation learning templates (a-c) from Table 7.1. **Information Extraction (IE)** includes only templates (d) and (e). **Markov Logic Networks (MLN)** is the Alchemy[4]'s implementation of Markov Logic Networks [83], using the first-order

[4]http://alchemy.cs.washington.edu/

clauses learned from SL method[5]. We used conjugate gradient weight learning [61] with 10 iterations. Finally, **Universal Schema** is a state-of-the-art matrix factorization based universal method for jointly learning surface patterns and relations. We used the code and parameter settings for the best-performing model (NFE) from [85].

As a final baseline method, we considered a simpler approach to clustering context words, which we called **Text Clustering**, which used the following template:

R(X,Y) :-

clusterID(C),link(X,Y,W),

cluster(C,W),related(R,W).

Here surface patterns are grouped to form latent clusters in a relation-*independent* fashion.

The Effectiveness of the Joint Model

Our experimental results are shown in 7.3. The leftmost part of the table concerns the Royal dataset. We see that the universal schema approach outperforms the MLN baseline in most cases, but ProPPR's SL method substantially improves over MLN's conjugated gradient learning method, and the universal schema approach. This is perhaps surprising, as the universal schema approach is also a joint method: we note that in our datasets, unlike the New York Times corpus used in [85], large numbers of unlabeled examples are not available. The unigram and bilexical IE models in ProPPR also perform well—better than SL on this data. The joint model outperforms the baselines, as well as the separate models. The difference is most pronounced when the background KB gets noisier: the improvement of 10% missing setting is about 1.5 to 2.3% MAP, where as for the 50% missing setting, the absolute MAP improvement is from 8% to 10%.

In the next few columns of Table 7.3, we show the KB completion results for the *Geo* dataset. This dataset has fewer relations, and the most common one is *country*. The overall MAP scores are much higher than the previous dataset. MLN's results are good, but still generally below the universal schema method. On this dataset, the universal schema method performs better than

[5]We also experimented with Alchemy's structure learning, but it were not able to generate results in 24 hours.

the IE only model for ProPPR in most settings. However, the ProPPRjoint model still shows large improvements over individual models and the baselines: the absolute MAP improvement is 22.4%.

Finally, in the rightmost columns of Table 7.3, we see that the overall MAP scores for the *American* dataset are relatively lower than other datasets, perhaps because it is the smallest of the three. The universal schema approach consistently outperforms the MLN model, but not ProPPR. On this dataset the SL-only model in ProPPR outperforms the IE-only models; however, the joint models still outperform individual ProPPR models from 1.5% to 6.4% in MAP.

The averaged training runtimes on an ordinary PC for unigram joint model on the above *Royal, Geo, American* datasets are 38, 36, and 29 seconds respectively, while the average testing times are 11, 10, and 9 seconds. For bilexical joint models, the averaged training times are 25, 10, and 10 minutes respectively, whereas the testing times are 111, 28, and 26 seconds respectively.

7.2.5 Related Work

In NLP, this thesis clearly aligns with recent work on joint models of individual text processing tasks. For example, Finkel and Manning [29] work on the problem of joint IE and parsing, where they use tree representations to combine named entities and syntactic chunks. Recently, Devlin et al. [28] use a joint neural network model for machine translation, and obtain an impressive 6.3 BLEU point improvement over a hierarchical phrase-based system.

In information extraction, weak supervision [24, 66] is a common technique for extracting knowledge from text, without large-scale annotations. In extracting Infobox information from Wikipedia text, Wu and Weld [116, 117] also use a similar idea. In an open IE project, Banko et al. [7] use a seed KB, and utilize weak supervision techniques to extend it. Note that weakly supervised extraction approaches can be noisy, as pair of entities in context may be associated with one, none, or several of the possible relation labels, a property which complicates the application of distant supervision methods [42, 66, 84, 96].

Lao et al. [55] learned syntactic rules for finding relations defined by "lexico-semantic" paths spanning KB relations and text data. Wang et al. [108] extends the methods used by Lao et al. to learn mutually recursive relations. Recently, Riedel et al. [85] propose a matrix factorization technique for relation embedding, but their method requires a large amount of negative and unlabeled examples. Weston et al. [112] connect text with KB embedding by adding a scoring term, though no shared parameters/embeddings are used. All these prior works make use of text and KBs. Unlike these prior works, our method is posed in an SRL setting, using a scalable probabilistic first-order logic, and allows learning of relational rules that are mutually recursive, thus allowing learning of multi-step inferences. Unlike some prior methods, our method also does not require negative examples, or large numbers of unlabeled examples.

7.3 Incorporating Latent Factors

Latent Context Invention Note that so far both the IE clauses (d-e) are fully observable: there are no latent predicates or variables. Recent work [85] suggests that learning latent representations for words improves performance in predicting relations. Perhaps this is because such latent representations can better model the semantic information in surface forms, which are often ambiguous.

We call our method latent context invention (LCI), and like the work in Chapter 5, it is inspired from literature in predicate invention [49]. The SL method is inspired by Cropper and Muggleton's Metagol system [25], which includes predicate invention. In principle predicates could be invented by SL, by extending the interpreter to consider "invented" predicate symbols as binding to its template variables (e.g., P and R); however, in practice invented predicates leads to close dependencies between learned rules is highly sensitive to the level of noise in the data and did not perform well in experiments. LCI applies the idea of predicate invention to the context space: instead of inventing new predicates, we now invent a *latent context property* that captures the regularities among the similar relational lexical items. To do this, we introduce

some additional rules of the form *latent(1) :- true*, *latent(2) :- true*, etc, and allow the learner to find appropriate weights for pairing these arbitrarily-chosen values with specific words. This is implemented by template (f) in Table 7.1. Adding this to the joint theory means that we will learn to map surface-level lexical items (words) to the "invented" latent context values and also to relation.

Another view of LCI is that we are learning a latent embedding of words jointly with relations. In template (f) we model a single latent dimension, but to model higher-dimensional latent variables, we can add the clauses such as (g), which constructs a two-dimensional latent space. Below we will call this variant method hLCI.

The Effectiveness of LCI

Finally we consider the latent context invention (LCI) approach. The last three rows of Table 7.3 show the performances of LCI and hHCI. We compare it here with the best previous approach, the joint IE + SL model, and text clustering approach.

For the *Royal* dataset, first, the LCI and hLCI models clearly improve over joint IE and SL. In noisy conditions of missing 50% facts, the biggest improvement of LCI/hLCI is 2.4% absolute MAP.

From the *Geo* dataset, we see that the joint models and joint+latent models have similar performances in relatively clean conditions (10%-30%) facts missing. However, in noisy conditions, we the LCI and hLCI model has an advantage of between 1.5% to 1.8% in absolute MAP.

Finally, the results for the *American* dataset show a consistent trend: again, in noisy conditions (missing 40% to 50% facts), the latent context models outperform the joint IE + SL models by 2.9% and 3.7% absolute MAP scores.

Although the LCI approach is inspired by predicate invention in inductive logic programming, our result is also consistent with theories of generalized latent variable modeling in probabilistic graphical models and statistics [93]: modeling hidden variables helps take into account

the measurement (observation) errors [31] and results in a more robust model.

