Efficient Random Walk Inference with Knowledge Bases

Ni Lao

CMU-LTI-12-010

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

Thesis Committee:

William W. Cohen, Chair
Teruko Mitamura
Tom Mitchell
C. Lee Giles, Pennsylvania State University

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

Dedication

To my great teachers for their passion and devotion.

"There are two ways to live:
you can live as if nothing is a miracle;
you can live as if everything is a miracle."
– Albert Einstein

Acknowledgments

The completion of this thesis is impossible without the support from my advisor, William W. Cohen. William is knowledgeable, easy going, and always fun to talk to. I sincerely thank him for allowing me to explore, and also nudging me back on track from time to time. I am also grateful for all his detailed suggestions on my writing.

I thank my committee members for their numerous suggestions and encouragements. Special thanks to Tom Mitchell, who was deeply involved in the knowledge base inference study in this thesis. When facing a problem, Tom always asks profound questions, which are great inspirations to me.

I thank my friends in LTI and CMU in general, who made my life at Pittsburgh full of laughter. Special thanks to my officemates Frank Lin and Le Zhao, for enjoyable discussions on white board, for laughing together at silly YouTube videos, and for finding good food together in Pittsburgh.

Finally, I thank my parents, Lianjian Gong and Linger Lao, and my wife, Wenyun Zuo, for their understanding, care, and love.

Abstract

Relational learning is a subfield of artificial intelligence, that learns with expressive logical or relational representations. In this thesis, I consider the problem of efficient relational learning. I describe a new relational learning approach based on path-constrained random walks, and demonstrate, with extensive experiments on IR and NLP tasks, how relational learning can be applied at a scale not possible before. This scalability is made possible by defining a family of easy-to-learn features, fast random walk methods, and distributed computing.

Contents

List of Figures

List of Tables

List of Tables

Glossary

$K = (C, R, T)$	a knowledge base graph
C	a set of concepts
R	a set of relation labels
$T \subseteq C \times R \times C$	the set of labeled edges (*triples*) (c, r, c')
$r \in R$	a relation label, e.g. *Father*, *Profession*
r^{-1}	the label which is semantically the inverse of label r
π	A path type, e.g. $\langle$*Father*, *Profession*$\rangle$
θ	a weight vector
ℓ	maximum path length
MRR	Mean Reciprocal Rank
MAP	Mean Average Precision

Chapter 1

Introduction

1.1 The Problem

Relational learning is the subfield of artificial intelligence, which considers both machine learning and knowledge representation. Machine learning studies systems that improve their behavior over time with experience (examples), which typically involves a search process through various generalizations of the examples. Much success has been had with techniques such as neural networks [42], decision trees [77], and support vector machines [18]. However, these approaches only handle data with a propositional representation, which is limited to objects with their attributes. A more expressive representation describes entities as well as the relationships that hold among them. These representations are often called *relational*, and when they are grounded in or derived from first-order logic they are called *logical representations*. These are intuitive ways of describing the world around us and describing human knowledge. Therefore, relational learning not only has a wide range of applications, but may also pave the road to better understanding of intelligence.

In this thesis, we work with a simplified knowledge base consisting of a set C of concepts and a set R of labels. Each label r denotes some binary relation partially

represented in the knowledge base. The concrete KB is a directed, edge-labeled graph $K = (C, R, T)$ where $T \subseteq C \times R \times C$ is the set of labeled edges (also known as *triples*) (c, r, c'). Each triple represents an instance $r(c, c')$ of the relation $r \in R$. Each concept may correspond to an entity in the world such as *CharlotteBronte*, or an abstract notion such as *Writer*. Each edge indicates that a relation exists between two entities such as *HasFather*(*CharlotteBronte*, *PatrickBronte*), or the category of an entity such as *IsA*(*CharlotteBronte*, *Human*). We denote by r^{-1} the inverse relation of r: $r(c, c') \Leftrightarrow r^{-1}(c', c)$. For instance *PeopleWithProfession*$^{-1}$ is equivalent to *ProfessionHasInstance*. It is convenient to take G as containing triple (c', r^{-1}, c) whenever it contains triple (c, r, c'). The KB may be incomplete, that is, $r(c, c')$ holds in the real world but $(c, r, c') \notin T$.

In particular, we consider a generic relational learning task called *link prediction*: given a directed edge-labeled graph KB representing background knowledge, a relation r, and a source node s (also called a query), find the set of nodes G, s.t. $r(s, t)$ for each t in G.

This kind of link prediction problem has been studied by the Inductive Logic Programming community. However, existing solutions cannot scale to large data sets mainly because of two intrinsic properties of relational learning:

- the need to explore an exponentially large feature space, and
- potentially exponentially many instantiations for each feature in a KB graph.

This thesis considers these efficiency problems of relational learning, so that relational learning can be applied to domains with large scale data.

1.2 Results

This thesis introduces *random walk inference*, which combines the expressiveness of logic, the robustness of statistical learning, and the scalability of random walk models. Random walk inference formulates relational learning as statistical classification problems—given a query answer node pair (s, t), generate random walk features that summarize their

relational neighborhood, and then train classifiers based on these features. These random walk features are denoted as $P(s \rightarrow t; \pi)$, the probability of reaching node t from node s through a particular path type π and following a particular random walk process. For example, when considering the question of "whether *Charlotte* has *Writer* as her *Profession*?", features such as

- $P(\textit{Charlotte} \rightarrow \textit{Writer}; \langle \textit{HasFather}, \textit{Profession} \rangle)$
- $P(\textit{Charlotte} \rightarrow \textit{Writer}; \langle \textit{Mention}, \textit{Mention}^{-1}, \textit{Profession} \rangle)$

are generated. These pathes belong to a very constrained subset of logical expressions (Horn clauses with chain structures), which can be efficiently learned. As will be described in later chapters, simple but effective metrics can be used to explore the large feature space, and the instantiation of such features can be computationally bounded using sampling based random walk approaches.

The main contribution of this thesis is to apply relational learning at scales not possible before. This scalability is made possible by

- a family of easy-to-learn path features (Chapter 2 and 7),
- fast random walk (Chapter 5), and
- distributed computing (Chapter 6).

1.3 Outline

Chapter 2 analyzes the nature of relational learning and identifies *feature type complexity*, and *feature instantiation complexity* as two main challenges for efficient relational learning. A path ranking algorithm (PRA), which combines the expressiveness of logic, the robustness of statistical learning, and the scalability of random walk models is introduced. Its relations to existing approaches are also discussed.

Chapter 3 applies PRA to an inference task. It considers the problem of performing learning and inference in a large scale knowledge base containing imperfect knowledge with incomplete coverage. We show that the Path Ranking Algorithm can be used to reliably infer new beliefs for the knowledge base [52].

Chapter 4 applies PRA to information retrieval in biology. It considers the problem of monitoring newly published literatures, and tasks encountered during the preparation of a new paper.

Chapter 5 discusses the problem of efficiency in random walk calculations, which corresponds to the instantiation complexity problem described in this chapter.

Chapter 6 describes how PRA can scale to large graphs which contain billions of parsed sentences, particularly for the task of extracting new beliefs from a large text corpora.

Chapter 7 discuss how PRA can effectively learn features that correspond to first order rules with constants as well as long relational paths. These features are applied to knowledge base inference tasks, and the problem of coordinate term extraction (i.e. given an initial set of concepts as seeds, discover more concepts of this type from parsed text).

Finally, chapter 8 concludes the thesis, and discusses promising future directions for relational learning based on random walks.

Chapter 2

The Framework

This chapter describes the basic path ranking algorithm, which combines the expressiveness of logic, the robustness of statistical learning, and the scalability of random walk models.

2.1 Motivation

2.1.1 Link Prediction in Knowledge Bases: An Example

One example of the link prediction task is to predict the profession of a person—e.g. Charlotte Bronte as shown in Figure 2.1. For this purpose, we might want to consider the professions of friends or family of this person—e.g. Charlotte's father is a writer. We might consider the professions of people with similar behavior in the past—e.g. both Charlotte and Charles wrote a novel, and Charles is a writer. We might also consider text mentioning these entities—e.g. if Charlotte is mentioned together with many other persons in the same sentences, and some of these persons are writers, and a few of them are painters. Although some of these rules are more useful than others, most of them are not very accurate. The relational learning task here is to efficiently find useful rules and

combine them to make an accurate prediction.

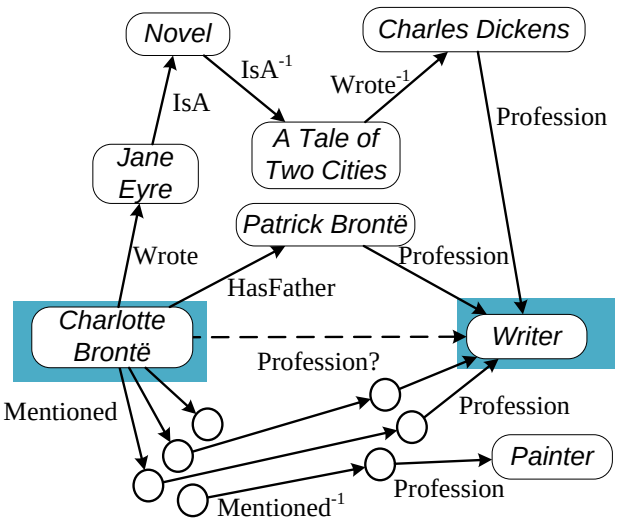


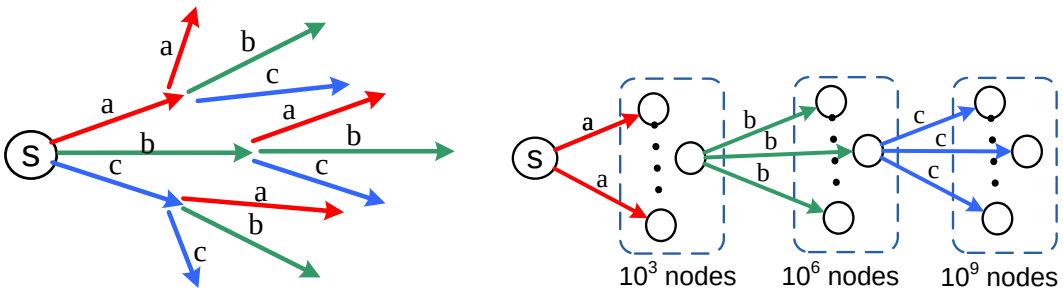
Figure 2.1: Link prediction in a knowledge base. IsA^{-1} is the inverse of *IsA*. $Wrote^{-1}$ is the inverse of *Wrote*. The *Mentioned* edges connect each entity to the sentences where this entity is mentioned

2.1.2 Why Is Relational Learning Challenging

From the link prediction task described above, one might conclude that the following three desirable characteristics of relational learning methods are desirable:

- Expressiveness—ability to define features expressing sequences of relations;
- Robustness—ability to combine many such features when making decisions;
- Scalablity—ability to efficiently discover and calculate such features.

The first two problems, *expressiveness* and *robustness*, have been studied extensively in the field of Inductive Logical Programming (ILP) [79], and Statistical relational learning (SRL) [34]. The third problem, *scalability*, though it has been studied from different angles in the past, is far from adequately addressed. Till now, both ILP and SRL methods have only been applied to relatively small domains. More specifically, the scalability problem has two parts, which we call *feature type complexity*, and *feature instantiation complexity*.



(a) (b)

Figure 2.2: Two efficiency challenges for relational learning: feature type complexity (a), and feature instantiation complexity (b)

Feature Type Complexity

The first efficiency challenge we call *feature type complexity* (Figure 2.2a). As in Chapter 1, we represent background knowledge as an edge-typed graph. Relational learning methods generally define features as *connectivity patterns*. For example, a pair of nodes s and t can be connected through edges sequences of type $\langle a, a\rangle$, $\langle a, b\rangle$, $\langle a, c\rangle$, etc. The number of such path types is exponential in their lengths. However, for complex domains useful path types are often of considerable sizes. For example, useful relational features in the language domain, where the underlying graph represents parsed texts, have been shown to involve edge sequences of length six or more [58]. This results in the problem of efficiently eliminating the many path types, that are useless for a task. Existing relational learning approaches such as FOIL [79] incrementally add edges to existing patterns (e.g. path types) to gradually expand the feature searching space. This procedure is often susceptible to local optima.

Feature Instantiation Complexity

The second efficiency challenge we call *feature instantiation complexity* (Figure 2.2b). Given a particular connectivity path type such as $\langle a, b, c\rangle$, there can be exponentially many

places in a graph where this path type is matched. This problem is acute for domains where there are nodes with high connectivity, such as a knowledge base with an *IsA* relation. Forced to consider each and every one of these path types instantiations, most relational learning methods are slow to perform inference.

2.1.3 Logic, Statistics, and Random Walks

Existing ILP systems such as First Order Inductive Learner (FOIL) [79] are expressive because they can learn first-order Horn clauses such as

$$\textit{HasFather}(a, b) \wedge \textit{Profession}(b, y) \longrightarrow \textit{Profession}(a, y),$$

which correspond to edge sequences in the graph representation of a knowledge base. However, these methods are not very robust or scalable—i.e. they lack a way to probabilistically combine low accuracy rule in the prediction process, and do not directly address the feature type and feature instantiation complexity problems.

Recently, there has been more and more work on Statistical Relational Learning (SRL), which seeks to combine statistical and relational learning, and has been applied to many domains such as social network analysis, reinforcement learning, natural language processing, and information extraction [34]. Most SRL approaches are defined in terms of directed or undirected graphical models [33, 86]. These are powerful joint inference formulations which give badly needed robustness to relational learning. However, they are still challenged by the complexity of relational inference and learning when applied to large scale problems, because again they do not directly address the problem of exploring an exponentially large relational feature space.

One type of very common and successful approach on graph data are random walk algorithms, for instance lazy random walks [59], personalized PageRank [10, 15], or Random Walk with Restart [98]. These algorithms produce a set of proximity measures based on random walk probabilities between nodes on a graph. Extensive study of

approximate algorithms for these tasks means that these probabilities can be estimated very efficiently. These methods are also robust in the sense that different types of edges in a knowledge base graph are used together to estimate these probabilities. However, regular graph-walk based similarity measures lack expressiveness, in the sense that the random walker does not distinguish the importance of different paths.

This thesis proposes *random walk inference*, which combines the expressiveness of logic, the robustness of statistical learning, and the scalability of random walk models.

2.2 Path Ranking Algorithm (PRA)

As described in Chapter 1, we work with a simplified knowledge base consisting of a set C of concepts, a set R of relation labels. Each label r denotes some binary relation partially represented in the KB. The concrete KB is a directed, edge-labeled graph $K = (C, T)$ where $T \subseteq C \times R \times C$ is the set of labeled edges (also known as *triples*) (c, r, c'). Each triple represents an instance $r(c, c')$ of the relation $r \in R$. The KB may be incomplete, that is, $r(c, c')$ holds but $(c, r, c') \notin T$. We consider a generic relational learning task called link prediction: given a directed edge-labeled graph K representing background knowledge, a relation r, and a source node s (also called a query), find the set of nodes G s.t. $r(s, t)$ for each $t \in G$.

2.2.1 Path Constrained Random Walks (PCRW)

A *path type* in K is defined as a sequence of edge types $\pi = \langle r_1, \ldots, r_m \rangle$, $r_i \in R$. An *instance* of π is a sequence of nodes $c_0, \ldots, c_m$ such that $r_i(c_{i-1}, c_i)$, $\forall i : 2 \leq i \leq m$. For instance, "the persons who were born in the same town as the query person", and "the nationalities of persons who were born in the same town as the query person" can be

reached respectively through paths matching the following types

$$\pi_1 : \langle \textit{BornIn}, \textit{BornIn}^{-1} \rangle$$
$$\pi_2 : \langle \textit{BornIn}, \textit{BornIn}^{-1}, \textit{Nationality} \rangle$$

Given a path type $\pi = \langle r_1, r_2, ..., r_\ell \rangle$, and a starting node $s = v_0$, we define $P(s \to t; \pi)$ as the probability of reaching t from s by a random walk that instantiates π. More specifically, suppose that the random walk has just reached v_i by traversing edges labeled $r_1, \ldots, r_i$. Then v_{i+1} is drawn at random from all nodes reachable from v_i by edges labeled r_{i+1}. A path type π is *active* for pair (s, t) if $P(s \to t; \pi) > 0$. This probability can be recursively defined

$$P(s \to t; \pi) = \sum_z P(s \to z; \pi') P(z \to t; r), \tag{2.2.1}$$

where r is the last relation in path π, and π' is its prefix, such that adding r to π' gives π. In case that π is the empty path ϕ (with zero length), $P(u \to v; \pi)$ is defined to be 1 if $u = v$, and 0 otherwise. The probability of following a particular relation $P(u \to v; r)$ is defined as $1/|r(u)|$ if $r(u, v)$, and 0 otherwise, where $r(u)$ is the set of nodes for which $r(u, v)$ is true. Using sampling techniques described in Chapter 5, this dynamic programming procedure can be done with bounded complexity, therefore we can efficiently calculate $P(s \to t; \pi)$ for a s together with all possible t's.

Note that by using this calculation procedure the summation $\sum_t P(s \to t; \pi)$ might be smaller than 1 due to the fact that certain nodes might not have out links of certain types. Therefore, strictly speaking, it is not a probabilistic distribution anymore.

2.2.2 Propositionalization Based on PCRW

A key idea for path ranking algorithm is that $P(s \to t; \pi)$ can be used as a feature for the task of prediction whether $r(s, t)$ is true. Specifically, let $Q = \{\pi_1, ..., \pi_n\}$ be a set of

path types that occur in the graph with $|\pi_i| \leq \ell$, where $|\pi_i|$ is the length of π, and θ_π a weight assigned to π, PRA's score for whether query node s is related to another node t via relation r is given by

$$score(s,t) = \sum_{\pi \in Q} P(s \rightarrow t; \pi)\theta_\pi. \tag{2.2.2}$$

This score encodes a PRA model's confidence that node s and t are connected by relation r. Note that given a query node s, this score function is likely to be sparse w.r.t. t (with $score(s,t)$=0 for many t's), because the PCRW probabilities $P(s \rightarrow t; \pi)$ are likely to be sparse w.r.t. t. The learning problem for PRA is to properly assign weights to different path types, so that the scoring function has high values for node ts where $r(s,t)$ is true, and low values for other node ts.

In general, this kind of technics that transform relational data into less expressive representations such as feature vectors is called *propositionalization*. The resulting problems can directly be solved by standard machine learning algorithms that require flat representations such as support-vector, and decision trees. PCRW is more efficient compared to traditional propositionalization methods such as average or summation [23], because PCRW can bound its computation complexity by sampling techniques.

2.2.3 Feature Selection for Random Walk Inference

Given a graph, the total number of path types is an exponential function of the maximum path length ℓ. Due to the large feature space, feature selection is required to allow effective learning in relational domains. First-order learning systems (e.g. FOIL[79], FOCL [71], and FORTE [84]) mostly rely on hill-climbing to explore a combinatorial model space, and are vulnerable to local maxima. PRA takes another approach, where a *goodness measure* that is inexpensive to calculate is first applied to rule out the majority of the features, and regularized learning methods (e.g., L1 and L2 regularized logistic regression)

serve as another selection mechanism, allowing PRA to process a relatively large number of features.

As preliminaries, we define several measures that can be derived from the random walk process. We organize our training data as a set of training queries $(s_i, G_i), i = 1...n$, where G_i is the set of "good" answers to query s_i, i.e. $G_i = \{t|r(s_i, t)\}$. The *accuracy* of a path type π reflects the probability of reaching any correct answer node following π

$$acc(\pi) = \frac{1}{n} \sum_i P(s_i \rightarrow G_i; \pi), \tag{2.2.3}$$

where $P(s_i \rightarrow G_i; \pi) \equiv \sum_{z \in G_i} P(s_i \rightarrow z; \pi)$ denotes the probability of reaching any correct answer in the training data following π.

The *hits* of a path π reflects the number of queries for which path π leads to any correct answer:

$$hits(\pi) = \sum_i I(P(s_i \rightarrow G_i; \pi) > 0), \tag{2.2.4}$$

where $I()$ is the indicator function.

In order for a path π to be included into a PRA model, it is required that $acc(\pi) \geq a$ and $hits(\pi) \geq h$, where the thresholds a and h are tuned empirically on training data. In addition to making the training more efficient, these constraints also have the nice effect of removing low quality path types. Both measures can be calculated efficiently, because the PCRWs can be calculated efficiently.

2.2.4 Generate Training Samples

For each relation r and a set of node pairs $\{(s_i, t_i)\}$, we can construct a training dataset $D = \{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i$ is a vector of all the path features for the pair (s_i, t_i). That is, the j-th component of $\mathbf{x}_i$ is $P(s_i \rightarrow t_i; \pi_j)$, and y_i is a boolean variable indicating whether $r(s_i, t_i)$ holds (i.e. if $t_i \in G_i$). We also assume that $\mathbf{x}_i$ contains a *bias feature*, with fixed

value 1.0. Following previous work [52, 60], we adopt a closed-world assumption – node pairs which are known to have relation r in the knowledge base are treated as positive examples, and all other pairs are treated as negative examples.

For most tasks, there are just a few "good" answers for each s_i, but thousands (or millions) of "bad" ones. Therefore using all of them in the objective function is expensive. Here we used a simple strategy similar to stratified random sampling [69]. First, the path types discovered in the previous (path discovery) step are used to construct an initial PRA model (all feature weights θ_π are set to 1); then, for each query node s_i, this model is used to retrieve candidate answer nodes, which are then sorted in descending order by their scores; finally, nodes at the $k(k+1)/2$-th positions are selected as negative samples, where $k = 0, 1, 2, ...$ This is helpful because, in generally, non-relevant entities highly ranked by this weak ranking function are more important than lower ranked ones. For in-depth comparisons of different selection strategies we refer the reader to Aslam et al.'s work [7].

2.2.5 Logistic Regression Training

Given a set of training samples, the parameters θ can be estimated by maximizing the following objective

$$\mathcal{F}(\theta) = \sum_i f_i(\theta) - \lambda_1 \|\theta\|_1 - \lambda_2 \|\theta\|_2^2 \tag{2.2.5}$$

where λ_1 and λ_2 respectively control the strength of the L_1-regularization, which helps with structure selection, and L_2-regularization, which prevents overfitting. Here $f_i(\theta)$ is given by

$$f_i(\theta) = y_i \ln p_i(\theta) + (1 - y_i) \ln(1 - p_i(\theta)), \tag{2.2.6}$$

and

$$p_i(\theta) \equiv P(y_i = 1 | \mathbf{x}_i; \theta) = \frac{\exp(\theta^T \mathbf{x}_i)}{1 + \exp(\theta^T \mathbf{x}_i)} \tag{2.2.7}$$

is the predicted probability. This composite L_1/L_2 norm is known as Elastic Net [102]. The maximization can be achieved using a special version of L-BFGS[1], which is designed to deal with non-differentiable L_1 norm [5].

In order to achieve good prediction accuracy, we first tune L_2 parameter on training data to achieve the best training accuracy, and then tune L_1 parameter so that the accuracy is not sacrificed.

2.2.6 Inference and Evaluation

After a model is trained for a relation r in the knowledge base, it can be used to produce new instances of r. We first generate unlabeled queries s which belong to the domain of r (queries which appear in the training set are excluded).

For each unlabeled query node s, we apply the trained PRA model to generate a list of candidate t nodes together with their scores by running inference on each path π with non-zero weight. We then sort all the prediction pairs (s, t) by their scores as in Eq. (2.2.2) in descending order, and evaluate the top ones manually.

For the purpose of parameter tuning, we would like to perform *cross-validation* with labeled queries. For each fold of cross-validation, we train a PRA model only using the rest of the queries, and then use this model to produce a ranked list of answers for each query in this fold. This collection of rank lists can be evaluated by information retrieval measures such as mean reciprocal rank (MRR) [2] and mean average precision (MAP)[3]. While MRR focuses on top part of the result list, MAP evaluates the overall quality of the result list, or how easy it is to find all the relevant documents.

[1] a commonly used second order optimization procedure in many machine learning problems

[2] For a set of queries Q, MRR is defined as the average inverse rank of the highest ranked relevant result in a set of results. $MRR = \frac{1}{|Q|}\sum_{q \in Q} \frac{1}{\text{rank of the first correct answer for } q}$ If the the first returned result is relevant, then reciprocal rank is 1.0, otherwise, it is smaller than 1.0.

[3] For a query Q, MAP is defined as the average precisions at the position of each relevant documents in a sorted result list.

Random Walk with Restart (RWR) (also called personalized PageRank [37]) is a general-purpose graph proximity measure which has been shown to be fairly successful for many types of tasks. In our experiments, we will compare PRA to two versions of RWR—an untrained RWR model and a trained RWR model. In brief, in the trained RWR model, the walker will probabilistically prefer to follow edges associated with different labels, where the weight for each edge label is chosen to minimize the same loss function as in Equation 2.2.5. In the untrained model, edge weights are uniform.

2.3 Comparison to Existing Approaches

2.3.1 Inductive Logic Programming

Since the very beginning of artificial intelligence, relational learning has been investigated [23]. Several cognitive scientists, such as a team involving the Nobel prize winner Herbert A. Simon [50], developed several models that explain how specific scientific theories could be obtained. Later studies [16, 88] show that logical representations are an excellent formalism to express the complexity of problems in different domains, and to employ the available background knowledge to obtain meaningful hypotheses. Muggleton [62] first defined the research field of Inductive Logic Programming (ILP) as the intersection of inductive concept learning and logic programming.

ILP uses logic programming as a uniform representation for examples, background knowledge and hypotheses. Given an encoding of the known background knowledge K, and a set of examples as ground facts E, an ILP system will derive a hypothesized logic program H which entails all the positive and none of the negative examples.

Various inductive logic programming systems such as FOIL [78], Golem [62], Progol [63], and Claudien [24] were developed. For instance, FOIL employs a separate-and-conquer rule-learning strategy: a set-covering approach, which repeatedly

searches for a rule covering many positive examples and none of the negative examples. Then the positive examples covered by the current clause are deleted, and this process is repeated until all positive examples have been covered. The rules are typically discovered by a general-to-specific heuristic search employing refinement operators. The idea of relational learning also became popular in the relational database research community and forms the subfield of (multi-)relational data mining (MRDM) [29].

In comparison, a PRA model is trained for each relation in a knowledge base, which corresponds to all the clauses that share the same head in a logic program. As in FOIL [78], PRA assumes that both background knowledge and examples are ground facts. This setting renders relational learning more like a function approximation problem, and thus more amiable to statistical machine learning techniques such as logistic regression.

Robustness

ILP systems are typically not robust, because they lack a way to probabilistically combine low accuracy rules for prediction. The goal of early ILP systems was to discover one hypothesis which is consistent—i.e., a hypothesis that covers all positive examples and no negative example, or, in IR terminology, has both high precision and recall. This goal is actually not attainable for many real life applications. Take the prediction of Charlotte Bronte's profession as an example: we might know that her father is a writer, we might also know that she is mentioned together with writers 60 percent of the time but with painters 40 percent of the time. However, none of these two rules is accurate enough to predict Charlotte's profession. A more accurate prediction might be made by combining many of such low precision rules— hence, PRA treats each of these hypothesis as a feature, and combines them with a logistic regression classifier. One might wonder how well would a logistic regression classifier perform, if we treat matched rules as binary features. It turns out that the *fractional values* of random walk features are very important to successful predictions (see Chapter 6). The feature values often represent some sort of voting of

importance—e.g. if Charlotte is mentioned together with many other persons with 80% of them being writers and 20% of them being painters, then the indication of her being a writer should be stronger than being a painter. However, this kind of weights are lost in binary features. Furthermore,

Furthermore, statistical learning theories are particularly important for relational learning, because having large hypothesis spaces is its nature. For realistic applications their learning problems are often under-determined—i.e. the number of training samples is smaller than the number of features even after some sort of feature selection routine is applies. For instance, there can be large number of hypothesis, which are correct on the training data set (or with very high accuracies), but perform poorly on the test set. From statistical learning theories we know that this is called overfitting, which can be dealt with by proper regularization to the model. PRA adds L_2-regularization to its logistic regression model, which performs quite well based on our experiments in several domains.

Inference Efficiency

Most PAC-learning results for relational learning are negative. Cohen and Page [17] distinguish three proof techniques for obtaining negative PAC-learning results. First, if the problem of finding a consistent hypothesis—i.e., a hypothesis that covers all positive examples and no negative examples—cannot be solved in polynomial time, then the problem is not PAC-learnable. For instance, it can be shown that the well-known SAT problem is polynomially reducible to the consistency problem for a particular language of patterns. A second proof technique, called *evaluation hardness*, states that a language cannot be PAC-learnable if there exists a concept in the language for which the covers test cannot be evaluated in polynomial time. A third proof technique, called *prediction-preserving reducibility*, directly reduces one PAC-learning problem to another. Testing whether a hypothesis covers a particular example is one of the key factors responsible for the computational complexity of ILP systems. It is typically NP-hard or

even undecidable in the general case, and repeated a large number of times in relational learning systems. Therefore, the overall performance of the relational learning systems can be significantly improved, if one can optimize the coverage test. PRA achieves this goal by doing approximated inference based on random walks, for which the computation can be bounded using sampling strategies. In order to define random walks over background knowledge, PRA also restricts background knowledge to be relations—allowing only ground predicates of arity two.

Model Expressiveness

For efficiency reasons, all systems that learn in first-order logic restrict the hypothesis space. Early work such as the Model Inference System [91] has a hypothesis space of logic programs, which is very computationally expensive. Later influential systems such as FOIL [78] restrict its hypothesis space to function-free Horn clauses. In both cases refinement operators, such as adding a term or replacing a variable with constant, are used to modify existing hypothesis in order to produce new hypothesis.

In comparison, PRA learns in a even more restricted hypothesis space—Horn clauses with chain structures (Chapter 7 will discuss how to learn chain structures, which contain constants). This hypothesis space is still expressive enough to cover many useful structures, but enables exhaustive enumeration and evaluation of all hypothesis above a goodness threshold with respect to a given set of examples. Being able to perform exhaustive search, PRA avoids the local mimina suffered by many relation learning approaches which use local search. However, it is an interesting future direction for PRA is to learn a less restricted hypothesis space—e.g. to learn clauses with tree structures.

2.3.2 Statistical Relational Learning

It has long been recognized that logical programs need robustness. Early approaches directly assign probabilities to the logical clauses, and in order to assign probability to derivations or proofs, probabilities of different clauses and different instantiations of the same clause are combined during inference by ad-hoc ways, which rely on certain independence assumptions. Stochastic logic programs [64] upgrade stochastic context-free grammars to definite clause logic. PRISM [89] and Probabilistic Horn Abduction [73] employ the assumption that the explanations for a goal are mutually exclusive; then the goal's probability is simply the sum of the probabilities of its explanations. Bayesian logic programs [46] assume that the explanations for a goal are mutually independent; then the goal's probability is simply the noisy-or or noisy-and of the probabilities of its explanations. Often, the knowledge-based model construction process takes into account a particular query to the network, and generates only that grounded part that is needed to answer the query. Sometimes, lifted-inference can reduce computation by collapsing many instantiations of the same pattern (path type in our case) into one instantiation with higher weight [74, 25].

Later approaches are more statistically rigorous by defining probabilities over possible worlds, and logical rules are only used to define the skeleton of either a directed or undirected graphical model. Probabilistic Relational Models [33] are based on Bayesian networks, for which parameters are easy to estimate, but the overall structure has to be restricted to directed acyclic graph. To reduce model complexity, aggregate functions (e.g. MIN, MAX, SUM, AVG), which are well-known from the database literature, are used to summarize possibly many instantiations of logical rules into a single observation. Markov Logic Networks [86] are based on Markov networks, for which parameters are hard to estimate due to inference complexity, but the overall structure is not restricted. Similar lifted-inference ideas can be applied [44, 93].

A key idea to all these approaches is that, in order to estimate the values of a

single ground fact, evidence from the instantiations of one or many clauses need to be all combined together. Different than non-probabilistic logical programming, where derivations can be terminated early, probabilistic logical programming fully explores all the derivation trees and combines them in order to achieve robustness. This is the instantiation complexity we described at the beginning of this chapter, and is one of the main obstacles to efficient statistical relational learning. Lifted inferences may reduce the complexity in certain applications, but, in most cases, the complexity of instantiation is still exponential to the size of rules.

PRA uses a sampling approach to deal with the instantiation complexity—i.e., the complexity of random walks can be bounded by the number of walkers (or particles). This also greatly reduces the feature type complexity problem, since inference is a subroutine for the evaluation and selection of rules. PRA defines efficient rule (feature) quality measures directly based on random walk results.

2.3.3 Random Walk Models

Many early approaches to the problem of retrieval on graph data are keyword-based database systems [4][43][10] based on proximity, which are designed mainly for ad-hoc queries, thus are not trainable for specific IR tasks. Another branch of works is using random walk on graphs as proximity measures, notably the PageRank [66] and the Personalized PageRank algorithms [37].

The efficiency of proximity search on graphs has been a concern for many previous systems. Most of them [80][38][20] build two-level representations of the graphs offline. Tong et al.[98] studied fast RWR methods based on low-rank matrix approximation, and graph partitioning. More recently, Chakrabarti [15] developed the HubRank algorithm, which precomputes offline the Personalized Pagerank Vectors (PPVs) for a small fraction of nodes, carefully chosen using query log statistics. It is not immediately apparent how to adapt these methods to path-constrained random-walk distance measures; hence, in this

study, we will focus on methods for maintaining a sparse representation of random walks.