7.3.1 Related Work

Our work clearly relates to recent studies on extending the PRA method: For example, Gardner et al. [35] combines the vector-space based semantic representation with PRA's logical inference scheme, and obtain interesting results in a multiple question answering task. Our work also aligns to a recent study [34] that models latent representation to improve the PRA approach.

In the NLP community, there has been strong interests of combining logical and distributional systems. For example, Garrette et al. [36] integrate first-order logical semantics with probabilistic models using Markov Logic Networks. Similarly, Betagy et al. [12] combine logical and distributional representation using Markov Logic Networks, and show interesting results on the task of textual entailment and semantic textual similarity.

Our work is also closely related to recent studies on matrix and tensor factorization methods [73, 85, 94] for relational learning. In particular, Rocktaschel et al. [86] propose a logical embedding method to combine logical inference and matrix factorization, whereas Grefenstette [38] describes a tensor representation of a formal distributional semantics model.

	Rule template	ProPPR clause
Structure learning		
(a)	P(X,Y) :- R(X,Y)	interp(P,X,Y) :- interp0(R,X,Y),abduce_if(P,R).
		abduce_if(P,R) :- true # f_if(P,R).
(b)	P(X,Y) :- R(Y,X)	interp(P,X,Y) :- interp0(R,Y,X),abduce_ifInv(P,R).
		abduce_ifInv(P,R) :- true # f_ifInv(P,R).
(c)	P(X,Y) :- R1(X,Z),R2(Z,Y)	interp(P,X,Y) :- interp0(R1,X,Z),interp0(R2,Z,Y), abduce_chain(P,R1,R2).
		abduce_chain(P,R1,R2) :- true # f_chain(P,R1,R2).
	base case for SL interpreter	interp0(P,X,Y) :- rel(R,X,Y).
	insertion point for learned rules	interp0(P,X,Y) :- *any rules learned by SL.*
Information extraction		
(d)	R(X,Y) :- link(X,Y,W), indicates(W,R).	interp(R,X,Y) :- link(X,Y,W),abduce_indicates(W,R).
		abduce_indicates(W,R) :- true #f_ind1(W,R).
(e)	R(X,Y) :- link(X,Y,W1), link(X,Y,W2), indicates(W1,W2,R).	interp(R,X,Y) :- link(X,Y,W1),link(X,Y,W2), abduce_indicates(W1,W2,R).
		abduce_indicates(W1,W2,R) :- true #f_ind2(W1,W2,R).

Table 7.1: The ProPPR template and clauses for joint structure learning and information extraction.

Dataset	Relations
Royal	birthPlace, child, commander, deathPlace, keyPerson, knownFor, monarch, parent, partner, predecessor, relation, restingPlace, spouse, successor, territory
American	architect, associatedBand, associatedMusicalArtist, author, birthPlace, child, cinematography, deathPlace, director, format, foundationOrganisation, foundationPerson, influenced, instrument, keyPerson, knownFor, location, musicComposer, narrator, parent, president, producer, relation, relative, religion, restingPlace, spouse, starring, successor, writer
Geo	archipelago, capital, country, daylightSavingTimeZone, largestSettlement, leaderTitle, mottoFor, timeZone, twinCity, twinCountry

Table 7.2: The relations that we consider in each dataset.

	Royal					Geo					American				
% missing	10%	20%	30%	40%	50%	10%	20%	30%	40%	50%	10%	20%	30%	40%	50%
Baselines															
MLN	60.8	43.7	44.9	38.8	38.8	80.4	79.2	68.1	66.0	68.0	54.0	56.0	51.2	41.0	13.8
Universal Schema	48.2	53.0	52.9	47.3	41.2	82.0	84.0	75.7	77.0	65.2	56.7	51.4	55.9	54.7	51.3
SL	79.5	77.2	74.8	65.5	61.9	83.8	80.4	77.1	72.8	67.2	73.1	70.0	71.3	67.1	61.7
IE only															
IE (U)	81.3	78.5	76.4	75.7	70.6	83.9	79.4	73.1	71.6	65.2	63.4	61.0	60.2	61.4	54.4
IE (U+B)	81.1	78.1	76.2	75.5	70.3	84.0	79.5	73.3	71.6	65.3	64.3	61.2	61.1	62.1	55.7
Joint															
SL+IE (U)	82.8	80.9	79.1	77.9	78.6	89.5	89.4	89.3	88.1	87.6	74.0	73.3	73.7	70.5	68.0
SL+IE (U+B)	**83.4**	**82.0**	**80.7**	**79.7**	**80.3**	**89.6**	**89.6**	**89.5**	**88.4**	**87.7**	**74.6**	**73.5**	**74.2**	**70.9**	**68.4**

Table 7.3: The MAP results for KB completion on three datasets. U: unigram. B: bigram. Best result in each column is highlighted in **bold**.

	Rule template	ProPPR clause
Latent context invention		
(a)	R(X,Y) :- latent(L), link(X,Y,W), indicate(W,L,R)	interp(R,X,Y) :- latent(L),link(X,Y,W),abduce_latent(W,L,R). abduce_latent(W,L,R) :- true #f_latent1(W,L,R).
(b)	R(X,Y) :- latent(L1),latent(L2) link(X,Y,W), indicate(W,L1,L2,R)	interp(R,X,Y) :- latent(L1),latent(L2),link(X,Y,W), abduce_latent(W,L1,L2,R). abduce_latent(W,L1,L2,R) :- true #f_latent2(W,L1,L2,R).

Table 7.4: The ProPPR template and clauses for latent context invention in the task of joint structure learning and information extraction.

	Royal					Geo					American				
% missing	10%	20%	30%	40%	50%	10%	20%	30%	40%	50%	10%	20%	30%	40%	50%
Baselines															
MLN	60.8	43.7	44.9	38.8	38.8	80.4	79.2	68.1	66.0	68.0	54.0	56.0	51.2	41.0	13.8
Universal Schema	48.2	53.0	52.9	47.3	41.2	82.0	84.0	75.7	77.0	65.2	56.7	51.4	55.9	54.7	51.3
SL	79.5	77.2	74.8	65.5	61.9	83.8	80.4	77.1	72.8	67.2	73.1	70.0	71.3	67.1	61.7
IE only															
IE (U)	81.3	78.5	76.4	75.7	70.6	83.9	79.4	73.1	71.6	65.2	63.4	61.0	60.2	61.4	54.4
IE (U+B)	81.1	78.1	76.2	75.5	70.3	84.0	79.5	73.3	71.6	65.3	64.3	61.2	61.1	62.1	55.7
Joint															
SL+IE (U)	82.8	80.9	79.1	77.9	78.6	89.5	89.4	89.3	88.1	87.6	74.0	73.3	73.7	70.5	68.0
SL+IE (U+B)	83.4	82.0	80.7	79.7	80.3	89.6	89.6	89.5	88.4	87.7	**74.6**	73.5	74.2	70.9	68.4
Joint + Latent															
Joint + Clustering	**83.5**	82.3	81.2	80.2	80.7	89.8	89.6	89.5	88.8	88.4	**74.6**	73.9	74.4	71.5	69.7
Joint + LCI	**83.5**	**82.5**	81.5	80.6	81.1	**89.9**	**89.8**	**89.7**	89.1	89.0	**74.6**	74.1	74.5	72.3	70.3
Joint + LCI + hLCI	**83.5**	**82.5**	**81.7**	**81.0**	**81.3**	**89.9**	89.7	**89.7**	**89.6**	**89.5**	**74.6**	**74.4**	**74.6**	**73.6**	**72.1**

Table 7.5: Comparing the MAP results of LCI and various baselines for KB completion on three datasets. U: unigram. B: bigram. Best result in each column is highlighted in **bold**.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, we address the problem of designing a scalable probabilistic logic. This is one of the central problems in statistical relational learning as well: we know that in prior work, most systems' runtime depend on the size of the ground formulas. This also prevents the application of statistical relational learning approaches to large real-world datasets—imagining that we are dealing with a knowledge graph completion problem with hundreds of thousands of relations, triples, and rules, if we do not have an efficient probabilistic programming language, the reasoning on such a large graph is inevitably slow.