On the other hand, there has been much follow up work in supervised learning of random walk models. Nie et al.[65] use simulated annealing to perform local search over each edge type, which is only applicable when the number of parameters is very small. Diligenti et al. [27] optimize relation weights using back-propagation, which has linear convergence, therefore requires many iterations to reach convergence. Agarwal et al. [2] applied efficient second order optimization procedure with preference labels of pairs of entities; however, instead of using real relevance data, document orderings generated from artificially manipulated random walk models were used.

One limitation to all these models is that by using one parameter per edge type, these models cannot leverage complex path features for relational data. Recently, there has been work on systems that learn to do graph retrieval with richer feature sets. Minkov et al. [59] showed that using n-grams of relation paths as reranking features can significantly improve the retrieval quality of a random walk based model. More recently, Minkov & Cohen[58] proposed a random walk based generative learning model that favors paths which are more likely to reach relevant entities. This thesis introduces a discriminative version of this learning technique that is based on Path Constrained Random Walks (PCRW), in which similarity is defined by a learned combination of constrained random walkers [52]. By avoiding the costly inference step (i.e. performing random walks) at each iteration of optimization, the training process of this work is significantly more efficient than that of Minkov et al. [58].

2.3.4 Propositionalization

The work of this thesis is most closely related to the relational learning approaches which do propositionalization—transform positive and negative examples into feature vectors. The LINUS system [56] first transforms facts in *deductive hierarchical database* (DHDB) form into feature vectors using utility predicates or utility functions. Then it induces a

concept description using attribute-value learning programs such as decision trees. Finally, these models are translated back to DHDB clauses corresponding to if-then rules. LINUS has a severely restricted hypothesis space (more restricted than PRA)—all variables in the body of a DHDB clause must appear in its head.

The structural Generalized Linear Regression (SGLR) framework [75] uses refinement graphs [91] and aggregation functions to summarize the relational neighborhood of a node, and then applies statistical learners such as logistic regression to this data representation. Statistical model selection criteria such as Bayesian Information Criterion is applied to evaluate each feature candidate. A major difference between PRA and SGLR is that PRA uses cheap feature goodness measures based on random walk results to evaluate a massive number of features without any model training. This allows us to exhaustively find all features which pass a certain goodness threshold, and include all of them in the model. In contrast, SGLR retrains its model for each candidate feature, and can thus consider only a small number of features. The aggregation functions of SGLR are also applicable to random walk inference, which is an interesting direction for future exploration.

2.4 Summary

In summary, no existing approach in the related fields possess all three desirable characteristics of relational learning — expressiveness, robustness, and scalablity. The logical programming approaches are expressive but not robust or scalable. The graphical model based SRL approaches are expressive and robust but not scalable. The random walk approaches are scalable and robust, but lacks of expressiveness. Therefore, we introduce *random walk inference*, which combines the expressiveness of logical programming, the robustness of statistical learning, and the scalability of random walk models.

Chapter 3

Case Study: Knowledge Base Inference

In this chapter we consider the problem of performing learning and inference in a large scale knowledge base containing imperfect knowledge with incomplete coverage. We show that Path Ranking Algorithm can be used to reliably infer new beliefs for the NELL knowledge base. More details on the experiments of this chapter can be found in our previous work [52].

3.1 Motivation

Although there is a great deal of recent research on extracting knowledge from text [3, 31, 95, 68, 9, 100], much less progress has been made on the problem of drawing reliable inferences from this imperfectly extracted knowledge. In particular, traditional logical inference methods [79] are too brittle to be used to make complex inferences from automatically-extracted knowledge, and probabilistic inference methods [86] suffer from scalability problems. Here we considers the problem of constructing inference methods that can scale to large knowledge bases (KB's), and that are robust to imperfect knowledge.

The KB we consider is a large triple store, which can be represented as a labeled,

Predicate	**Instance**
cityInState	(troy, Michigan)
musicArtistGenre	(Nirvana, Grunge)
tvStationInCity	(WLS-TV, Chicago)
sportUsesEquip	(soccer, balls)
athleteInLeague	(Dan Fouts, NFL)
starredIn	(Will Smith, Seven Pounds)
productType	(Acrobat Reader, FILE)
athletePlaysSport	(scott shields, baseball)
cityInCountry	(Dublin Airport, Ireland)

Table 3.1: Sample of NELL beliefs

directed graph in which each entity a is a node, each binary relation $r(a, b)$ is an edge labeled r between a and b, and unary concepts $C(a)$ are represented as an edge labeled "isa" between the node for the entity a and a node for the concept C. We apply the PRA algorithm to infer relations by combining the results of different random walks through this graph, and show that the method achieves good scaling properties and robust inference in a KB containing over 500,000 triples extracted from the web by the NELL system [13].

To evaluate our approach experimentally, we study it 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 read (extract structured facts from) the web [13]. NELL began operation in January 2010. As of March 2011, NELL had built a knowledge base containing several million candidate beliefs which it had extracted from the web with varying confidence (see Table 3.1 for examples). Among these, NELL had fairly high confidence in approximately half a million, which we refer to as NELL's *(confident) beliefs*. NELL had lower confidence in a few million others, which we refer to as its *candidate beliefs*.

3.2 Knowledge Base Inference with FOIL

Although NELL has now grown a sizable knowledge base, its ability to perform inference over this knowledge base is currently very limited. Until this work, its only inference method (beyond simple inheritance) involves applying first order Horn clause rules to infer new beliefs from current beliefs. For example, it may use a Horn clause such as

$$\textit{AthletePlaysForTeam}(a, b) \wedge \textit{TeamPlaysInLeague}(b, c) \Rightarrow \textit{AthletePlaysInLeague}(a, c) \tag{3.2.1}$$

to infer that *AthletePlaysInLeague(HinesWard,NFL)*, if it has already extracted the beliefs in the preconditions of the rule, with variables a, b and c bound to *HinesWard*, *PittsburghSteelers* and *NFL* respectively as shown in Figure 3.1. NELL had a set of approximately 600 such rules, which it has learned by data mining its knowledge base of beliefs. Each learned rule carries a conditional probability that its conclusion will hold, given that its preconditions are satisfied.

NELL learned these Horn clause rules using a variant of the FOIL algorithm [79], henceforth N-FOIL. N-FOIL takes as input a set of positive and negative examples of a rule's consequent (e.g., $+$*AthletePlaysInLeague(HinesWard,NFL)*, $-$*AthletePlaysInLeague(HinesWard,NBA)*), and uses a "separate-and-conquer" strategy to learn a set of Horn clauses that fit the data well. Each Horn clause was learned by starting with a general rule and progressively specializing it, so that it still covers many positive examples but covers few negative examples. After a clause is learned, the examples covered by that clause are removed from the training set, and the process repeats until no positive examples remain.

Learning first-order Horn clauses is computationally expensive—not only is the search space large, but some Horn clauses can be costly to evaluate [17]. N-FOIL uses two tricks to improve its scalability. First, it assumes that the consequent predicate is functional—e.g., that each *Athlete* plays in at most one

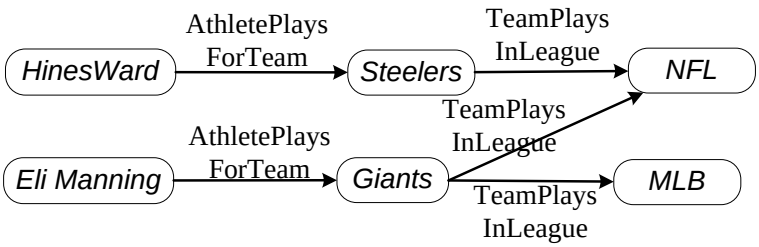


Figure 3.1: An example subgraph

League. This means that explicit negative examples need not be provided [101]: e.g., if *AthletePlaysInLeague(HinesWard,NFL)* is a positive example, then *AthletePlaysInLeague(HinesWard,*c'*)* for any other value of c' is negative. In general, this constraint guides the search algorithm toward Horn clauses that have fewer possible instantiations, and hence are less expensive to match. Second, N-FOIL uses "relational pathfinding" [85] to produce general rules—i.e., the starting point for a predicate r is found by looking at positive instances $r(a, b)$ of the consequent, and finding a clause that corresponds to a bounded-length path of binary relations that link a to b. In the example above, a start clause might be the clause (3.2.1). The clause is then (potentially) specialized by greedily adding additional conditions (like *ProfessionalAthlete*(a)) or by replacing variables with constants (eg, replacing c with *NFL*).

For each N-FOIL rule, an estimated conditional probability $\hat{P}(conclusion|preconditions)$ is calculated using a Dirichlet prior according to

$$\hat{P} = (N_+ + m \cdot prior)/(N_+ + N_- + m) \tag{3.2.2}$$

where N_+ is the number of positive instances matched by this rule in the FOIL training data, N_- is the number of negative instances matched, $m = 5$ and $prior = 0.5$. As the results below show, N-FOIL generally learns a small number of high-precision inference rules. One important role of these inference rules is that they contribute to the bootstrapping procedure, as inferences made by N-FOIL increase either the number of candidate beliefs, or (if the inference is already a candidate) improve NELL's confidence

in candidate beliefs.

3.3 Knowledge Base Inference with PRA

In this chapter, we consider an alternative approach, based on PRA. As an example, consider a path from a to b via the sequence of edge types *isa*, isa^{-1} (the inverse of *isa*), and *AthletePlaysInLeague*, which corresponds to the Horn clause

$$isa(a, c) \wedge isa^{-1}(c, a') \wedge AthletePlaysInLeague(a', b) \Rightarrow AthletePlaysInLeague(a, b) \tag{3.3.1}$$

Suppose a random walk starts at a query node a (say a=*HinesWard*). If *HinesWard* is linked to the single concept node *ProfessionalAthlete* via *isa*, the walk will reach that node with probability 1 after one step. If A is the set of *ProfessionalAthlete*'s in the KB, then after two steps, the walk will have probability $1/|A|$ of being at any $a' \in A$. If L is the set of athletic leagues and $l \in L$, let A_l be the set of athletes in league l: after three steps, the walk will have probability $|A_l|/|A|$ of being at any point $b \in L$. In short, the ranking associated with this path gives the prior probability of a value b being an athletic league for a—which is useful as a feature in a combined ranking method, although not by itself a high-precision inference rule.

Note that the rankings produced by this "expert" will change as the knowledge base evolves—for instance, if the system learns about proportionally more soccer athletes than hockey athletes over time, then the league rankings for the path of clause (3.3.1) will change. Also, the ranking is specific to the query node a. For instance, suppose the KB contains facts which reflect the ambiguity of the team name "Giants"[1] as in Figure 3.1. Then the path for clause (3.2.1) above will give lower weight to $b =$ *NFL* for $a =$ *EliManning* than to $b =$ *NFL* for $a =$ *HinesWard*.

[1] San Francisco's Major-League Baseball and New York's National Football League teams are both called the "Giants".

Table 3.2: Number of paths in PRA of maximum path length ℓ, averaged over 96 tasks

	ℓ=3	ℓ=4
all paths up to length ℓ	$15,376$	$1,906,624$
+query accuracy$\geq \alpha = 0.01$	522	5016
+query support ≥ 2	136	792
+L_1 regularization	63	271

Here we introduce and evaluate PRA as an algorithm for making probabilistic inference in large KBs. Compared to Horn clause inference, the key characteristics of this new inference method are as follows:

- The evidence in support of inferring a relation instance $r(s,t)$ is based on many existing paths between s and t in the KB, combined by a learned logistic function.
- The confidence in an inference is sensitive to the current state of the knowledge base, and the specific entities being queried.
- Experimentally, the inference method yields many more moderately-confident inferences than the Horn clauses learned by N-FOIL.
- The learning and inference are more efficient than N-FOIL, in part because we can exploit efficient approximation schemes for random walks (Chapter 5 [51]). The resulting inference is as fast as 10 milliseconds per query on average.

For each relation r in the knowledge base we train a model for the link prediction task: given a concept a, find all other concepts b which potentially have the relation $r(a,b)$. This prediction is made based on an existing knowledge base extracted imperfectly from the web. Although a model can potentially benefit from predicting multiple relations jointly, such joint inference is beyond the scope of this work.

To ensure a reasonable number of training instances, we generate labeled training example queries from 48 relations which have more than 100 instances in the knowledge base. We create two tasks for each relation—i.e., predicting b given a and predicting a given b— yielding 96 tasks in all. Each node a which has relation r in the knowledge

base with any other node is treated as a training query, the actual nodes b in the knowledge base known to satisfy $r(a, b)$ are treated as labeled positive examples, and any other nodes are treated as negative examples. Table 3.6 shows example names and number of training queries for a subset of relations.

Table 3.2 investigates how goodness measures help filtering out most of the useless features. Define a query s to be *supporting* a path P if $h_{s,P}(e) \neq 0$ for any entity e. We require that any path node created during path finding needs to have accuracy of at least a, as well as being of length no more than ℓ (In the experiments, we set $a = 0.01$) We also require that in order for a relation path to be included in the PRA model, it must retrieve at least $h = 2$ target entities t_i in the training set. We can see that together these two constraints dramatically reduce the number of relation paths that need to be considered, relative to systematically enumerating all possible relation paths. L1 regularization reduces the size of the model even more.

Table 3.3 shows the top two weighted PRA features for each task on which N-FOIL can successfully learn. These PRA rules can be categorized into broad coverage rules which set priors over answers (e.g. 1-2, 4-6, 15), accurate rules which leverage specific relation sequences (e.g. 9, 11, 14), rules which leverage information about the synonyms of the query node (e.g. 7-8, 10, 12), and rules which leverage information from a local neighborhood of the query node (e.g. 3, 12-13, 16). The synonym paths are useful, because an entity may have multiple names on the web. We find that all 17 general rules (no specialization of variables to constants) learned by N-FOIL can be expressed as length two relation paths such as path 11. In comparison, PRA explores a feature space with many length three paths.

Next we report empirical results of applying random walk inference to NELL's knowledge base after the 165th iteration of its learning process. We first investigate PRA's behavior by cross validation on the training queries. Then we compare PRA and N-FOIL's ability to reliably infer new beliefs, by leveraging the Amazon Mechanical Turk service.

Table 3.3: The top two weighted PRA paths for tasks on which N-FOIL discovers confident rules.

ID	PRA Path (Comment)
	athletePlaysForTeam
1	$\langle athletePlaysInLeague, leagueathletes, athletePlaysForTeam\rangle$ (teams with many athletes in the athlete's league)
2	$\langle athletePlaysInLeague, leagueTeams, teamAgainstTeam\rangle$ (teams that play against many teams in the athlete's league)
	athletePlaysInLeague
3	$\langle athletePlaysSport, athletes, athletePlaysInLeague\rangle$ (the league that athletes of a certain sport belong to)
4	$\langle isa, isa^{-1}, athletePlaysInLeague\rangle$ (popular leagues with all athletes)
	athletePlaysSport
5	$\langle isa, isa^{-1}, athletePlaysSport\rangle$ (popular sports of all athletes)
6	$\langle athletePlaysInLeague, superpartOfOrganization, teamPlaysSport\rangle$ (popular sports of a certain league)
	stadiumLocatedInCity
7	$\langle stadiumHomeTeam, teamHomeStadium, stadiumLocatedInCity\rangle$ (city of the stadium with the same team)
8	$\langle latitudeLongitude, latitudeLongitudeOf, stadiumLocatedInCity\rangle$ (city of the stadium with the same location)
	teamHomeStadium
9	$\langle teamPlaysInCity, cityStadiums\rangle$ (stadiums located in the same city with the query team)
10	$\langle teamMember, athletePlaysForTeam, teamHomeStadium\rangle$ (home stadium of teams which share athletes with the query)
	teamPlaysInCity
11	$\langle teamHomeStadium, stadiumLocatedInCity\rangle$ (city of the team's home stadium)
12	$\langle teamHomeStadium, stadiumHomeTeam, teamPlaysInCity\rangle$ (city of teams with the same home stadium as the query)
	teamPlaysInLeague
13	$\langle teamPlaysSport, athletes, athletePlaysInLeague\rangle$ (the league that the query team's members belong to)
14	$\langle teamPlaysAgainstTeam, teamPlaysInLeague\rangle$ (the league that the query team's competing team belongs to)
	teamPlaysSport
15	$\langle isa, isa^{-1}, teamPlaysSport\rangle$ (sports played by many teams)
16	$\langle teamPlaysInLeague, leagueTeams, teamPlaysSport\rangle$ (the sport played by other teams in the league)

Table 3.4: Comparing PRA with RWR models. MRRs and training times are averaged over 96 tasks. p-value=9×10^{-8} comparing untrained and trained RWR using paired t-test. p-value=4×10^{-4} comparing trained RWR with PRA using paired t-test.

	ℓ=2		ℓ=3	
	MRR	**Time**	**MRR**	**Time**
RWR(no train)	0.271		0.456	
RWR	0.280	3.7s	0.471	9.2s
PRA	0.307	5.7s	0.516	15.4s

3.4 Cross Validation on the Training Queries

We compare PRA to an untrained RWR model and a trained RWR model (defined in Chapter 2) on the 96 tasks of link prediction with NELL's knowledge base. We explored a range of values for the regularization parameters λ_1 and λ_2 using cross validation on the training data, and then fix both of them to 0.001 for all tasks. The maximum path length is fixed to 3.[2] Table 3.4 compares the three methods using 5-fold cross validation. We can see that supervised training can significantly improve retrieval quality, and leveraging path information can produce further improvement. Besides to its effectiveness, the average training time for a predicate is only a few seconds.

3.5 Evaluation by Mechanical Turk

The cross-validation result above assumes that the knowledge base is complete and correct, which we know to be untrue. To accurately compare PRA and N-FOIL's ability to reliably infer new beliefs from an imperfect knowledge base, we use human assessments obtained from Amazon Mechanical Turk. To limit labeling costs, and since our goal is to improve the performance of NELL, we do not include RWR-based approaches in this comparison. Among all the 24 functional predicates, N-FOIL discovers confident rules for 8 of them (it produces no result for the other 16 predicates). Therefore, we

[2] Results with maximum length 4 are not reported here. Generally models with length 4 paths produce slightly better results, but are one order of magnitude slower to train (see Chapter 7)

Table 3.5: Comparing Mechanical Turk workers' voted assessments with our gold standard labels based on 100 samples.

	AMT=F	AMT=T
Gold=F	25%	15%
Gold=T	11%	49%

compare the quality of PRA to N-FOIL on these 8 predicates only. Among all the 72 non-functional predicates—which N-FOIL cannot be applied to—PRA exhibits a wide range of performance in cross-validation. There are 43 tasks for which PRA obtains MRR higher than 0.4 and builds a model with more than 10 path features. We randomly sampled 8 of these predicates to be evaluated by Amazon Mechanical Turk.

For each relation r to be evaluated, we generate test queries s which belong to $domain(r)$. Queries which appear in the training set are excluded. For each query node s, we applied a trained model (either PRA or N-FOIL) to generate a ranked list of candidate t nodes. For PRA, the candidates are sorted by their scores as in Eq. (2.2.2). For N-FOIL, the candidates are sorted by the estimated accuracies of the rules as in Eq. (3.2.2). Since there are about 7 thousand (and 13 thousand) test queries s for each functional (and non-functional) predicate r, and there are (potentially) thousands of candidates t returned for each query s, we cannot evaluate all candidates of all queries. Therefore, we only evaluate the top ranked candidate t_i for each query s_i. They are first sorted in descending order by their scores, and then precision at the top 10, 100 and 1000 are calculated. To reduce the labeling load, we judge all top 10 results for each predicate, but randomly sample 50 out of the top 100, and randomly sample 50 out of the top 1000.

Each belief is evaluated by 5 workers at Mechanical Turk, who are given assertions like "*Hines Ward* plays for the team *Steelers*", as well as Google search links for each entity, and the combination of both entities. Statistics shows that the workers spend on average 25 seconds to judge each belief. We also removed some workers' judgments which are obviously incorrect[3]. We sampled 100 beliefs, and compared their voted result

[3]Certain workers labeled all the questions with the same answer

to gold-standard labels produced by the author. Table 3.5 shows that 74% of the time the workers' voted result agrees with our judgement.

Table 3.6 shows the evaluation result. The $P_{majority}$ column shows for each predicate the accuracy achieved by the majority prediction: given a query $r(a, ?)$, predict the b that most often satisfies r over all possible a in the knowledge base. Thus, the higher $P_{majority}$ is, the simpler the task. The #Query column shows the number of training queries for each task. Predicting the functional predicates is generally easier than predicting the non-functional predicates. On average N-FOIL is only able to produce results for 91 queries for each functional per predicate. In comparison, PRA is able to produce results for 6,599 queries on average for each functional predicate, and 12,519 queries on average for each non-functional predicate. Although the precision at 10 (p@10) of N-FOIL is comparable to that of PRA, precision at 100 and at 1000 (p@100 and p@1000) is much lower[4].

The #Path column shows the number of paths learned by PRA, and the #Rule column shows the number of rules learned by N-FOIL, with the numbers before brackets correspond to unspecialized rules, and the numbers in brackets correspond to specialized rules. Generally, specialized rules have much lower recall than unspecialized rules. Therefore, the PRA approach achieves high recall partially by combining a large number of unspecialized paths, which correspond to unspecialized rules. Learning more accurate specialized paths will be discussed in Chapter 7.

[4]If a method makes k predictions, and $k < n$, then p@n is the number correct out of the k predictions, divided by n

Table 3.6: Amazon Mechanical Turk evaluation for the promoted knowledge. Using paired t-test at task level, PRA is not statistically different from N-FOIL for p@10 (p-value=0.3), but is significantly better for p@100 (p-value=0.003)

			PRA				**N-FOIL**			
Task	**#Query**	$P_{majority}$	**#Paths**	**p@10**	**p@100**	**p@1000**	**#Rules**	**p@10**	**p@100**	**p@1000**
athletePlaysForTeam	498	0.07	125	0.4	0.46	0.66	1(+1)	0.6	0.08	0.01
athletePlaysInLeague	892	0.60	15	1.0	0.84	0.80	3(+30)	0.9	0.80	0.24
athletePlaysSport	1119	0.73	34	1.0	0.78	0.70	2(+30)	1.0	0.82	0.18
stadiumLocatedInCity	254	0.05	18	0.9	0.62	0.54	1(+0)	0.7	0.16	0.00
teamHomeStadium	186	0.02	66	0.3	0.48	0.34	1(+0)	0.2	0.02	0.00
teamPlaysInCity	135	0.10	29	1.0	0.86	0.62	1(+0)	0.9	0.56	0.06
teamPlaysInLeague	341	0.26	36	1.0	0.70	0.64	4(+151)	0.9	0.18	0.02
teamPlaysSport	339	0.42	21	0.7	0.60	0.62	4(+86)	0.9	0.42	0.02
average	0.28	43	0.79	0.668	0.615		91	0.76	0.38	0.07
teamMember	142	0.01	203	0.8	0.64	0.48				
companiesHeadquarteredIn	393	0.05	42	0.6	0.54	0.60				
publicationJournalist	68	0.02	25	0.7	0.70	0.64				
producedBy	134	0.19	13	0.5	0.58	0.68	N-FOIL does not produce results			
competesWith	226	0.19	74	0.6	0.56	0.72	for non-functional predicates			
hasOfficeInCity	398	0.03	262	0.9	0.84	0.60				
teamWonTrophy	149	0.24	56	0.5	0.50	0.46				
worksFor	363	0.13	62	0.6	0.60	0.74				
average	0.11	92	0.650	0.620	0.615					

A significant advantage of PRA over N-FOIL is that it can be applied to non-functional predicates. The last eight rows of Table 3.6 show PRA's performance on eight of these predicates. Compared to the result on functional predicates, precisions at 10 and at 100 of non-functional predicates are slightly lower, but precisions at 1000 are comparable. We note that for some predicates precision at 1000 is better than at 100. After some investigation we found that for many relations, the top portion of the result list is more diverse: i.e. showing products produced by different companies, journalist working at different publications. While the lower half of the result list is more homogeneous: i.e. showing relations concentrated on one or two companies/publications. On the other hand, through the process of labeling the Mechanical Turk workers seem to build up a prior about which company/publication is likely to have correct beliefs, and their judgments are positively biased towards these companies/publications. These two factors combined together result in positive bias towards the lower portion of the result list. In future work we hope to design a labeling strategy which avoids this bias.

3.6 Related Work

The TextRunner system [12] answers list queries on a large knowledge base produced by open domain information extraction. Spreading activation is used to measure the closeness of any node to the query term nodes. This approach is similar to the random walk with restart approach which is used as a baseline in our experiment. The FactRank system [45] compares different ways of constructing random walks, and combining them with extraction scores. However, the shortcoming of both approaches is that they ignore edge type information, which is important for achieving high accuracy predictions.

The HOLMES system [90] derives new assertions using a few manually written inference rules. A Markov network corresponding to the grounding of these rules to the knowledge base is constructed for each query, and then belief propagation is used for

inference. In comparison, our proposed approach discovers inference rules automatically from training data.

Similarly, the Markov Logic Networks [86] are Markov networks constructed corresponding to the grounding of rules to knowledge bases. In comparison, our proposed approach is much more efficient by avoiding the harder problem of joint inferences and by leveraging efficient random walk schemes [51].

3.7 Summary

We have shown that PRA can be used to reliably infer new beliefs for the knowledge base. We applied this approach to a knowledge base of approximately 500,000 beliefs extracted imperfectly from the web by NELL. This new system improves significantly over NELL's earlier Horn-clause learning and inference method: it obtains nearly double the precision at rank 100. The inference and learning are both very efficient—our experiment shows that the inference time is as fast as 10 milliseconds per query on average, and the training for a predicate takes only a few seconds.

Chapter 4

Case Study: Relational Retrieval in the Scientific Literature

The rapid growth of research in biology, and the increasing degree to which different subareas of biology are connected, make it difficult to monitor the published literature effectively. To address this problem, we develop a reading recommendation system that requires no other input from users except their reading or citation history. This frees the users from the problem of expressing their information need using query languages. We use a graph representation for publication databases with rich metadata. With this representation, a PRA model is trained to discover effective recommendation strategies, represented as edge paths on the graph. Experiments on citation-based and history-based reading recommendation tasks show that by leveraging rich context information the PCRW-based approach outperforms random walk with restart based approaches as well as traditional content-based and collaborative filtering approaches. Experiments on eight tasks in two subdomains of biology show that the new learning method significantly outperforms the RWR model (both trained and untrained) [52].

4.1 Motivation

One approach to accessing the scientific literature is to formulate it as a *recommendation task*. The goal of a recommender system is to generate meaningful recommendations to a collection of users for items or products that might interest them. Effective recommendation of scientific papers also relies relating the paper's content and metadata to biological background knowledge, such as the relationships known to hold among genes mentioned in the paper (and its metadata) and genes of interest to the end user. For example, Figure 4.1 illustrates several such strategies one might use to recommend new papers for a scientist to read. The relational task here is to efficiently find such strategies and combine them to make an informed recommendation.

One-parameter-per-edge label RWR proximity measures are limited because the *context* in which an edge label appears is ignored. For example, in the reference recommendation task, one of the query nodes is a year. There are two ways in which one might use a year y to find candidate papers to cite: (H1) find papers published in year y, or (H2) find papers frequently cited by papers published in year y. Intuitively, the second heuristic seems more plausible than the first; however, a system that insists on a using a single parameter for the "importance" of the edge label *PublishedIn* cannot easily encode this intuition.

The applications in this chapter involve multiple node types, and each relation has a predefined range and domain—e.g., edges of relation $PublishedIn$ always always connect nodes of type $paper$ to nodes of type $year$. In order to emphasize the types associated with each step in a path, we will write the path type $P = r_1 \dots r_\ell$ as

$$T_0 \xrightarrow{r_1} \dots \xrightarrow{r_\ell} \dots T_\ell$$

where $T_0 = dom(r_1) = dom(P)$, $T_1 = range(r_1) = dom(r_2)$ and so on. In this

notation,the two heuristics suggested above would be written as:

$$H1: \quad year \xrightarrow{PublishedIn^{-1}} paper$$
$$H2: \quad year \xrightarrow{PublishedIn^{-1}} paper \xrightarrow{Cite} paper$$

This notation makes it clear that the range of each relation path is *paper*, the desired type for reference recommendation.

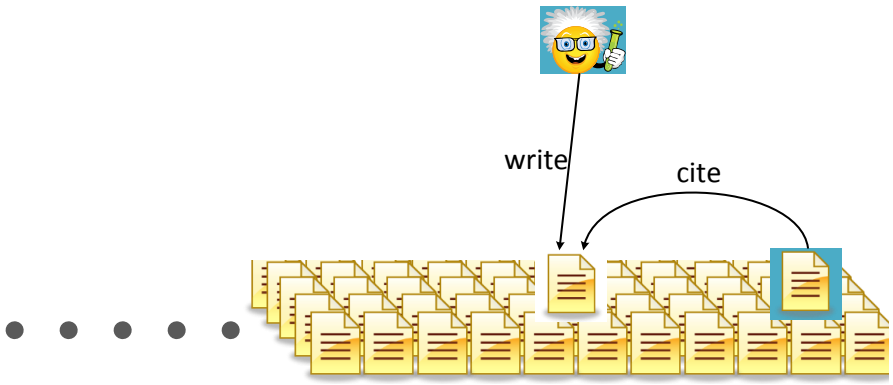
Different than the knowledge base inference task described in the previous chapter, a query for the recommendation task might contain different types of context information—e.g., a few search words given by the user, names of a few biology entities that the user finds relevant, or the identity of the user. Therefore, the link prediction task defined in Chapter 1 needs to be generalized. Instead of having a single source node s, each query has d sets of source nodes $[S_1, ..., S_d]$, $d \geq 1$, where each set contains source nodes of a certain type. For instance, a query $[\{w_1, ...w_W\}, \{e_1, ...e_E\}, \{u\}]$ consists of a set of nodes $\{w_1, ...w_W\}$ of type $word$ representing search words, a set of nodes $\{e_1, ...e_E\}$ of type $entity$ representing biology entities, and a node u of type $user$ representing the user who issued the query. The *target type* for this task is $paper$.

Given a candidate answer t, we define $P(S_i \rightarrow t; \pi)$ as the probability of reaching node t starting from any of the nodes in S_i with equal chance and following path π. Note that each source node set S_i may have its own set of paths, and a PRA model employs features of all these paths together.

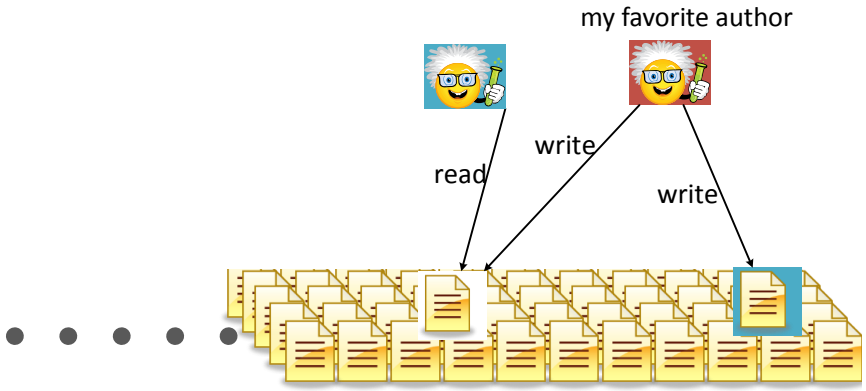
4.2 Datasets

We created two publication data sets (Yeast and Fly) in the biological domain.

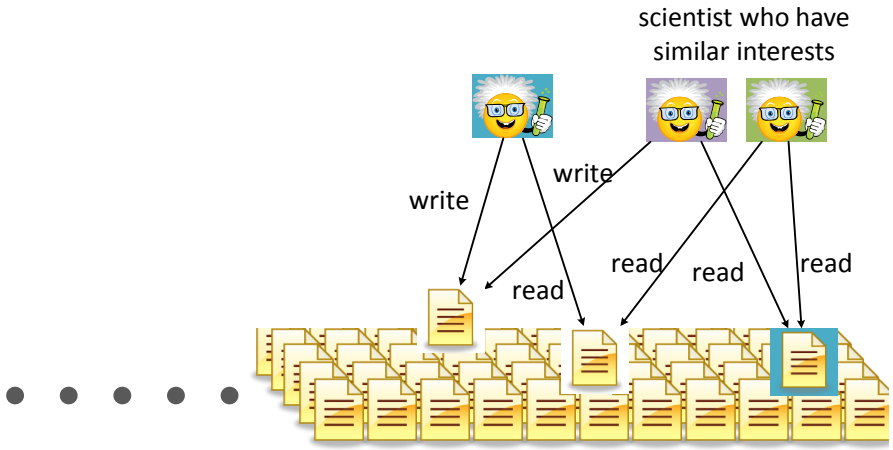
Figure 4.2 shows the schema of the yeast corpus. We extracted gene mentions from



(a) new development of an interesting topic—i.e., papers which cite my own papers



(b) papers of my favorite author—i.e., papers written by authors, who's paper I've read



(c) social recommendation—i.e., papers read by people who coauthored with me, or read the same papers as I did before

Figure 4.1: Strategies for recommending papers to a scientist from a stream of papers.

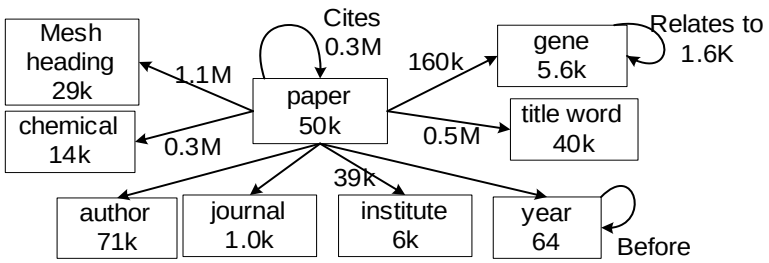


Figure 4.2: Schema of the yeast data.