The core research problem that we are addressing in this thesis is about how to design scalable learning and inference algorithms to operate over rich knowledge representations. To start with, we adopt the semantics of Prolog, which was widely used in the AI community in the early days. To make inference efficient, we rely on an approximate version of the personalized PageRank algorithm. A key property of this algorithm is that we are essentially performing random-walk-with-restarts procedures to find multiple short, direct answers that are closer to the query node. One benefit is clearly from the correctness perspective: we prefer answers that are supported by direct evidence, rather than potentially incorrect answers that are in the indirect, distance paths.

Another benefit of the design is obviously from the efficiency standpoint: essentially we are building subgraphs for local grounding which can be done in a fixed amount of time, that is not influenced by the size of the input knowledge graph. Unlike the Prolog programming language that cannot learn from data, we design a scalable parallel stochastic gradient descent approach for learning weights for each logical formula. This is essential for many tasks in this line of research: we would like to learn a set of weights for more plausible clauses that can lead us towards the positive solutions using labels provided by human, such that in the testing time, we plug in the weights for each formulas, and the hope is that we are able to locate the correct answers using prior evidence.

Another key contribution of this thesis is the introduction of an efficient structure learning algorithm for ProPPR. This is an essential component to deal with big data: with large datasets, it is simply impossible for human to write all these formulas by hand, so we must address the problem of automatically inducing and learning logical programs. In prior work, it is noted that for one algorithm, it took 28 days for structure learning to finish on a Cora dataset that has only 47K triples. This is not practical for any real-world applications. In my work, we take a completely different approach: instead of solving the unknown structure learning problem directly, we reduce it to something we already know how to solve—which is parameter learning in ProPPR. The main idea is that we would like to create a higher level of abstraction in the form of template based second-order abductive logic. Now, we can map the candidates for the first-order logic formulas that we would like to learn, into the parameter space of this second order logic. In this way, we can use parallel stochastic gradient descent to solve the expensive search problem very efficiently. Moreover, we also provide an iterative algorithm and incrementally adds plausible inference formulas to the templates. Empirical results suggest that this method outperforms various strong baselines, and we significantly reduce the runtime relative to earlier systems.

In addition to these two fundamental contributions, my thesis also touches on some very

challenging, long-lasting problems in AI—including predicate invention, where machines are trying to learn new concepts from existing knowledge graph. Predicate invention is extremely complicated, because it is not clear to many of us that whether using this invented new predicate can be helpful to the end task. In this thesis, instead of working on traditional, hard predicate invention, my work focuses on bridging the theory in structured regularization in machine learning with predicate invention. Instead of creating new predicate, we implicitly learn the correlations among similar formulas using structured regularization techniques. Empirical evidence shows that it is quite useful to take into account structures in the form of regularization for structure learning.

In addition to the standard parallel stochastic gradient descent based parameter learning, we have also explored an alternative parameter learning method based on matrix factorization. In prior work on (probabilistic) first-order logics, most systems uses binary states (true or false) or a single scalar-valued parameter to represent first-order logic formulas. We would like to exploit the possibility of learning a low-dimensional latent vector embedding to represent a first-order logic formula. By using a low-rank approximation method, we have learned latent embeddings for examples and logic formulas, which improve the knowledge graph completion performances on two standard benchmark datasets.

We also worked on many practical applications of ProPPR. One major contribution is about scaling up ProPPR to the NELL dataset that includes 1 million entities, where we also show that there is a benefit of modeling the joint, recursive inference of multiple relations, comparing to systems such as the path-ranking algorithm. In the natural language processing domain, we also propose the use of ProPPR to close the gap between text and knowledge base research. The motivation is that for most of the information extraction systems that extract relations, often lexical features are the only source of information, and important, direct evidence from knowledge graphs are completely ignored. In the knowledge base reasoning community, this is actually opposite—most research are done using only the triples rather than with the context of where

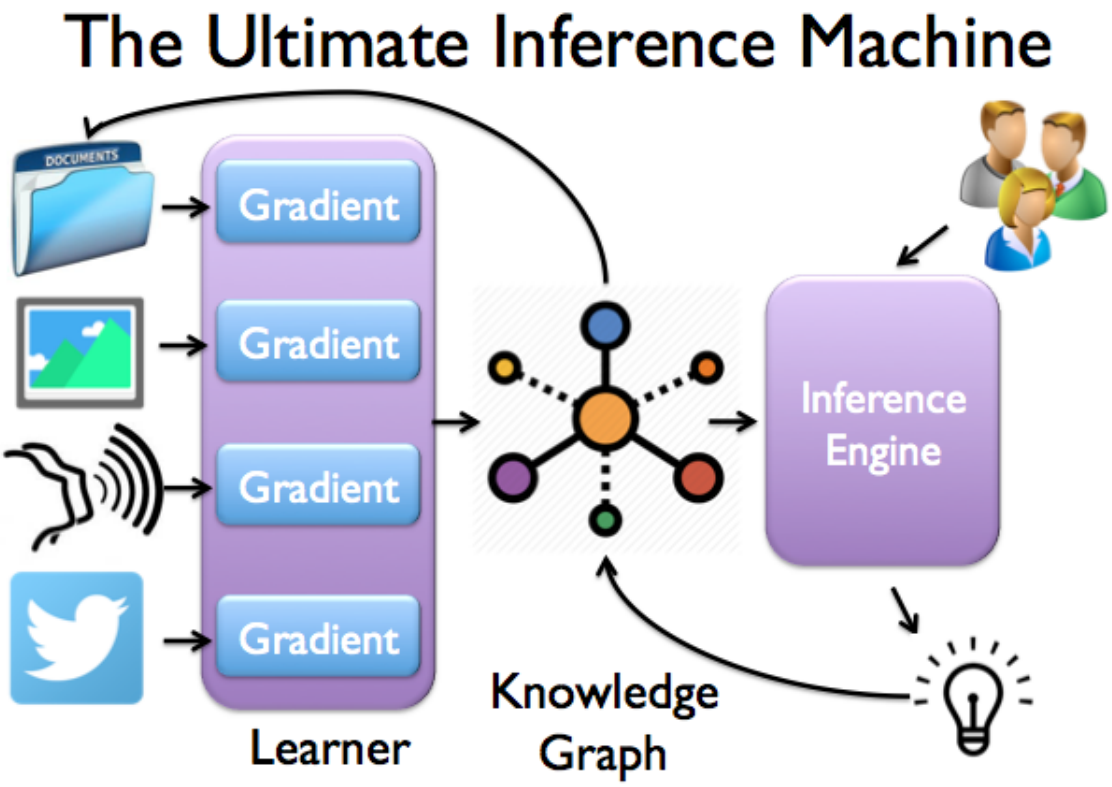


Figure 8.1: Future work: building an ultimate inference machine.

these triples happen, missing out a lot of useful insights about the data. In our work, we use ProPPR for joint information extraction, and reasoning, improving these individual components respectively.

8.2 Future Work

We believe that this is an exciting time to work on combining machine learning and reasoning. In the future, would like to build an ultimate inference machine, shown in Figure 8.1. I believe that only working on structured knowledge graph is not enough: we need to take other sources of information, such as texts, images, audio/video, and social media data. I propose that we can use gradient based learning methods to unify the multimodal learning problem. The learned

parameters could be used to inform the inference engine, and the machine should be able to answer any questions raised by the users, no matter in any form, or whether it has an existing answer in the system. We can also add inferred answer to the knowledge graph, and use the new knowledge graph to collect more training data.

I have three projected subgoals. First, I am interested in robust open-world learning and reasoning. By "robust", I mean we can consider integrating the recent advances of deep learning with our reasoning methods to achieve robust performance on large datasets. By "open-world learning", I am interested in building systems that can handle unlimited entities and relations via predicate invention, taking the temporal information into account, and also make the inference more efficient. In addition to this, I believe that automatic optimization of hyperparameters would also be an important topic: in many modern statistical relational learning systems, there are dozens of hyperparameters, and it is very difficult for amateurs to optimize the system. Traditional hyperparameter optimization methods, such as grid search and cross-validation based turning, do not scale up for large number of hyperparameters. In the future, I would like to investigate Bayesian hyperparameter optimizating techniques to reduce the search space of automatic hyperparameter tuning.

Secondly, I am also interested in breaking the information barrier, for example, how to combine knowledge from multimodalities, and how to jointly learn and reason across multimodal data. Applying existing statistical relational learning techniques to multimodal data is challenging, because the different characteristics and properties of different sources. For example, text data is often sparse and discrete, whereas image data is often dense and continuous. To the best of my knowledge, current statistical relational learning systems only deal with triples in knowledge bases, and they are not directly applicable to text and images. In my opinion, we can consider using embedding-based methods to unify learning and reasoning for multimodal statistical relational learning.

Finally, I believe that now it is also a great opportunities to introduce these machine learning

and reasoning techniques for better decision making in all scientific disciplines. For example, even though there exists vast amount of historical documents in digital form, machine learning and data-driven based reasoning approaches are still not the major research methodology in history research. Similarly, even electronic medical records and biomedical knowledge bases are available, predictive technologies are still not widely used to support the decision making for doctors. Scalable machine learning approaches applied to big relational data have huge potentials in these areas, and I believe these future research directions can fundamental change how people access knowledge and make decisions.

Appendix A

Notation

Symbol	Meaning
$O(\cdot)$	the Big-O asymptotic notation that describes the limiting behavior of a function when the argument tends towards infinity
$o(\cdot)$	the Little-O asymptotic notation that describes the limiting behavior of a function when the argument tends towards infinity
$\|DB\|$	the number of facts in a database
n^k	a total of n KB constants with arity k
α	a reset parameter in personalized PageRank
ε	a stopping criterion and the desired degree of approximation
LP	a logic program
c	a clause in a logic program that takes the form of $R' \leftarrow S'_1, \ldots, S'_\ell$
Q	a conjunctive query
v_0	a root vertex
$G'_{Q,LP}$	a graph to be constructed by using Q and LP
R	a subgoal
u	a vertex of the form $(Q, (R_1, \ldots, R_k))$
θ	a most general unifier $mgu(R_1, R')$ for subgoals R_1 and R'

$Q\theta$	transformed query, which denotes the result of applying the substitution θ to Q
$\square$	an empty subgoal list
$Pr(v\|u)$	the transition probability for reaching v from u
$\hat{G}$	the approximated personalized PageRank subgraph
α'	a lower-bound on $Pr(v_0\|u)$ for any node u to be added to the graph $\hat{G}$
$\mathbf{p}$	an approximate personalized PageRank vector
$\mathbf{r}$	a residual vector
$N(u)$	the degree of vertex u
$\Phi(\cdot)$	the feature function for each clause c
ϕ	a feature vector produced by the use of a clause
$\phi_{u\to v}$	a feature vector produced for reaching vertex v from u
$\mathbf{w}$	a parameter vector
$f(\mathbf{w},\phi)$	a weighting function
$\Phi_{\text{restart}}(R)$	a restart feature function
$\Phi_{\text{selfloop}}(R)$	a self-loop feature function
d	the depth in a proof graph
F_i	a feature literal
$\mathbf{ppr}(v_0)$	the personalized PageRank vector for v_0
$\mathbf{W}$	the transition probability matrix
$id(c)$	an identifier for the clause c
ℓ	a loss function
μ	the regularization constant for default stochastic gradient descent learning
$epoch$	the number of passes over data
β	the learning rate
η	the initial learning rate
RE	a relational path of a sequence of relations $r_1,...,r_\ell$

s	the seed node
$h_{s,RE}(e)$	a path-constrained random walk distribution
$P(e\|e';r_\ell)$	the probability of $e \leftarrow e'$ with a random walk of edge type r_ℓ
μ_1 and μ_2	regularization coefficients for PRA
b	the binary predicate being learned in PRA
$j_i(w)$	the per-instance objective function for PRA
M	the number of entities in a knowledge base
t	the index for epoch
$R(\mathbf{w})$	a regularization term
w	a scalar-valued weight component
$signum(\cdot)$	a sign function that extracts the sign of a real number
σ	a shrinkage value
δ	the total number of accumulated regularization update
ζ	the regularization constant for structured penalty
A	an adjacency matrix that indicates the pair-wise similarity among logic clauses
$\|RHS\|$	the number of predicates on the right-hand side of a clause
D	a degree matrix
L	a graph Laplacian matrix
g	the number of group in group Lasso
H	a hashtable of features from each logic clause
$\mathbf{x}_i, y_i$	a training example of features and label
$\mathbf{P}$	a low-rank matrix
$\mathbf{Q}$	a low-rank matrix
F	a target matrix to be approximated
$F_{(i,j)}$	a cell item in a matrix
ROW_i	the i-th row in the matrix

$\vec{p}_i$	the latent embedding vector for the i-th example
$\vec{q}_j$	the latent embedding vector for the j-th column
λ_P and λ_Q	regularization coefficients used in matrix factorization
E	a set of triples as examples
E^{tr}	training examples
E^{te}	testing examples
S	a set of learned clauses
$\vec{A}$	an answer vector
$\mathscr{R}$	a set of relations

Table A.1: The notion used throughout this thesis.

Appendix B

Derivations for Parameter Learning

B.1 Derivation: PPR and its derivative

Notation: $\mathbf{s}$ is seeds, M is a transition matrix, $\mathbf{w}$ are parameters, $\mathbf{p}^\infty$ is the PPR stationary distribution, $\mathbf{p}^t$ is a point in the power-rule iteration for computing PPR, and $\mathbf{d}^t$ is the partial derivative of $\mathbf{p}^t$ with regard to the parameters $\mathbf{w}$.

$$\mathbf{p}^{t+1} \equiv \alpha\mathbf{s} + (1-\alpha)\mathrm{M}^\top\mathbf{p}^t \tag{B.1}$$

$$\mathbf{d}^t \equiv \frac{\partial}{\partial\mathbf{w}}\mathbf{p}^t \tag{B.2}$$

Note $\mathbf{p}^t$ and $\mathbf{d}^t$ have different dimensions: while $\mathbf{p}^t_u$ is a scalar score for u under PPR, $\mathbf{d}^t_u$ is a vector, giving the sensitivity of that score to each parameter in $\mathbf{w}$.