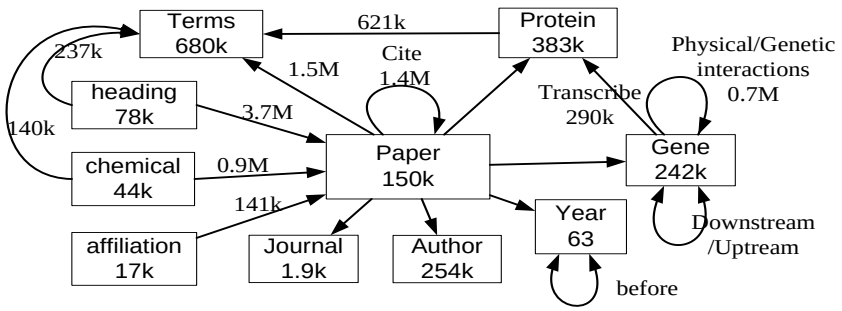


Figure 4.3: Schema of the fly data.

the *Saccharomyces Genome Database(SGD)*[1], which is a database of various types of information concerning the yeast organism *Saccharomyces cerevisiae*, including about 48K papers, each annotated with the genes it mentions. The title words are filtered by a stop word list of size 429. The *Authorship* relations are further distinguished into three sub-types: any author, first author, and last author. We extracted gene-gene relations from *Gene Ontology* (GO)[2], which is a large ontology describing the properties of and relationships between various biological entities across numerous organisms.

Figure 4.3 shows the schema of the fly corpus. It is extracted from *Flymine*[3], which is an integrated database for Drosophila and Anopheles genomics, and contains about 127K papers tagged with genes and proteins. The schema is similar to that of the yeast data,

[1]www.yeastgenome.org
[2]www.geneontology.org
[3]www.flymine.org

except for a new entity type *Protein*[4], and several relations among genes. *Downstream* and *Upstream* relation connect a gene to its two neighbors on the DNA strand.

Paper content and metadata information of both data sets are crawled from two resources: *PubMed*[5] is a free on-line archive of over 18 million biological abstracts for papers published since 1948; *PubMed Central* (PMC)[6] contains full-text and references to over one million of these papers.

4.3 Two Baseline Approaches

In order to compare the effectiveness of this random walk model to existing approaches, we first describe two commonly used recommendation approaches– content-based and community-based recommendations.

4.3.1 Content-based Recommendation

Recommendation systems are usually classified into *content-based* and *community-based* approaches [1]. Both of them have proved to be useful under different settings. The content-based recommendation approaches [8, 70, 49, 61] are rooted in information retrieval settings, where descriptive features can be extracted from the items, but no rating information from users other than the current one is needed. The users are recommended items similar to the ones the user preferred in the past.

More formally, each item i is represented by an item profile $Item(i)$, which is a vector of features. In our case of publication recommendation, each item is an paper, and each feature is a piece of metadata information such as author, venue, topic category, title word,

[4]In yeast, there is a nearly one-to-one relationship between genes and proteins, as most genes are transcribed to a unique protein; in flies, alternative splicing means that a gene can be transcribed to several different proteins.

[5]www.ncbi.nlm.nih.gov/pubmed

[6]www.ncbi.nlm.nih.gov/pmc

gene, chemical substance, etc. One of the best known weighting scheme for the features is the TF-IDF measure, in which the weight of the j-th feature to the k-th item is defined as

$$w_{k,j} = TF_{k,j} \cdot IDF_j = \frac{f_{k,j}}{\max_z f_{k,z}} \cdot \log \frac{N}{n_j}, \tag{4.3.1}$$

where $f_{k,j}$ is the number of times feature j appears in item k, N is the total number of items, and the j-th feature appears in n_j of them. Similarly, each user u is represented by a user profile $User(u)$, which is usually the average profile [8, 49] of all the items user u have liked. In our case of publication recommendation, it is the set of all papers that a user have read in the past. Finally, the utility of item i to user u is estimated by the similarity between their profile vectors, often measured by cosine function:

$$U(u, i) = cos(User(u), Item(i)). \tag{4.3.2}$$

4.3.2 Community-based Recommendation

The community-based recommendation approaches (also called collaborative filtering) [83, 35, 48, 82, 41, 92] predict the utility of items for a particular user based on the items previously rated by a sub-community, and no content information is assumed. The users are recommended items that people with similar tastes and preferences liked in the past.

More formally, the unknown rating $r_{u,i}$ for user u and item i is usually computed as an aggregation of the ratings of some other (usually, the K most similar) users for the same item i:

$$Rate(u, i) = aggr_{u' \in N(u,K)} Rate(u', i),$$

where $N(u, K)$ is the set of K users that are the most similar to user u (and who have rated item i). In the simplest case, the aggregation can be the average:

$$Rate(u, i) = \frac{1}{K} \sum_{u' \in N(u,K)} Rate(u', i).$$

However, the most common approach is a weighted sum:

$$Rate(u,i) = \frac{1}{Z} \sum_{u' \in N(u,K)} Sim(u,u') Rate(u',i), \tag{4.3.3}$$

where $Sim(u,u')$ is a similarity measure between user u and u', and Z is a normalization constant $\sum_{u' \in N(u,K)} Sim(u,u')$. In our case of publication recommendation, we can only observe if a user read a document or not, instead of rating scores. Therefore, we choose the cosine-based approach when calculating the similarities between users:

$$Sim(u,u') = \cos(f_u, f_{u'}), \tag{4.3.4}$$

where f_u is a binary vector indicating whether or not user u has read each of all the documents.

Furthermore, the content-based model's prediction for the current user can be combined with the prediction to its neighbors. We use a parameter α to control their relative importance, and the final scoring function has the form

$$Rate(u,i) = \alpha U(u,i) + \frac{1}{Z} \sum_{u' \in N(u,K)} Sim(u,u') U(u',i). \tag{4.3.5}$$

4.4 Citation-based Reading Recommendation

We define *citation-based reading recommendation* as recommending new publications which might interest a user given the publication history of this user. In order to train PRA in a supervised manner, here we describe how to generate training data automatically from an existing publication database.

For each user in a publication database, we assume that we know all the papers that he/she has published. From here on, we use the term user and author interchangeably. We also assume that each citation in a user's paper is somewhat related to his/her research interest. Therefore, predicting whether a paper is going to be cited by a particular user is approximately predicting whether the user is interested in this paper. We would like to analyze the importance of users' behavior information to the recommendation tasks, so we

only consider users who are recorded as publishing more than M papers in the database. This filtering process has the extra benefit of reducing the amount of training data and speeding up the training process.

For each year and each user, our task is to predict which papers from that year are going to be cited by the user. Each of the year-author pairs, for which the user has made any citations, is used to generate a query, and the set of papers actually cited are treated as the relevant targets. Papers written by the author him/herself are not included. We use these cited papers of a user to approximate his/her reading history, and we augment the graphs with one extra relation *Read*, which links a user to all the papers he/she has ever cited. We assume that the users always read a paper, if ever, the year after it was published. This way of generating training data is suitable for learning models which are good at suggesting the most recent publications to the users.

We set $M = 100$, and 1143 year-author queries are generated. We randomly reserve 1/3 of the queries for testing, and use the rest for parameter tuning (with 5-fold cross validation) and training. As a practical concern, we need to prevent the system from using information obtained later than a query when processing that query. Therefore, we define a *time variant graph* in which each edge in the graph is tagged with a time tag (the year in which this edge is added to the graph). When random walk is performed for a particular query, we only consider edges that are earlier than the query's year.

Figure 4.4 shows the effect of L_2- and L_1-regularization on recommendation quality on the tuning set. More complex models ($\ell = 3, 4$) have better accuracies. The L_2 regularizer is very important for preventing overfitting, and the L_1 regularizer can slightly improve recommendation quality. Though not shown, the L_1 regularizer can significantly reduce the number of features with non-zero weights. We report results for models up to maximum path length $\ell = 4$, because the random walk features of length 5 cannot be fit into the 24Gb memory of our machine. For all PRA experiment in this chapter, we fix hit and accuracy thresholds to $h = 1$ and $a = 0.01$ respectively.

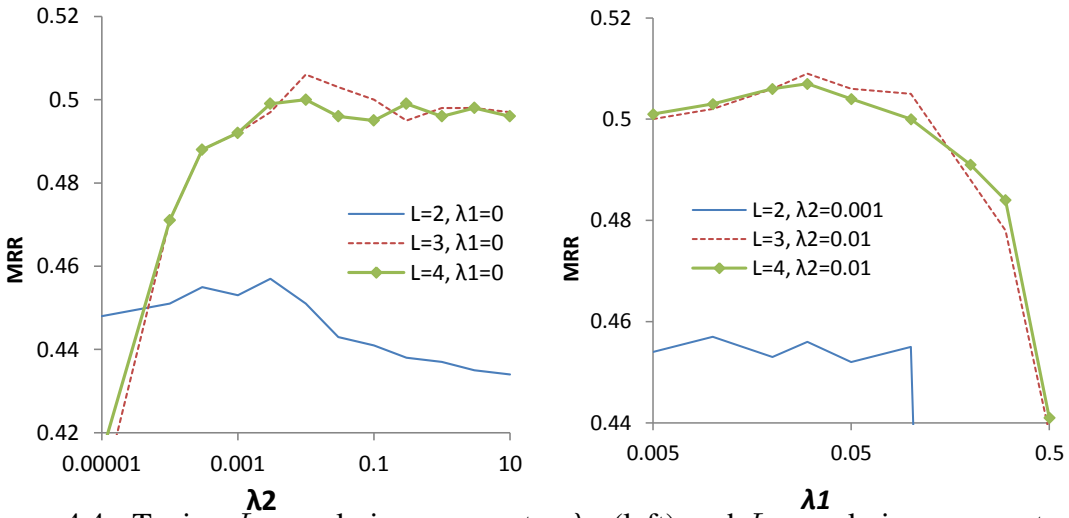


Figure 4.4: Tuning L_2-regularizer parameter λ_2 (left) and L_1-regularizer parameter λ_1 (right) for the citation-based reading recommendation on the yeast dataset. L is the maximum length of PRA paths.

Table 4.1 shows highest and lowest weighted features for the trained model of ℓ=3. All the high positive weight features are related to the user's reading or publication history. Most of these features correspond to content-based recommendation strategies (feature 1, 3, 4, 6, and 7)—first find papers read by the current user before, then find similar paper through shared contents like authors, genes, chemicals, MeSH descriptors, or title words. Interestingly, feature 2 takes a collaborative filtering approach—first find other users with similar reading history, and then find what they published in the current year.

All the high negative weight features (9-14) are independent of the user's behaviors but related to the recency of a paper's topics, authors, mentioned genes or chemicals. Take feature 13 as an example. If an paper p of the current year is written by an author who has published no paper in the past years, then a random walker starting from p and following the path $\text{paper} \xrightarrow{\text{Write}^{-1}} \text{author} \xrightarrow{\text{Write}} \text{paper}$ is very likely to return to paper p, so p would receive a relatively big negative weight. However, if the author has published many papers in the past years, the walker is very likely to end up in a paper from the past years, so the paper p would only receive a small negative weight in the PRA scoring function.

Table 4.1: Highest and lowest weighted paths for citation-based reading recommendation task on the yeast data. Maximum path length $\ell = 3$. θ_π shows the weights. *In* is a shorthand for the *PublishedIn* relation.

ID	θ_π	Path	Comments
1	4.8	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{Write}^{-1}}$ author $\xrightarrow{\text{Write}}$ paper	papers from my favorite authors
2	3.6	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{Read}^{-1}}$ author $\xrightarrow{\text{Write}}$ paper	papers of scientist who read what I read
3	2.1	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{HasGene}}$ gene $\xrightarrow{\text{HasGene}^{-1}}$ paper	papers about my favorite genes
4	1.4	author $\xrightarrow{\text{Write}}$ paper $\xrightarrow{\text{HasChem.}}$ chemical $\xrightarrow{\text{HasChem.}^{-1}}$ paper	papers about chemicals that I have worked on
5	1.4	author $\xrightarrow{\text{Write}}$ paper $\xrightarrow{\text{Read}^{-1}}$ author $\xrightarrow{\text{Write}}$ paper	papers from scientists who read my papers
6	1.4	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{HasMajorMD}}$ topic $\xrightarrow{\text{HasMD}^{-1}}$ paper	papers about my favorite MeSH descriptors
7	1.4	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{HasTitle}}$ word $\xrightarrow{\text{HasTitle}^{-1}}$ paper	papers with similar titles to what I read before
8	1.3	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{Cite}^{-1}}$ paper	follow up papers to what I read
	...		
9	-0.3	year $\xrightarrow{\text{In}^{-1}}$ paper $\xrightarrow{\text{HasMD}}$ topic $\xrightarrow{\text{HasMD}^{-1}}$ paper	papers of uncommon MeSH descriptors
10	-0.4	year $\xrightarrow{\text{In}^{-1}}$ paper $\xrightarrow{\text{HasGene}}$ gene $\xrightarrow{\text{HasGene}^{-1}}$ paper	papers of uncommon genes
11	-0.6	year $\xrightarrow{\text{In}^{-1}}$ paper $\xrightarrow{\text{HasMajorMQ}}$ topic $\xrightarrow{\text{HasMajorMQ}^{-1}}$ paper	papers of uncommon major MeSH qualifiers
12	-0.9	year $\xrightarrow{\text{In}^{-1}}$ paper $\xrightarrow{\text{HasMQ}}$ topic $\xrightarrow{\text{HasMQ}^{-1}}$ paper	papers of uncommon MeSH qualifiers
13	-1.2	year $\xrightarrow{\text{In}^{-1}}$ paper $\xrightarrow{\text{Write}^{-1}}$ author $\xrightarrow{\text{Write}}$ paper	papers of uncommon authors
14	-1.6	year $\xrightarrow{\text{In}^{-1}}$ paper $\xrightarrow{\text{HasChem.}}$ chemical $\xrightarrow{\text{HasChem.}^{-1}}$ paper	papers of uncommon chemicals

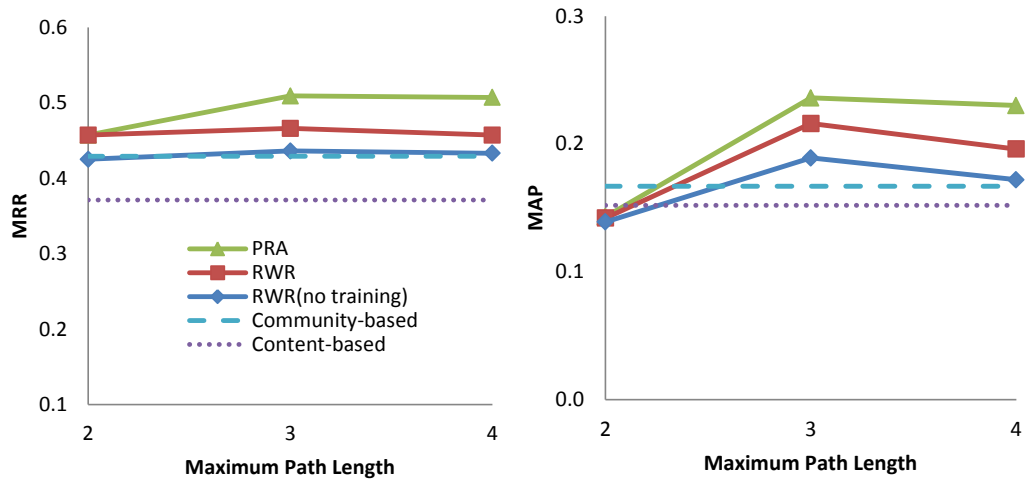


Figure 4.5: Comparing different approaches for the citation-based reading recommendation task on the yeast data.

Figure 4.5 shows the final evaluation results on the test queries. The community-based approach is noticeably better than the content-based approach, but close to the untrained RWR model. The trained RWR model performs better than the untrained version, but only to a certain extent. More complex versions (ℓ=3 and 4) of PCRW-based models are more accurate than RWR-based approaches and traditional recommendation approaches.

4.5 History-based Reading Recommendation

In the case where users' reading activities are accessible we define the following *history-based reading recommendation* task—given the papers read by a user for the past years, recommending new publications which might interest this user.

In this study a biologist (Dr. Woolford) whose major research interest is *Saccharomyces* volunteered to provide his reading information. We collect all the papers Dr. Woolford has read during 1988-2008 (the data were collected during 2009). We have found 364 papers from his computer, 265 of which can be matched in the yeast dataset. Since Dr. Woolford cannot remember exactly in which year he read each of the papers,

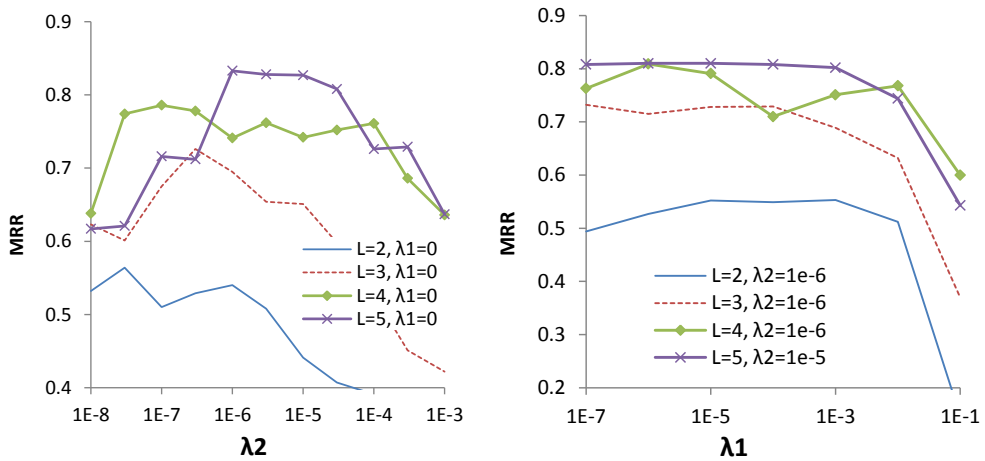


Figure 4.6: Tuning L_2-regularizer parameter λ_2 (left) and L_1-regularizer parameter λ_1 (right) for the citation-based reading recommendation on the yeast dataset. L is the maximum depth of relation trees.

we assume that he read each paper the following year after its publication. This way of generating training data is suitable for learning models, which are good at suggesting the most recent publications to the users. The yeast graph is augmented with two extra relations. One is the relation *Read*, which links a user (Dr. Woolford) to all the papers he has read in the past (265 papers). Another, with some notation overloading, is the *Read* relation, which links each year to the set of papers read by the user during that year. Dr. Woolford is recorded to have published 56 papers in the yeast database.

Our task is to predict for each year, which papers Dr. Woolford actually read, based on the papers he had read before that year. A query consisting of the user node (corresponding to Dr. Woolford) and the year node is generated for each year, and its relevant target nodes are the papers which are actually read by Dr. Woolford in that year. Therefore, we have 21 labeled queries. We tune parameters with leave-one-out cross validations on all 21 queries. We further evaluate the final model (trained on all queries) by Dr. Woolford's judgement of papers recommended for him to read in 2009.

Table 4.2: Top five weighted (non-zero) features for PRA models with different maximum path length ℓ trained for history-based reading recommendation task on the Yeast dataset.

ID	Path	Comments
ℓ=2		
1	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{Cite}^{-1}}$ paper	follow up papers to what I read
ℓ=3		
2	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{Write}^{-1}}$ author $\xrightarrow{\text{Write}}$ paper	papers from my favorite authors
3	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{Read}^{-1}}$ author $\xrightarrow{\text{Write}}$ paper	papers of scientist who read the same papers as I do
4	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{HasMajorMQ}}$ topic $\xrightarrow{\text{HasMajorMQ}^{-1}}$ paper	papers about my favorite topics
5	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{HasTitle}}$ word $\xrightarrow{\text{HasTitle}^{-1}}$ paper	papers with similar titles to what I read before
6	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{Cite}}$ paper $\xrightarrow{\text{Cite}^{-1}}$ paper	papers which cite the same papers as what I read
ℓ=4		
7	year $\xrightarrow{\text{After}}$ year $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{Cite}}$ paper $\xrightarrow{\text{Cite}^{-1}}$ paper	papers which share citations with what I read last year
8	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{Write}^{-1}}$ author $\xrightarrow{\text{Write}}$ paper	papers from my favorite authors
9	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{HasMajorMQ}}$ topic $\xrightarrow{\text{HasMajorMQ}^{-1}}$ paper	papers about my favorite topics
10	year $\xrightarrow{\text{After}}$ year $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{HasMD}}$ topic $\xrightarrow{\text{HasMD}^{-1}}$ paper	papers involving MeSH descriptors I read about last year
11	author $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{Read}^{-1}}$ author $\xrightarrow{\text{Write}}$ paper	papers of scientist who read the same papers as I do
ℓ=5		
12	year $\xrightarrow{\text{After}}$ year $\xrightarrow{\text{After}}$ year $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{HasMajorMQ}}$ topic $\xrightarrow{\text{HasMajorMQ}^{-1}}$ paper	papers which share MeSH descriptors with what I read 2 years back
13	year $\xrightarrow{\text{After}}$ year $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{Cite}}$ paper $\xrightarrow{\text{Read}^{-1}}$ author $\xrightarrow{\text{Write}}$ paper	papers by users who read what I cited last year
14	year $\xrightarrow{\text{After}}$ year $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{HasTitle}}$ word $\xrightarrow{\text{HasTitle}^{-1}}$ paper	papers which share title words with what I read last year
15	author $\xrightarrow{\text{Write}}$ paper $\xrightarrow{\text{HasMajorMQ}}$ topic $\xrightarrow{\text{HasMQ}^{-1}}$ paper	papers which share MeSH qualifier with what published
16	year $\xrightarrow{\text{After}}$ year $\xrightarrow{\text{After}}$ year $\xrightarrow{\text{Read}}$ paper $\xrightarrow{\text{HasTitle}}$ word $\xrightarrow{\text{HasTitle}^{-1}}$ paper	papers involving title words I read about 2 years back

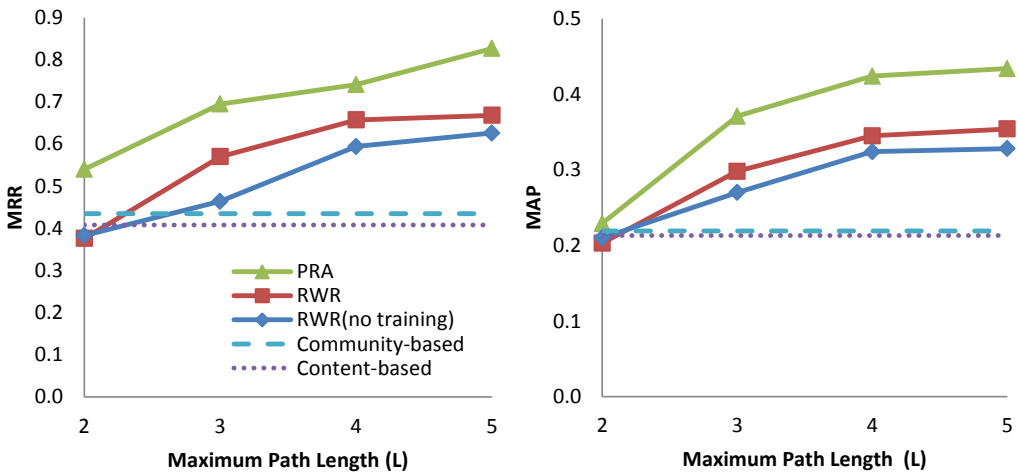


Figure 4.7: Comparing different approaches for the history-based reading recommendation task on the yeast data by MRR (left) and MAP (right). L is the maximum length of relation paths. $K = 30, \alpha = 0$.

Figure 4.6 shows the effect of L_2- and L_1-regularization to recommendation quality measured by MRR on the tuning query set. Although L_2 regularizer is very important for preventing overfitting, L_1 is less useful in this task. We can see that if L_2 regularizer is too small or too big MRR is significantly lower. This indicates that with very small number of training queries (21 for this task) the model can easily overfit or underfit.

Table 4.2 shows the top weighted features for models of different complexities. For the simplest type of model (ℓ=2), only one feature (beside the bias term) has non-zero weight. It leverages the user's reading behaviors in the past. More complex models (ℓ=4 and 5) are able to differentiate the user's reading information one, two, and three years back through the following paths

$$\text{author} \xrightarrow{\text{Read}} \text{paper}$$

$$\text{year} \xrightarrow{\text{After}} \text{year} \xrightarrow{\text{Read}} \text{paper}$$

$$\text{year} \xrightarrow{\text{After}} \text{year} \xrightarrow{\text{After}} \text{year} \xrightarrow{\text{Read}} \text{paper}$$

Figure 4.7 shows the final evaluation results. The community-based approach is

significantly better than the content-based approach, but close to the untrained RWR model. Again the trained RWR model performs better than the untrained version but only to a certain extent, and more complex versions (ℓ=4 and 5) of PCRW-based models are significantly more accurate than RWR-based approaches and traditional recommendation approaches. However, it would be an interesting future work to apply efficient path finding and feature selection techniques to explore features which correspond to longer paths.

We further applied the PRA model trained on all the queries to recommend papers for Dr. Woolford for 2009 and 2010. The number of relevant documents for the 2010 query is small, because the SGD database and PubMed papers were downloaded in 2010, when many papers of that year have not been included in the databases. For each query, results with negative scores (the bias feature is ignored) were removed, because they are even less relevant, judged by our model, than most documents in the database. We showed top 50 results from 2009 and 16 results from 2010 to Dr. Woolford, and he labeled each of them as relevant to his research interest or not. The accuracy judged by Dr. Woolford is p@50=0.82 and p@16=0.63 respectively , which is reasonably high for such a recommendation task.

4.6 Two Extensions to PRA

Here we describe two extensions to PRA, which are applicable to settings where there are few relations. Chapter 7 will describe generalizations to these extensions, which are applicable to more general settings such as knowledge base inference. The *query-independent experts* are generalizations to PageRank to multi-relational graphs. The *popular entity experts* directly capture correlations between query and answer nodes.

4.6.1 Query-Independent Experts

The basic PRA features describe an entity only in terms of its position in the graph relative to the query entities. However, the relevance of an entity may also depend on

query-independent qualities—for instance, its recency of publication, its citation count, or the authoritativeness of the venue in which it was published. To account for these intrinsic properties of entities, we extend every query set E_q to include a special entity e^*. We then extend the graph so that for each type T, there is a relation *AnyT* such that *AnyT*(e^*,e) is true for every $e \in T$. For example, the relation *AnyPaper* maps e^* to each paper, and the relation $AnyYear$ maps e^* to each year.

For example, the path $e^* \xrightarrow{AnyPaper} paper \xrightarrow{Cite} paper$ defines this random-walk process: start from any paper with equal probability, and then jump to one of its referenced papers. This results in higher probability mass to the papers with high citation count. A path that starts with *AnyPaper* and then follows two *Cite* edges assigns weight to papers frequently cited by other highly-cited papers: as path length increases, this type of *query-independent path* begins to approximate PageRank for papers on the citation graph. For a more involved example, a feature generated from the path $e^* \xrightarrow{AnyPaper} paper \xrightarrow{Cite} paper \xrightarrow{In} journal$ has higher weights on journals, which contain many well cited papers.

These query-independent features can express a rich set of semantics, and can be combined with query-dependent path features to rank the target entities. To use them, the scoring function remains unchanged. However, since these paths are query-independent, we can improve performance by computing their values for every entity offline. In particular, using the time-variant-graph, we calculate, for each year, the random walks scores for all query-independent paths, using only edges earlier than that year.

4.6.2 Popular Entity Experts

Previous work in information retrieval has shown that entity specific characteristics can be leveraged for retrieval [28, 99]. For the ad-hoc retrieval task, for instance, some lower ranked document got clicked very often by the users, because of features not captured by the ranking function of the system. In this case, promoting these popular documents to higher rank would result in a better user experience [99]. For personalized search [28],

different users may have different information needs for the same query: for instance, the word "mouse" can mean different things for a biologist and a programmer. In this case, modeling the correlation between query entities (users) and target entities (documents) can be useful.

In this work, we provide a simple yet general way of modeling entity popularities by adding biases and query-conditioned biases to the target entities. For a task with source node type T_0[7], and target type T_q, we introduce a *popular entity* bias θ_t^{pop} for each target entity $t \in T_q$. We also introduce a *conditional popular entity* bias $\theta_{s,t}$ for each query-target entity pair (s,t), where $s \in T_0, t \in T_q$. The scoring function in Eq.(2.2.2) is extended to

$$score(s,t) = \sum_{\pi \in B} P(s \to t; \pi) + \theta_t^{pop} + \sum_{e' \in E_q} \theta_{s,t}^{pop}, \theta_\pi. \tag{4.6.1}$$

or in matrix form $s = A\theta + \theta^{pop} + \Theta q$, where θ^{pop} is an concatenation of all bias parameters, Θ is an matrix of all conditional bias parameters, and q is a binary vector indicating whether each entity is included in the query.

We can see that the number of parameters is potentially very large. For example, θ^{pop} has length equal to $|T_q|$ (the total number of nodes of the target type), and Θ is a huge matrix with dimension $|T_q| \cdot |T_0|$ (the number of nodes of target and query entity type). Since it is impractical to include all of them to the model (consider the task of retrieving documents using words), we use an efficient induction strategy which only add the most important features [72]. After training an initial logistic regression model with only regular path features, we repeatedly add to PRA model the top J popular entity expert parameters, which have the largest gradient (in magnitude) w.r.t the objective function in Eq.(2.2.5), and retrain the model, until no more features to be added. We call J the *batch size*. Instead of considering all possible popular entity experts, we only need to consider those matched in the positive training examples. In our experiment, we found $J = 20$ gives relatively good performance. We also restrict the interactive process to be applied no more than 20

[7] Here we assume that there is only one type of source nodes. Discussion can be easily generalized to situations, where there are several types of source nodes.

times. In this way, the computation cost is not bounded by the size of θ^{pop} and Θ, but by the number of non-zero elements in them. In practice, we found that training a PRA model with popular entity experts is not much more expensive than training a regular PRA model; the details will be given in the next section.

4.7 Paper Completion Tasks

We consider four more tasks that a scientist may encounter during the process of publishing a paper. For all these tasks, we can generate large number of training queries from an existing publication database.

Venue recommendation is the problem of finding a venue to publish a new research paper. Here the query is a heterogeneous set of nodes: the terms in the title of the new paper, the set of entities (genes or proteins) associated with the paper, and the current year. The answer type is "journal", so the answer will be a list of biological journals, ranked by suitability for the new paper.

Reference recommendation (or citation recommendation) is the problem of finding relevant citations for a new paper. The query is, as in venue recommendation, the title terms and relevant entities for the new paper, the current year, and the answer type is "paper". The desired answer is a list of papers ranked by appropriateness as citations in the new paper. This task is similar to The TREC-CHEM Prior Art Search Task [57], and can also be seen as a simplified version of *context-aware citation recommendation* [39].

Expert finding is the problem of finding a domain expert to cooperate with for a particular topic. The query is again a list of terms and relevant entities, the current year, and the answer type is "person". The desired answer is a list of people with expertise on this topic.

The first two of these tasks are encountered in preparing a new paper, and the third

is encountered in finding reviewers, or new collaborators. To evaluate performance on these tasks, we will compare the ranked list from a query associated with a paper to the actual metadata associated with the paper: specifically, we will compare actual venue to the recommended venues, and the actual citations to the recommended citations. Perhaps more speculatively, we will also compare the authors of a paper to the experts recommended by the query based on the title and related-entity set for the paper. In each case the predictions will be made using a graph that does not contain the actual paper in question—see the next subsection for details.

As a fourth task, we will consider the *gene recommendation* task considered by Arnold and Cohen [6]—i.e., predicting, given past publishing history, which genes an author will publish about over the next year. Here the query nodes are an author and a year, and the expected answer is of type "gene". This task is an approximation to predicting future interests.

Each paper can be used to simulate a query and relevance judgements for any of the four above mentioned tasks. For each task on any of the two corpora, we randomly hold out 2000 queries for development, and another 2000 queries for testing. We evaluate models by Mean Average Precision (MAP).

4.7.1 Parameter Tuning on Development Data

In this subsection, we tune PRA parameters for reference recommendation task on the yeast data. Other tasks have similar trends, but their plots are not shown here.