M is defined as follows. There is a weight vector $\mathbf{w}$, and for an edge $u \to v$, there is a feature vector $\vec{\phi}_{uv}$, which is used to define a basic score s_{uv} for the edge, which is passed through a squashing function f, e.g., $f(x) \equiv e^x$, and then normalized to form M.

$$s_{uv} \equiv \vec{\phi}_{uv} \cdot \mathbf{w} \tag{B.3}$$

$$t_u \equiv \sum_{v'} f(s_{uv'}) \tag{B.4}$$

$$\mathrm{M}_{u,v} \equiv \frac{f(s_{uv})}{t_u} \tag{B.5}$$

We can define $\mathbf{d}^t$ recursively:

$$\mathbf{d}^{t+1} = \frac{\partial}{\partial \mathbf{w}} \mathbf{p}^{t+1} \tag{B.6}$$

$$= \frac{\partial}{\partial \mathbf{w}} \left(\alpha \mathbf{s} + (1-\alpha) \mathrm{M}^\top \mathbf{p}^t \right) \tag{B.7}$$

$$= (1-\alpha) \frac{\partial}{\partial \mathbf{w}} \mathrm{M}^\top \mathbf{p}^t \tag{B.8}$$

$$= (1-\alpha) \left((\frac{\partial}{\partial \mathbf{w}} \mathrm{M}^\top) \mathbf{p}^t + \mathrm{M}^\top \frac{\partial}{\partial \mathbf{w}} \mathbf{p}^t \right) \tag{B.9}$$

$$= (1-\alpha) \left((\frac{\partial}{\partial \mathbf{w}} \mathrm{M}^\top) \mathbf{p}^t + \mathrm{M}^\top \mathbf{d}^t \right) \tag{B.10}$$

Now let's look at $\frac{\partial}{\partial \mathbf{w}} \mathrm{M}$, which we denote dM below. Note that each dM_{uv} is a vector, again giving the sensitivity of the weight M_{uv} to each parameter in $\mathbf{w}$.

$$\mathrm{dM}_{uv} = \frac{\partial}{\partial \mathbf{w}} \frac{f(s_{uv})}{t_u} \tag{B.11}$$

$$= \frac{1}{t_u^2} \left(t_u \frac{\partial}{\partial \mathbf{w}} f(s_{uv}) - f(s_{uv}) \frac{\partial}{\partial \mathbf{w}} t_u \right) \tag{B.12}$$

To continue this, define **df** and **dt** as the vectors

$$\mathbf{df}_{uv} \equiv \frac{\partial}{\partial \mathbf{w}} f(s_{uv}) = f'(s_{uv}) \vec{\phi}_{uv} \tag{B.13}$$

$$\mathbf{dt}_u \equiv \frac{\partial}{\partial \mathbf{w}} t_u = \sum_{v'} \mathbf{df}_{uv'} \tag{B.14}$$

Note that $\mathbf{df}_{uv}$ has no more non-zero components than $\vec{\phi}_{uv}$, so it is sparse, and $\mathbf{dt}_u$ is also sparse, but somewhat less so.

We can continue the derivation as

$$\begin{aligned} \mathrm{dM}_{uv} &= \frac{1}{t_u^2}\left(t_u \frac{\partial}{\partial \mathbf{w}} f(s_{uv}) - f(s_{uv}) \frac{\partial}{\partial \mathbf{w}} t_u\right) && \text{(B.15)} \\ &= \frac{1}{t_u^2}\left(t_u \mathbf{df}_{uv} - f(s_{uv}) \mathbf{dt}_u\right) && \text{(B.16)} \end{aligned}$$

B.2 Computation

Computing M and dM is one pass over the graph, shown in Table B.1.

Then you can start with $\mathbf{p}^0 = \mathbf{d}^0 = \mathbf{0}$ and iterate

$$\begin{aligned} \mathbf{p}^{t+1} &= \alpha \mathbf{s} + (1-\alpha)\mathrm{M}^\top \mathbf{p}^t && \text{(B.17)} \\ \mathbf{d}^{t+1} &= (1-\alpha)\left(\mathrm{dM}^\top \mathbf{p}^t + \mathrm{M}^\top \mathbf{d}^t\right) && \text{(B.18)} \end{aligned}$$

In more detail, the iteration for the updates on $\mathbf{p}$ are shown in Table B.2.

B.3 Loss functions and lazy regularization

For SRW each example is a triple $(\mathbf{s}, P, N)$ where $\mathbf{s}$ is the seed distribution, $P = \{a^1, \ldots, a^I\}$ are the positive examples, and $N = \{b^1, \ldots, b^J\}$ are the negative examples. We use $\mathbf{p}$ for the PPR distribution starting at $\mathbf{s}$, and write $\mathbf{p}[u]$ for $\mathbf{p}_u$ if I run out of space for subscripts.

The loss function is

$$L(\mathbf{w}) \equiv -\left(\sum_{k=1}^{I} \log \mathbf{p}[a^k] + \sum_{k=1}^{J} \log(1 - \mathbf{p}[b^k])\right) + \mu R(\mathbf{w}) \qquad \text{(B.19)}$$

where $R(\mathbf{w})$ is the regularization, e.g. $R(\mathbf{w}) \equiv ||\mathbf{w}||_2^2$. This means loss increases as the objective function decreases, where the objective function is proportional to the probability of either hitting a positive solution or not-hitting a negative solution. Once we have $\mathbf{d}$, it is easy to compute the gradient of this as

1. For each node/row u

   (a) $t_u = 0$

   (b) $\mathbf{dt}_u = \mathbf{0}$, an all-zeros vector

   (c) For each neighbor v of u

      i. $s_{uv} = \mathbf{w} \cdot \vec{\phi}_{uv}$, a scalar.

         In detail: For $i \in \vec{\phi}_{uv}$: increment s_{uv} by $\mathbf{w}_i \vec{\phi}_i$

      ii. $t_u \mathrel{+}= f(s_{uv})$, a scalar

      iii. $\mathbf{df}_{uv} = f'(s_{uv})\vec{\phi}_{uv}$, a vector, as sparse as $\vec{\phi}_{uv}$

         In detail: For $i \in \vec{\phi}_{uv}$: set $\mathbf{df}_{uv,i} = \vec{\phi}_f * c$, where $c = f'(s_{uv})$

      iv. $\mathbf{dt}_u \mathrel{+}= \mathbf{df}_{uv}$, a vector, as sparse as $\sum_{v'} \vec{\phi}_{uv'}$

         In detail: For $i \in \mathbf{df}_{uv}$: increment $\mathbf{dt}_{u,i}$ by $\mathbf{df}_{uv,i}$

   Now $t_u = \sum_{v'} f(s_{uv'})$ and $\mathbf{dt}_u = \sum_{v'} \mathbf{df}_{uv'}$

   (d) For each neighbor v of u create the vector

$$\mathrm{dM}_{uv} = \frac{1}{t_u^2}\left(t_u \mathbf{df}_{uv} - f(s_{uv})\mathbf{dt}_u\right)$$

   Or in detail: For $i \in \mathbf{dt}_u$: $\mathrm{dM}_{uv,i} = \frac{1}{t_u^2}\left(t_u \mathbf{df}_{uv,i} - f(s_{uv})\mathbf{dt}_u, i\right)$.