Figure 4.8 shows the relation between maximum path length and model complexity (measured by number of features). We can see that both model complexity and query execution time are exponential in the path length. The query independent path (qip) extension introduces about twice as many paths than the basic PRA algorithm. Since their random walks are performed offline, they do not significantly affect query execution

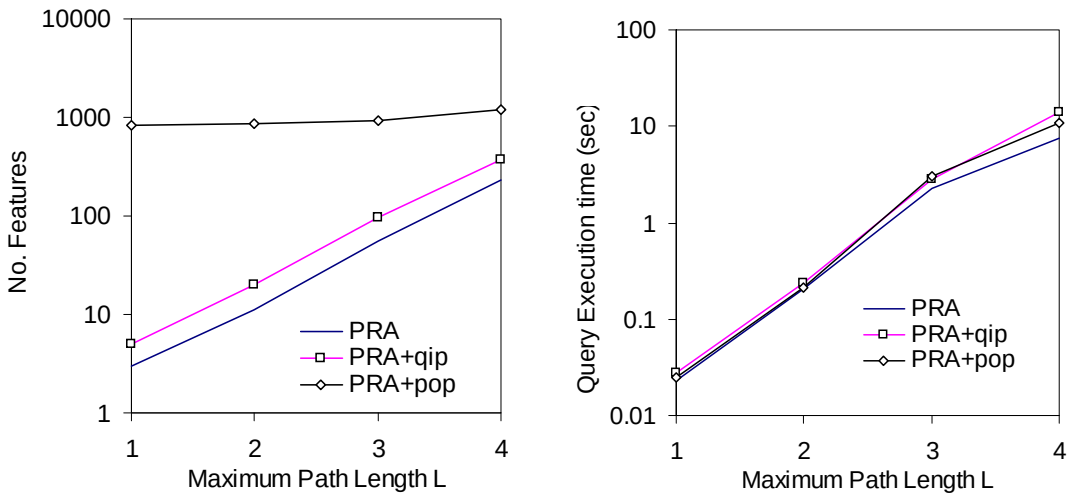
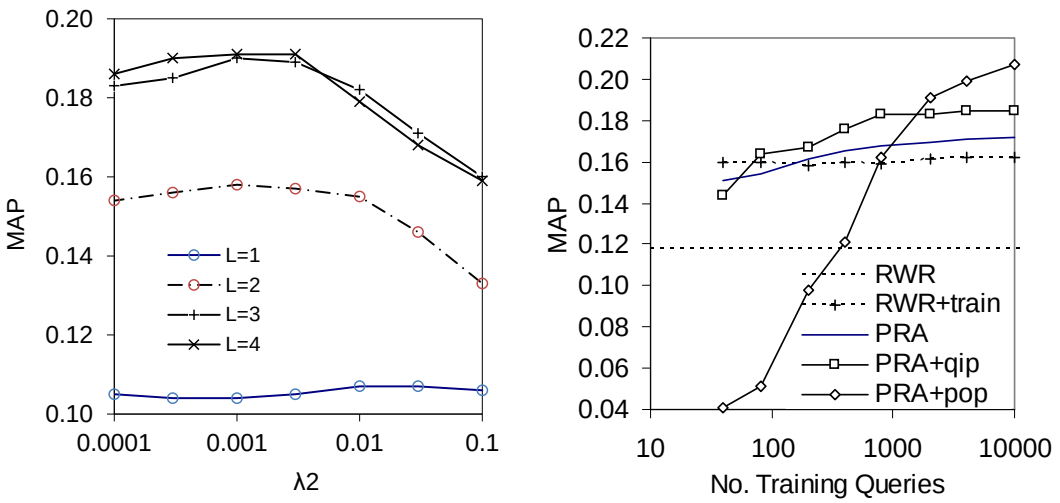


Figure 4.8: Model complexity verses maximum path length ℓ for reference recommendation task on the yeast data. Execution time is an average of 2000 test queries.

time. Although the popular entity experts introduce a large number of features, they are easy to calculate, and do not significantly affect query execution time.



(a) (b)

Figure 4.9: (a) Compare different regularization parameter λ_2 and different path length ℓ. (b) Effect of training data size for reference recommendation task on yeast data

Table 4.3: Subset of features from a PRA+qip+pop model trained for reference recommendation task on yeast data.

ID	θ_π	Path	Comments
1	272.4	word $\xrightarrow{\text{HasTitle}^{-1}}$ paper $\xrightarrow{\text{Cite}^{-1}}$ paper $\xrightarrow{\text{Cite}}$ paper	Papers cited together with papers which have title words matched with the query
2	156.7	word $\xrightarrow{\text{HasTitle}^{-1}}$ paper $\xrightarrow{\text{Cite}}$ paper	Papers cited by papers which have title words matched with the query
3	100.5	gene $\xrightarrow{\text{HasGene}^{-1}}$ paper $\xrightarrow{\text{Cite}^{-1}}$ paper $\xrightarrow{\text{Cite}}$ paper	Papers cited together with papers which have gene entities matched with the query
4	83.7	word $\xrightarrow{\text{HasTitle}^{-1}}$ paper $\xrightarrow{\text{Cite}^{-1}}$ paper	Papers which cite papers which have title words matched with the query
5	50.2	gene $\xrightarrow{\text{HasGene}^{-1}}$ paper $\xrightarrow{\text{Cite}}$ paper	Papers cited by papers which have gene entities matched with the query
6	41.4	word $\xrightarrow{\text{HasTitle}^{-1}}$ paper	Papers that have title words matched with the query
7	29.3	year $\xrightarrow{\text{In}^{-1}}$ paper $\xrightarrow{\text{Cite}}$ paper	Papers which are commonly cited by this year's paper
8	13.0	year $\xrightarrow{\text{Before}^{-1}}$ year $\xrightarrow{\text{In}^{-1}}$ paper $\xrightarrow{\text{Cite}}$ paper	Papers which are commonly cited by last year's paper
9	3.7	e* $\xrightarrow{\text{AnyPaper}}$ paper $\xrightarrow{\text{Cite}}$ paper	Papers which are commonly cited by any paper
10	2.9	GAL4 > Nature. 1988. GAL4-VP16 is an unusually potent transcriptional activator.	Cite a particular Nature paper if gene GAL4 is in the query
11	2.1	CYC1 > Cell. 1979. Sequence of the gene for iso-1-cytochrome c in *Saccharomyces cerevisiae*.	Cite a particular Cell paper if gene CYC1 is in the query
12	-5.4	year $\xrightarrow{\text{Before}^{-1}}$ year $\xrightarrow{\text{In}^{-1}}$ paper	Papers published in last year
13	-39.1	year $\xrightarrow{\text{In}^{-1}}$ paper	Papers published in this year
14	-49.0	e* $\xrightarrow{\text{Anyyear}}$ year $\xrightarrow{\text{In}^{-1}}$ paper	Old papers—since there are fewer papers published each year in the past, they are getting higher weights

Figure 4.9a shows the effect of L_2-regularization and path length on retrieval quality. We can see that a small amount of L_2-regularization can slightly improve MAP, and longer path lengths give better performance, but only to a certain level. In order to balance between retrieval quality and computational complexity, we fix max path length, for the rest of the experiment, to 4 for the venue recommendation task and 3 for the other three tasks. Figure 4.9b shows how training data size affects the quality of the model. We can see that all learning methods benefit from more training data, and this is especially evident when popular entity experts are used. This is due to the fact that they have a large number of parameters to estimate, and we need at least 1000 training queries to prevent overfitting.

4.7.2 Examples of Important Path Features

In the TREC-CHEM Prior Art Search Task [57], people found that instead of directly searching for patents with the query words, it is much more effective to first find patents with similar topics, then aggregate these patents' citations. The relation path of this strategy can be expressed as "*query word* $\xrightarrow{ContainedBy}$ *patent* $\xrightarrow{Cite}$ *patent*". In our experiment, PRA not only successfully identifies this path as an important feature in scientific literature domain (path #2 in Table 4.3), but also finds several other useful paths.

Table 4.3 shows a subset of features for a PRA+qip+pop model trained for the reference recommendation task on the yeast data. Feature #1-#8 are regular path features. Among them, feature #6 resembles what most ad-hoc retrieval systems would do to find relevant papers: finding papers with many words overlapping with the query. However, we can see that this feature is not considered the most important by the model. Instead, the model favors the papers that are well cited by on-topic papers (#2), and the papers cited together with the on-topic papers (#1). Papers cited during the past two years (#7,#8) are also favored. In contrary, general papers published during the past two years (#12,#13) are disfavored.

Table 4.4: Subset of features from a PRA+qip+pop model trained for venue recommendation task on fly data.

ID	θ_π	Feature	Comments
1	26.9	word $\xrightarrow{\text{HasTitle}^{-1}}$ paper $\xrightarrow{\text{In}}$ journal	Journals which have many papers matching the query words
2	4.5	word $\xrightarrow{\text{HasTitle}^{-1}}$ paper $\xrightarrow{\text{FirstAuthor}}$ author $\xrightarrow{\text{FirstAuthor}^{-1}}$ paper $\xrightarrow{\text{In}}$ journal	First find papers matching the query words. Then expand to papers having the same first author.
3	2.8	word $\xrightarrow{\text{HasTitle}^{-1}}$ paper $\xrightarrow{\text{AnyAuthor}}$ author $\xrightarrow{\text{AnyAuthor}^{-1}}$ paper $\xrightarrow{\text{In}}$ journal	First find papers matching the query words. Then expand to papers having the same authors.
4	1.1	gene $\xrightarrow{\text{GeneticallyRelated}}$ gene $\xrightarrow{\text{HasGene}^{-1}}$ paper $\xrightarrow{\text{In}}$ journal	Journals which have papers mentioning genes related to the query genes
5	0.9	gene $\xrightarrow{\text{HasGene}^{-1}}$ paper $\xrightarrow{\text{In}}$ journal	Journals which have many papers matching the query words
6	0.6	e* $\xrightarrow{\text{AnyPaper}}$ paper $\xrightarrow{\text{Cite}}$ paper $\xrightarrow{\text{In}}$ journal	Journals which have many well cited papers
7	3.5	virus > J_Virol	*Journal of Virology* if paper has "virus" in title
8	2.7	deficiency > Am_J_Hum_Genet	*The American Journal of Human Genetics* if paper title has "deficiency"
9	2.0	drosophila > Dev_Biol	*Developmental Biology* if paper title has "drosophila"
10	-2.6	> J_Virol	Negative bias to *Journal of Virology*
11	-3.2	drosophila > J_Bacteriol	*Journal of Bacteriology* if paper title has "drosophila"
12	-3.5	drosophila > Am_J_Hum_Genet	*The American Journal of Human Genetics* if paper title has "drosophila"
13	-3.6	drosophila > J_Med_Genet	*Journal of Medical Genetics* if paper title has "drosophila"
14	-35.8	e* $\xrightarrow{\text{AnyYear}}$ year $\xrightarrow{\text{In}^{-1}}$ paper $\xrightarrow{\text{In}}$ journal	Journals with old papers—since there are fewer papers in the past years, they are getting higher weights

Features starting with e^* are query-independent path features. We can see that well cited papers are generally favored (#9). Since the number of papers published is increasing every year, feature #14 actually disfavors old papers.

Features of the form "$> XXX$" are popular entity biases on specific entities. Features of the form "$XXX > XXX$" are conditional popular entity biases that associate a query entity with a target entity. We can see that papers about specific genes (e.g. CAL4, CYC1) often cite specific early works (#10,#11).

Table 4.4 shows a subset of features for a PRA+qip+pop model trained for the venue recommendation task on the fly data. We can see that different journals have different preferred topics (#7-#9), and some journals are less likely to accept drosophila related papers (#11-#13). Although journals of old papers are disfavored (#14), journals of popular papers are favored (#6). Interestingly, journals with many on-topic first authors (#2) are more favored than those with just any on-topic authors (#3).

4.7.3 Main Results

Table 4.5 compares the effectiveness of different ranking algorithms on all four tasks and two corpora. We can see that PRA performs significantly better than RWR under most tasks. The query-independent path experts (PRA+qip) manage to improve over basic PRA model in all tasks, and especially in reference recommendation and expert finding tasks. The popular entity experts (PRA+pop) also manage to improve over the basic PRA model in all tasks, and the difference is statistically significant on the yeast tasks.

4.8 Summary

We use a graph representation for publication databases with rich metadata. With this representation, PRA models are trained to discover effective recommendation strategies represented as edge paths on the graph. Experiments on citation-based and history-based

Table 4.5: Comparing baseline RWR with PRA and its two extensions: query-independent path experts (+qip) and popular entity experts (+pop) by MAP. Except these†, all improvements over trained RWR are statistically significant at $p<0.05$ by paired t-test.

Corpus/Task	**RWR**		**PRA**			
Corpus=yeast	**untrained**	**trained**	**trained**	**+qip**	**+pop**	**+qip+pop**
Venue	40.4	44.2	45.7	46.4	48.7	49.3
Reference	11.8	16.0	16.9	18.3	19.1	19.8
Expert	9.9	11.1	11.9	12.4	12.5	12.9
Gene	14.4	14.4	14.9	15.1	15.1	15.3
Corpus=fly	**untrained**	**trained**	**trained**	**+qip**	**+pop**	**+qip+pop**
Venue	45.4	48.3	50.4	51.1	50.7	51.7
Reference	18.8	20.5	20.8†	21.0	21.6	21.7
Expert	5.6	7.2	7.6†	8.3	7.9	8.5
Gene	18.7	19.2	20.7	21.1	21.1	21.0

reading recommendation tasks show that by leveraging rich context information the PCRW-based approach outperforms random walk with restart based approaches as well as traditional content-based and collaborative filtering approaches. Experiments on eight paper completion tasks in two subdomains of biology show that the new learning method significantly outperforms the RWR model (both trained and untrained). We also extend the method to support two additional types of experts to model intrinsic properties of entities: *query-independent experts*, which generalize the PageRank measure, and *popular entity experts* which allow rankings to be adjusted for particular entities that are especially important.

Chapter 5

Efficient Random Walk

In previous chapters we see that PRA which is based on Path-Constrained Random Walks (PCRW), significantly outperforms unsupervised random walk based queries, and models with learned edge weights. Unfortunately, PCRW query systems are expensive to evaluate. In this chapter we evaluate the use of approximations to the computation of the PCRW distributions, including fingerprinting, particle filtering, and truncation strategies [52]. We compare these strategies using recommendation and retrieval tasks, for which large number of training and testing queries can be generated. Our experiments show speedups of factors of 100 with little loss in accuracy. We also discuss using low-variance sampling to further improve the random walk qualities.

5.1 Motivation

As discussed in Chapter 2 PCRW query systems are expensive to evaluate—on a well connected graph, the number of nodes with non-zero probability generally grows exponentially with respect to path lengths. However, one important fact about random walks is that, in general, we expect that the random walk will lead to very uneven

distributions over all the entities: high probability on a few entities, usually entities of high in-degrees, and low probability on the remainder. For example, this kind of uneven distribution (called power law distribution) has been observed [67] on PageRank scores of web pages; PageRank is also a type of random walk model. As a consequence, a few nodes have a large influence on the retrieval result, and most nodes have very small influence, and it is plausibly acceptable to ignore, or approximate, the weight of most nodes. In past work, a sampling based random walk strategy has been shown to give inaccurate estimations for low ranked nodes; in spite of this, however, Fogaras et al. [32] showed that using a Monte Carlo algorithm and a small number of trials is sufficient to distinguish between the high, medium and low ranked nodes accurately in Personalized PageRank scores. As well, Chakrabarti [14] used a dynamic pruning strategy for the calculation of Personalized Pageranks, in which weights smaller than a threshold are pruned, and showed that this operation has minimal effect on accuracy. Therefore, we can expect that keeping the distribution reached by random walks sparse may significantly reduce the amount of time and memory spent on query execution.

In order to investigate the trade-off between inference efficiency and retrieval quality for supervised learning of PCRW models, we compare the query execution speedups by different strategies that help maintain sparsity of random walk, including fingerprinting, particle filtering, and truncation strategies.

5.2 Sparse Random Walks

We describe four strategies approximate the exact random walk distribution defined by equation (2.2.1), but with different ways of generating or sparsifying $P(s \to t; \pi)$ at each step of random walk. Each of these strategies will be used in both parameter estimation at training time and query execution at test time.

From here on, we assume that $\pi = \langle r_1, r_2, ..., r_m \rangle$ has m steps. The prefixes of π are

$\pi_0 = \langle\rangle$, $\pi_1 = \langle r_1 \rangle$, $\pi_2 = \langle r_1, r_2 \rangle$, etc. We denote the approximate distributions at each step of random walk as $g_i(e) \approx P(s \rightarrow e; \pi_i)$.

5.2.1 The Fingerprinting Strategy

Fogaras et al. [32] suggested a Monte Carlo algorithm to approximate the distributions of personalized PageRank, where K independent random walks are simulated starting from the query node. The probability of a node u is approximated by the normalized count of number of times it is visited by the random walkers, and the amount of computation can be easily controlled by varying K. The authors showed that using only a relatively small number of random walkers is sufficient to distinguish between the high, medium and low ranked nodes in the fully computed Personalized PageRank scores. Although the ordering of the low ranked nodes are usually not as accurate using sampling, it is most often the high ranked nodes that determine the quality of retrieval.

In this study, we test the effectiveness of this sampling strategy in the context of PCRW. The distribution $h_i(e)$ at the i-th step can be approximated by the normalized count of the number of walkers visiting a node e after following label sequence π_i starting from node s

$$h_i(e) = \frac{\text{\#walkers visiting } e \text{ at the i-th step}}{\text{\#walkers}}.$$

5.2.2 Weighted Particle Filtering

One possible downside for the fingerprinting strategy is the waste of computation when the number of walkers is much larger than the number of links. For example, if we start with 30k walkers from a node which only has three outlinks, the fingerprinting strategy will draw 30k random numbers, and assign each of the walkers to follow a specific outlink. However, with knowledge of probabilities, we know that it is expected to have around 10k walkers following each of the outlinks.