   There aren't any dimensions i that are present in $\mathbf{df}_{uv}$ but not $\mathbf{dt}_u$, since $\mathbf{dt}_u$ is a summation. Also create the scalar

$$\mathrm{M}_{uv} = \frac{f(s_{uv})}{t_u}$$

   You can now discard the intermediate values like t_u, $\mathbf{dt}_u$, $\mathbf{df}_{uv}$.

Table B.1: Computing M and dM

Updating $\mathbf{p}$:

1. $\mathbf{p}^{t+1} = \mathbf{0}$
2. For each node u
   (a) $\mathbf{p}_u^{t+1} \mathrel{+}= \alpha \mathbf{s}_u$
   (b) For each neighbor v of u
      i. $\mathbf{p}_u^{t+1} \mathrel{+}= (1-\alpha)\mathrm{M}_{vu}\mathbf{p}_v^t$

Updating $\mathbf{d}$:

1. $\mathbf{d}^{t+1} = \langle \mathbf{0}, \ldots, \mathbf{0} \rangle$ — i.e., for each node u there is an all-zeros vector of weights.
2. For each node u
   (a) For each neighbor v of u
      i. For each i in dM_{vu}

$$\mathbf{d}_{u,i}^{t+1} \mathrel{+}= (1-\alpha)\mathrm{dM}_{vu,i}\mathbf{p}_v^t$$

      ii. For each i in $\mathbf{d}_u^t$

$$\mathbf{d}_{u,i}^{t+1} \mathrel{+}= (1-\alpha)\mathrm{M}_{vu}\mathbf{d}_{u,i}^t$$

Table B.2: Updates for $\mathbf{d}$ and $\mathbf{p}$

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}) \equiv -\left(\sum_{k=1}^{I} \frac{1}{\mathbf{p}[a^k]} \mathbf{d}[a^k] - \sum_{k=1}^{J} \frac{1}{1-\mathbf{p}[b^k]} \mathbf{d}[b^k] \right) + \mu \frac{\partial}{\partial \mathbf{w}} R(\mathbf{w}) \tag{B.20}$$

We can split this into two parts: the empirical loss gradient, which is

$$-\left(\sum_{k=1}^{I} \frac{1}{\mathbf{p}[a^k]} \mathbf{d}[a^k] - \sum_{k=1}^{J} \frac{1}{1-\mathbf{p}[b^k]} \mathbf{d}[b^k] \right)$$

and the regularization gradient,

$$\mu \frac{\partial}{\partial \mathbf{w}} R(\mathbf{w})$$

If an example does not contain all features then the empirical gradient will be a sparse vector, but the regularization gradient will be dense. So the following code is more efficient for SGD than just computation of the full gradient and taking a step in that direction.

1. Maintain a "clock" counter m which is incremented when each example is processed. Also maintain a history $\mathbf{h}_i$ which says, for each feature i, the last time t an example containing i was processed.
2. When a new example $(\mathbf{s}, P, N)$ arrives at time t
   (a) For each feature active in the example, initialize it, if necessary, and them perform the regularization-loss gradient update $t - \mathbf{h}_i$ times.
   (b) Perform the empirical-loss update, using the new weights.
3. When you finish learning at final time T, consider every feature i, and perform the regularization-loss gradient update $T - \mathbf{h}_i$ times. Then write out the final parameters.

Bibliography

[1] Babak Ahmadi, Kristian Kersting, and Scott Sanner. Multi-evidence lifted message passing, with application to pagerank and the kalman filter. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, 2011. 2.4

[2] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486, 2006. 2.1.2, 11

[3] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local partitioning for directed graphs using pagerank. *Internet Mathematics*, 5(1):3–22, 2008. 2.1.2, 11

[4] Gabor Angeli, Julie Tibshirani, Jean Y Wu, and Christopher D Manning. Combining distant and partial supervision for relation extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014. 1

[5] Bogdan Babych and Anthony Hartley. Improving machine translation quality with automatic named entity recognition. In *Proceedings of the 7th International EAMT workshop on MT and other Language Technology Tools, Improving MT through other Language Technology Tools: Resources and Tools for Building MT*, pages 1–8. Association for Computational Linguistics, 2003. 7.1

[6] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, 2011. 2.2.3

[7] Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren

Etzioni. Open information extraction for the web. In *IJCAI*, volume 7, pages 2670–2676, 2007. 7.2.5

[8] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *The Journal of Machine Learning Research*, 7:2399–2434, 2006. 5.2

[9] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75–79, 2007. 6.3.2

[10] Elena Bellodi and Fabrizio Riguzzi. Learning the structure of probabilistic logic programs. In *Inductive Logic Programming*, pages 61–75. Springer, 2012. 4.4

[11] Elena Bellodi and Fabrizio Riguzzi. Structure learning of probabilistic logic programs by searching the clause space. *CoRR*, abs/1309.2080, 2013. 4.4

[12] Islam Beltagy, Cuong Chau, Gemma Boleda, Dan Garrette, Katrin Erk, and Raymond Mooney. Montague meets markov: Deep semantics with probabilistic logical form. In *2nd Joint Conference on Lexical and Computational Semantics: Proceeding of the Main Conference and the Shared Task, Atlanta*, pages 11–21. Citeseer, 2013. 7.3.1

[13] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *AAAI*, 2011. 6.1, 6.2, 6.4.2, 6.4.2

[14] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013. 6.1, 6.2, 6.4.1, 6.4.2, 6.4.2

[15] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014. 6.2, 6.4.2, 6.4.2

[16] Matthias Brocheler, Lilyana Mihalkova, and Lise Getoor. Probabilistic similarity logic. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2010. 1.1, 2.4

[17] Jamie Callan, Mark Hoy, Changkuk Yoo, and Le Zhao. Clueweb09 data set, 2009. 3.1

[18] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010. 1.1, 1.3, 3.1, 5

[19] Bob Carpenter. Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. *Alias-i, Inc., Tech. Rep*, pages 1–20, 2008. 5.2

[20] Soumen Chakrabarti. Dynamic personalized PageRank in entity-relation graphs. In *Proceedings of the 16th international conference on World Wide Web*, 2007. 1.3, 2.2.1

[21] Wei-Sheng Chin, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. A fast parallel stochastic gradient method for matrix factorization in shared memory systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(1):2, 2015. 6.3.2

[22] William W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems*, 18(3):288–321, July 2000. 1.1

[23] William W Cohen. *Graph Walks and Graphical Models*. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2010. 17

[24] Mark Craven, Johan Kumlien, et al. Constructing biological knowledge bases by extracting information from text sources. In *ISMB*, volume 1999, pages 77–86, 1999. 7.2.5

[25] Andrew Cropper and Stephen H Muggleton. Can predicate invention in meta-interpretive learning compensate for incomplete background knowledge? *Proceedings of the 24th International Conference on Inductive Logic Programming*, 2014. 5.3, 7.3

[26] James Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 44 (3):245–271, 2001. 1.3, 2.1.3, 2.4, 4.4

[27] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th international joint*

conference on Artifical intelligence, 2007. 1.1, 2.4, 4.4

[28] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In *52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, MD, USA, June*, 2014. 7.2.5

[29] Jenny Rose Finkel and Christopher D Manning. Joint parsing and named entity recognition. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 326–334. Association for Computational Linguistics, 2009. 7.1, 7.2.5

[30] George Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of machine learning research*, 3:1289–1305, 2003. 5

[31] Claes Fornell and David F Larcker. Evaluating structural equation models with unobservable variables and measurement error. *Journal of marketing research*, pages 39–50, 1981. 7.3

[32] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. A note on the group lasso and a sparse group lasso. *arXiv preprint arXiv:1001.0736*, 2010. 5, 5.2

[33] Norbert Fuhr. Probabilistic datalog—a logic for powerful retrieval methods. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 282–290. ACM, 1995. 1.1

[34] Matt Gardner, Partha Pratim Talukdar, Bryan Kisiel, and Tom M Mitchell. Improving learning and inference in a large knowledge-base using latent syntactic cues. In *EMNLP*, pages 833–838, 2013. 7.3.1

[35] Matt Gardner, Partha Talukdar, and Tom Mitchell. Combining vector space embeddings with symbolic logical inference over open-domain text. In *2015 AAAI Spring Symposium Series*, 2015. 7.3.1

[36] Dan Garrette, Katrin Erk, and Raymond Mooney. Integrating logical representations with probabilistic information using markov logic. In *Proceedings of the Ninth International Conference on Computational Semantics*, pages 105–114. Association for Computational Linguistics, 2011. 7.3.1

[37] Lise Getoor and Ben Taskar. *Introduction to statistical relational learning*. MIT press, 2007. 7.1, 7.1

[38] Edward Grefenstette. Towards a formal distributional semantics: Simulating logical calculi with tensors. *arXiv preprint arXiv:1304.5823*, 2013. 7.3.1

[39] Bernd Gutmann, Angelika Kimmig, Kristian Kersting, and Luc De Raedt. Parameter estimation in problog from annotated queries. *CW Reports*, 2010. (document), 2.3, 2.8, 2.3.2

[40] Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. In *EMNLP*, 2015. 6.2

[41] Geoffrey E Hinton. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, pages 1–12. Amherst, MA, 1986. (document), 4.1

[42] Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 541–550. Association for Computational Linguistics, 2011. 3.1, 7.2.5

[43] Tuyen N Huynh and Raymond J Mooney. Discriminative structure and parameter learning for markov logic networks. In *Proceedings of the 25th international conference on Machine learning*, pages 416–423. ACM, 2008. 4.4

[44] Rodolphe Jenatton, Nicolas L Roux, Antoine Bordes, and Guillaume R Obozinski. A

latent factor model for highly multi-relational data. In *NIPS*, 2012. 6.2, 6.4.2, 6.4.2

[45] Abhay Jha and Dan Suciu. Probabilistic databases with markoviews. *Proceedings of the VLDB Endowment*, 5(11):1160–1171, 2012. 1.1

[46] Charles Kemp, Joshua B Tenenbaum, Thomas L Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *AAAI*, volume 3, page 5, 2006. 5

[47] Boonserm Kijsirikul, Masayuki Numao, and Masamichi Shimura. Discrimination-based constructive induction of logic programs. In *AAAI*, pages 44–49, 1992. 5, 5.1, 5.2

[48] Stanley Kok and Pedro Domingos. Learning the structure of markov logic networks. In *Proceedings of the 22nd international conference on Machine learning*, pages 441–448. ACM, 2005. 4, 4.1, 4.4

[49] Stanley Kok and Pedro Domingos. Statistical predicate invention. In *Proceedings of the 24th international conference on Machine learning*, pages 433–440. ACM, 2007. 5, 7.3

[50] Stanley Kok and Pedro Domingos. Learning markov logic networks using structural motifs. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 551–558, 2010. (document), 4, 4, 4.4

[51] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009. 6.3.2

[52] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001. 8

[53] Ni Lao and William W. Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 2010. 1.3, 2.4, 3.2, 3.2, 1, 4.4

[54] Ni Lao, Tom M. Mitchell, and William W. Cohen. Random walk inference and learning

in a large scale knowledge base. In *EMNLP*, pages 529–539. ACL, 2011. ISBN 978-1-937284-11-4. 3, 3.2, 4.4, 7.1

[55] Ni Lao, Amarnag Subramanya, Fernando C. N. Pereira, and William W. Cohen. Reading the web with learned syntactic-semantic inference rules. In *EMNLP-CoNLL*, pages 1017–1026. ACL, 2012. ISBN 978-1-937284-43-5. 7.2.5

[56] Saskia Le Cessie and JC Van Houwelingen. Ridge estimators in logistic regression. *Applied statistics*, pages 191–201, 1992. 5.2

[57] Jiwei Li, Alan Ritter, and Eduard Hovy. Weakly supervised user profile extraction from twitter. ACL, 2014. 7.1

[58] Yankai Lin, Zhiyuan Liu, and Maosong Sun. Modeling relation paths for representation learning of knowledge bases. *EMNLP*, 2015. 6.2, 6.4.2

[59] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, pages 2181–2187, 2015. 6.2, 6.4.2, 6.4.2

[60] J. W. Lloyd. *Foundations of Logic Programming: Second Edition*. Springer-Verlag, 1987. 2.1.1

[61] Daniel Lowd and Pedro Domingos. Efficient weight learning for markov logic networks. In *Knowledge Discovery in Databases: PKDD 2007*, pages 200–211. Springer, 2007. (document), 1.1, 2.3, 2.8, 2.3.2, 3.4.2, 7.2.4

[62] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008. 1

[63] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Knowledge Discovery and Data Mining*, pages 169–178, 2000. URL `citeseer.nj.nec.com/article/mccallum00efficient.html`. 2.3.2

[64] Lilyana Mihalkova and Raymond J Mooney. Bottom-up learning of markov logic network structure. In *Proceedings of the 24th international conference on Machine learning*, pages 625–632. ACM, 2007. 4.4

[65] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 6.1

[66] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009. 7.2.5

[67] Diego Mollá, Menno Van Zaanen, and Daniel Smith. Named entity recognition for question answering. *Proceedings of ALTW*, pages 51–58, 2006. 7.1

[68] Stephen Muggleton. Stochastic logic programs. *Advances in inductive logic programming*, 32:254–264, 1996. 1.3, 2.1.3

[69] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the fifth international conference on machine learning*, pages 339–352, 1992. 5.1

[70] Stephen Muggleton and Dianhuan Lin. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1551–1557. AAAI Press, 2013. 4, 4.4

[71] Stephen H Muggleton, Dianhuan Lin, Jianzhong Chen, and Alireza Tamaddoni-Nezhad. Metabayes: Bayesian meta-interpretative learning using higher-order stochastic refinement. 2014. 4.4

[72] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th international conference*

on machine learning (ICML-11), pages 809–816, 2011. 6.2, 6.4.2, 6.4.2

[73] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*, pages 271–280. ACM, 2012. 7.3.1

[74] Masaaki Nishino, Akihiro Yamamoto, and Masaaki Nagata. A sparse parameter learning method for probabilistic logic programs. In *StarAI*, 2014. 5.2

[75] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS. *Proceedings of the VLDB Endowment*, 4(6):373–384, 2011. 2.4, 4.4

[76] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *arXiv preprint arXiv:1106.5730*, 2011. 2.2.3

[77] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997. 5

[78] Larry Page, Sergey Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. In *Technical Report, Computer Science department, Stanford University*, 1998. 1.3, 2.1.2, 2.2.1, 2.2.1

[79] Hoifung Poon and Pedro Domingos. Joint inference in information extraction. In *Proceedings of the National Conference on Artificial Intelligence*, 2007. 1.1

[80] Hoifung Poon and Pedro Domingos. Joint unsupervised coreference resolution with markov logic. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 650–659. Association for Computational Linguistics, 2008. 1.1

[81] J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3): 239–266, 1990. 4.1

[82] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr:

Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461. AUAI Press, 2009. 6.3.2, 6.4.5

[83] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 2006. (document), 1.1, 2.4, 4, 7.2.4

[84] Sebastian Riedel, Limin Yao, and Andrew McCallum. Modeling relations and their mentions without labeled text. In *Machine Learning and Knowledge Discovery in Databases*, pages 148–163. Springer, 2010. 7.2.5

[85] Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M Marlin. Relation extraction with matrix factorization and universal schemas. In *NAACL-HLT*, 2013. 1.3, 1, 6.2, 7.1, 7.2.4, 7.2.4, 7.2.5, 7.3, 7.3.1

[86] Tim Rocktäschel, Matko Bosnjak, Sameer Singh, and Sebastian Riedel. Low-dimensional embeddings of logic. *ACL 2014*, page 45, 2014. 6.2, 7.3.1

[87] Jude Shavlik and Sriraam Natarajan. Speeding up inference in markov logic networks by preprocessing to reduce the size of the resulting grounded network. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09)*, 2009. 2.4, 4.4

[88] Jiaxin Shi and Jun Zhu. Building memory with concept learning capabilities from large-scale knowledge base. *NIPS 2015 Cognitive Computation workshop*, 2015. 6.2, 6.4.2

[89] Amit Singhal. Introducing the knowledge graph: things, not strings. *the official Google blog: http://googleblog.blogspot.com/2012/05/ introducing-knowledge-graph-things-not.html*, 2012. 3.1

[90] Parag Singla and Pedro Domingos. Entity resolution with markov logic. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, 2006. 2.3, 2.3.2, 2.4

[91] Parag Singla and Pedro Domingos. Memory-efficient inference in relational domains. In

Proceedings of the national conference on Artificial intelligence, 2006. 2.4, 4.4

[92] Parag Singla and Pedro Domingos. Lifted first-order belief propagation. In *Proceedings of the 23rd national conference on Artificial intelligence*, 2008. 2.4, 4.4

[93] Anders Skrondal and Sophia Rabe-Hesketh. *Generalized latent variable modeling: Multilevel, longitudinal, and structural equation models*. CRC Press, 2004. 7.3

[94] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems*, pages 926–934, 2013. 7.1, 7.2.4, 7.3.1

[95] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007. 1.1, 3.1

[96] Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D Manning. Multi-instance multi-label learning for relation extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 455–465. Association for Computational Linguistics, 2012. 7.2.5

[97] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, pages 93–128, 2006. 7.1

[98] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. 5, 5.2

[99] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast random walk with restart and its applications. In *ICDM*, pages 613–622. IEEE Computer Society, 2006. 2.2.1

[100] Guy Van den Broeck, Ingo Thon, Martijn van Otterlo, and Luc De Raedt. Dtproblog: A decision-theoretic probabilistic prolog. In *AAAI*, 2010. 2.4, 4.4

[101] Quan Wang, Bin Wang, and Li Guo. Knowledge base completion using embeddings and rules. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 1859–1865, 2015. 6.2

[102] William Yang Wang and William W Cohen. Joint information extraction and reasoning: A scalable statistical relational learning approach. In *ACL*, 2015. 1.3, 7.1

[103] William Yang Wang and William W Cohen. Learning first-order logic embeddings via graph rewriting. In *under review at IJCAI*, 2016. (document), 1.3

[104] William Yang Wang and Zhenhao Hua. A semiparametric gaussian copula regression model for predicting financial risks from earnings calls. In *Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, Baltimore, MD, USA, June 2014. ACL. 7.1

[105] William Yang Wang, Kathryn Mazaitis, and William W Cohen. Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *CIKM*, 2013. (document), 4.4, 5, 6.1

[106] William Yang Wang, Kathryn Mazaitis, and William W Cohen. Structure learning via parameter learning. *CIKM*, 2014. 1.3, 4, 4.2, 5, 5.1, 5.3, 5.3.1, 5.3.1, 6.3.1, 6.3.3, 7.1, 7.2.4

[107] William Yang Wang, Kathryn Mazaitis, and William W Cohen. Soft predicate invention with structured sparsity. In *IJCAI*, 2015. (document), 1.3

[108] William Yang Wang, Kathryn Mazaitis, Ni Lao, and William W Cohen. Efficient inference and learning in a large knowledge base: Reasoning with extracted information using a locally groundable first-order probabilistic logic. *Machine Learning*, 2015. (document), 1.3, 4, 3, 7.2.5

[109] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, pages 1112–1119. Citeseer, 2014. 6.2, 6.4.2, 6.4.2

[110] Zhuoyu Wei, Jun Zhao, Kang Liu, Zhenyu Qi, Zhengya Sun, and Guanhua Tian. Large-scale knowledge base completion: Inferring via grounding network sampling over selected instances. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1331–1340. ACM, 2015. 6.2

[111] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd international conference on World wide web*, pages 515–526. International World Wide Web Conferences Steering Committee, 2014. 7.1, 7.2.4

[112] Jason Weston, Antoine Bordes, Oksana Yakhnenko, Nicolas Usunier, et al. Connecting language and knowledge bases with embedding models for relation extraction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1366–1371, 2013. 7.2.5

[113] Benjamin Roth Tassilo Barth Michael Wiegand and Mittul Singh Dietrich Klakow. Effective slot filling based on shallow distant supervision methods. *Proceedings of NIST KBP workshop*, 2013. 1

[114] James Wogulis and Pat Langley. Improving efficiency by learning intermediate concepts. In *IJCAI*, pages 657–662. Citeseer, 1989. 5.1

[115] John Wright, Yi Ma, Julien Mairal, Guillermo Sapiro, Thomas S Huang, and Shuicheng Yan. Sparse representation for computer vision and pattern recognition. *Proceedings of the IEEE*, 98(6):1031–1044, 2010. 5

[116] Fei Wu and Daniel S Weld. Autonomously semantifying wikipedia. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 41–50. ACM, 2007. 7.2.5

[117] Fei Wu and Daniel S Weld. Open information extraction using wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 118–

127. Association for Computational Linguistics, 2010. 7.2.5

[118] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1): 49–67, 2006. 5, 5.2

[119] Martin Zinkevich, Alex Smola, and John Langford. Slow learners are fast. *Advances in Neural Information Processing Systems*, 22:2331–2339, 2009. 2.2.3

[120] Martin Zinkevich, Markus Weimer, Alex Smola, and Lihong Li. Parallelized stochastic gradient descent. *Advances in Neural Information Processing Systems*, 2010. 2.2.3