Here we describe a *weighted particle filtering* procedure (Algorithm 1), which is an combination of exact random walk and sampling. Conceptually, we can treat the initial 30k walkers as a single particle (of size 30k), and at the first step of random walk it splits into three equal-sized particles, each of size 10k and following a different link. If we let the particles split to arbitrarily small sizes, then we just get the exact probability distribution defined by equation (2.2.1)(with proper normalization). In order to keep the distribution sparse and random walk fast, we set a threshold ε_{min} on the minimum size of the particles. If a potential split breaks a particle to particles with sizes smaller than the threshold, we switch from exact calculation to a sampling strategy. We let the particle split to a smaller number of child particles, each having the same size as the threshold ε_{min}. Each of these child particles randomly picks one of the outlinks to follow.

5.2.3 Truncation Strategies

As we explained before, random walks usually have an uneven distribution over all the entities: high probability on a few important entities, and low probability on many noisy entities. Therefore, we hypothesize that zeroing the weights on the low-weight entities will not significantly affect the random walk's ability to identify important entities, but may significantly reduce the amount of time and memory spent on random walk. Recently, Chakrabarti [14] applied a dynamic pruning strategy to the calculation of Personalized Pageranks, for which elements in the fingerprint vectors smaller than a threshold are pruned. They discover that this operation has a dramatic effect on keeping the fingerprint vectors sparse, while having a minimal effect on accuracy.

Here we apply this strategy in the context of PCRW. At each step of the random walk, we add a truncation step to the distribution estimated by equation (2.2.1):

$$h_{i+1}(e) = max(0, h_{i+1}(e) - \varepsilon),$$

where ε is a parameter to control the harshness of truncation. This procedure also has the

Algorithm 1 Weighted Particle Filtering

Input: distribution $h_i(e)$, relation r, threshold ε_{min}
Output: $h_{i+1}(e)$
Set $h_{i+1}(e) = 0$
for each e with $h_i(e) \neq 0$ **do**
  $size_{new} = h_i(e)/|r(e)|$
  if $size_{new} > \varepsilon_{min}$ **then**
    for each $e' \in r(e)$ **do**
      $h_{i+1}(e') += size_{new}$
    end for
  else
    for k=1..$floor(h_i(e)/\varepsilon_{min})$ **do**
      randomly pick $e' \in r(e)$
      $h_{i+1}(e') += \varepsilon_{min}$
    end for
  end if
end for

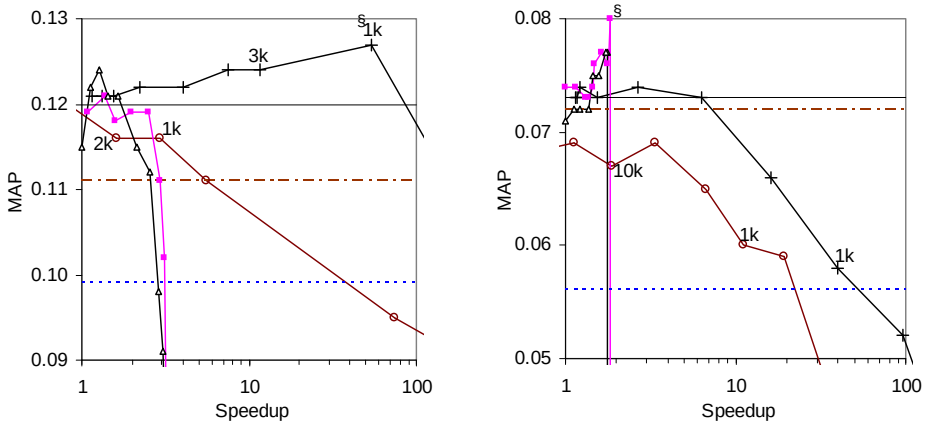
effect of penalizing longer paths. Since longer paths are generally reduced more harshly by this truncation procedure, their weights need to be larger than the short paths in order to achieve the same effect in the ranking function. We call this approach *fixed truncation*.

One possible disadvantage of fixed truncation is that the truncation parameter ε is not directly related to the sparsities of probability distributions. Therefore, we design an adaptive truncation strategy called *beam truncation*, which explicitly constrains the random walk to the desired sparseness. The truncation step is defined as

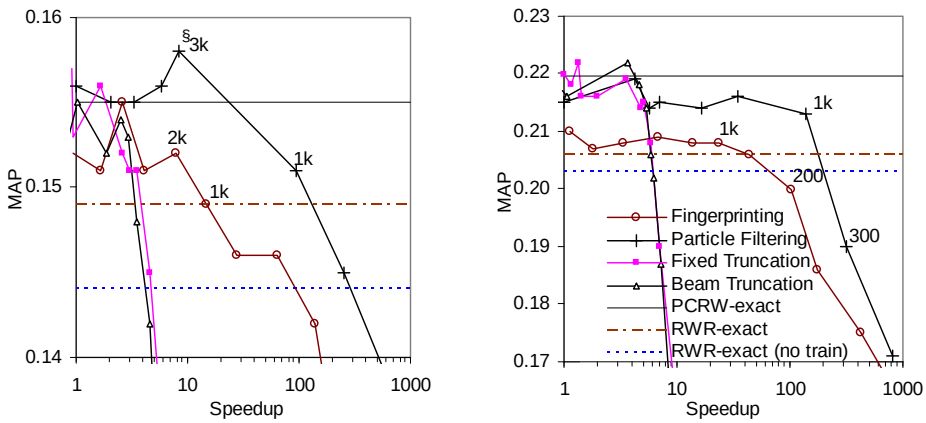
$$h_{i+1}(e) = max(0, h_{i+1}(e) - \varepsilon_W(h_{i+1})),$$

where $\varepsilon_W(h_{i+1})$ is the W-th highest probability in distribution h_{i+1}. Below W is called the width of the beam.

Expert Finding: (yeast) $T_0 = 0.17s, \ell = 3$, (fly) $T_0 = 0.15s, \ell = 3$



Gene Recommendation: (yeast) $T_0 = 1.6s, \ell = 4$, (fly) $T_0 = 1.8s, \ell = 4$



Reference Recommendation: (yeast) $T_0 = 2.7s, \ell = 3$, (fly) $T_0 = 0.9s, \ell = 3$

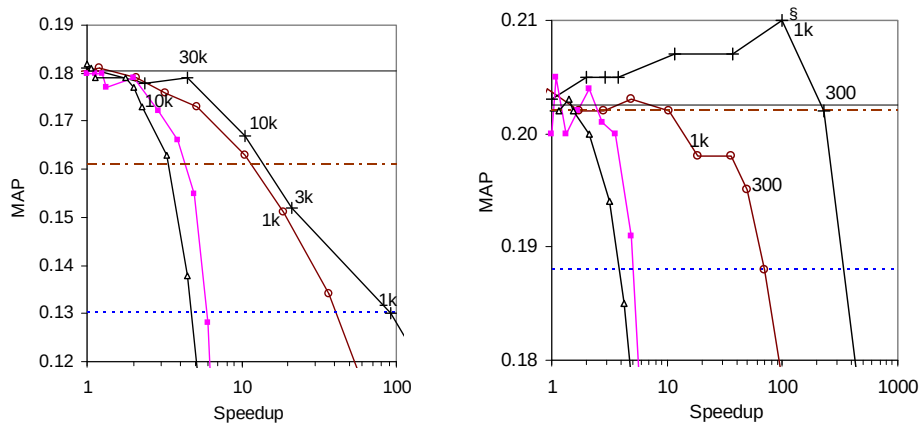


Figure 5.1: Speedup (w.r.t. exact PCRW) *vs.* MAP on yeast (left) and fly (right) data. T_0 is the average query execution time using exact PCRW. ℓ is the maximum length of PCRW. Points are marked with the number of random walkers (particles). Some points[§] are significantly better than exact PCRW ($p < 10^{-5}$ by paired t-test).

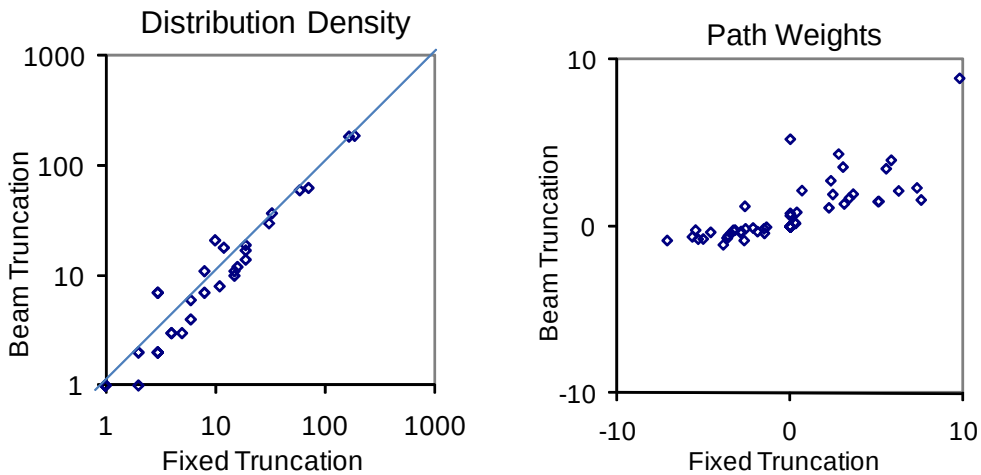
5.3 Case Study: Paper Completion Tasks

5.3.1 PRA with Approximate Random Walks

Here we compare different sparsity strategies by their retrieval qualities on the paper completion tasks defined in Chapter 4. The baseline for these strategies is the exact calculation of random walk distributions. In order to investigate the trade-off between the speedup of query execution and retrieval quality, we vary, for each sparsity method, its sparsity parameter— from the most inefficient to the most efficient setting— and see how the retrieval speed and quality are affected. We provide three baselines as references to the quality of retrieval: exact but unsupervised RWR (uniform weight 1.0), exact RWR, and exact PCRW. For supervised models we fix the regularization parameter λ_2 to 0.001. Note that, in our experiment, train and test of PRA always use the same random walk method.

Figure 5.1 compares the speedup of query execution verses MAP for different sparsity strategies. From top to bottm we order the tasks in increasing order of random walk complexity. The average single query execution times range from 0.15 seconds to 2.7 seconds when using exact calculation of PCRW. We can see that all four strategies manage to speedup query execution to a certain extent without significantly sacrificing the quality of retrieval. The strategies are relatively more effective on complex tasks like gene recommendation and reference recommendation.

The two truncation strategies (fixed and beam truncation) have a relatively limited ability to speedup query execution (ranging from 2- to 10-fold). This is because, at each step of random walk, they need to calculate the full distribution before truncation. When the truncation is harsh, on the other hand, it is likely to produce empty distributions, which are useless for retrieval. Fixed truncation generally has slightly better retrieval quality than beam truncation. After close inspection, we found that although fixed and beam truncation can produce random walk distributions with the same sparsity when their parameters are properly set, fixed truncation has the extra effect of demoting longer paths as discussed



(a) (b)

Figure 5.2: Comparing sparsity and model weights of fixed and beam truncations for the reference recommendation task on Yeast data. (a) Comparing the random walk density (by the number of nodes with non-zero weights averaged over all training queries). Each dot is a particular path type. (b) Comparing the learned weight for each path type.

above. Since random walks of longer paths are reduced more harshly by this truncation procedure, their weights need to be larger than the short paths in order to achieve the same effect in the ranking function. The beam truncation on the other hand might not truncate the distribution at all if the number of non-zero nodes is smaller than the beam size, which is very likely to happen in the first few steps of random walk.

This regularization effect to long paths is evident in Figure 5.2, where we compare sparsity and learned weights for fixed and beam truncation. Parameters are set so both strategies give a 2.7-fold speedup over the exact random walk baseline. However, fixed truncation has statistically significantly better performance (MRR=0.407, MAP=0.205) than beam truncation (MRR=0.398, MAP=0.200). We can see that although the sparsities of these two strategies correlates with each other very well, the learnt weights of fixed truncations are generally much smaller than that of beam truncation. This is because the absolute values of the probabilities produced by fixed truncation are smaller than that of beam truncation, so the path weights of fixed truncation need to be larger than those of

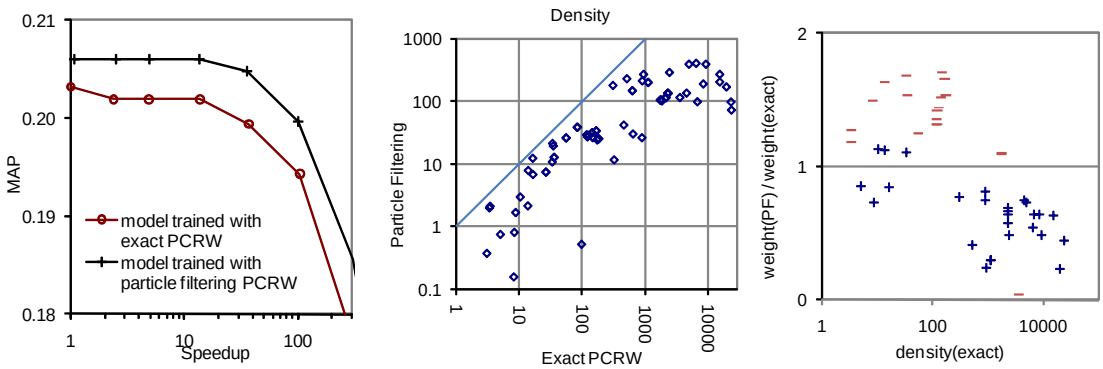
beam truncation in order to achieve the same effect to ranking. Therefore, fixed truncation has the effect of putting heavier regularization on the weights of longer paths.

The two sampling based strategies (fingerprinting and particle filtering) have relatively larger speedups (ranging from 10- to 100-fold on various tasks). We mark, in Figure 5.1, the number of random walkers and the maximum number of particles (estimated by $1/\varepsilon_{min}$) at the points where retrieval quality starts to drop quickly. We can see that, compared to fingerprinting, particle filtering is almost always 2- to 4-fold faster. This is what we have expected, since particle filtering can potentially represent a large number of walkers with a small number of particles during the first few steps of random walk. Overall, we can see that around 1k to 10k particles (or random walkers) are enough for producing good retrieval quality.

Furthermore, for the same speedup, particle filtering almost always produces better retrieval quality than fingerprinting. This is a result of the fact that although both strategies rely on sampling to estimate the exact distribution, particle filtering will have lower variance than fingerprinting. For example, imagine a node with only two outlinks, and let the two strategies each have two particles (minimum particle size 0.5) and two walkers respectively. Fingerprinting has a high chance (0.5) to put all probability mass on only one of the outlinks. Particle filtering, on the other hand, will always assign equal probability 0.5 to both of the links.

5.3.2 Why Approximate Inference Can Improve Retrieval Accuracy

More interestingly, in many cases applying the sparsity strategies not only speeds up the query execution but also produces better retrieval quality (in four out of six tasks). It is not immediately evident whether this is because that the sparsity strategies produce distributions that are concentrated on important nodes or because that they produce models with better weighting of the relation paths. In this section, we argue that particle filtering helps because it imposes a penalty on paths with dense distributions, which is often helpful.



(a) (b) (c)

Figure 5.3: Comparing exact PCRW and particle filtering ($\varepsilon_{min} = 0.001$) for the reference recommendation task on Yeast data. (a) Retrieval qualities with fixed model but various particle filtering settings at query time. The differences of the two models at the same speed up levels are statistically significant at $p < 10^{-5}$ using paired t-test. (b) For each relation path the number of nodes with non-zero probability (averaged over all training queries). (c) For each relation path the ratio of its weight in the particle filtering model *vs.* its weight in the exact PCRW model. The '$+$' and '$-$' signs correspond to paths with positive and negative weights in the model trained with exact PCRW.

In Figure 5.3a we fix the retrieval model to either the one trained with exact PCRW or the one with particle filtering. Then we apply particle filtering during test time and vary ε_{min}. We can see that when the model is fixed, particle filtering does not improve retrieval quality. However, the model trained with particle filtering still performs better than the one trained with exact PCRW. This result indicates that training with particle filtering PCRW can produce better models than exact PCRW.

Figure 5.3b shows the density of each relation path (i.e., the number of nodes with non-zero probability). We can see that for those paths with dense distribution (density $>$ 1000) under exact PCRW, particle filtering effectively reduces densities to a few hundred. Figure 5.3c further shows that, in the model trained with exact PCRW, most of the high density paths have positive weights, and their weights are significantly dropped in the corresponding model trained with particle filtering. Therefore, we can see that the sparsity strategies have the effect of putting a higher regularization on the paths with dense distributions, which turns out to be useful in producing good retrieval models.

5.4 Low-Variance Sampling

We have shown that sampling techniques like finger printing and particle filtering can significantly speedup random walk without sacrificing retrieval quality. However, the sampling procedures can induce a loss of diversity in the particle population. For example, consider a node in the graph with just two out links with equal weights, and suppose we are required to generate two walkers starting from this node. A disappointing result is that with 50 percent chance both walkers will follow the same branch, and leave the other branch with no probability mass.

To overcome this problem, we apply a technique called Low-Variance Sampling (LVS) [97], which is commonly used in robotics to improve the quality of sampling. Instead of generating independent samples from a distribution, LVS uses a single random number to generate all samples, which are evenly distributed across the whole distribution. Note that given a distribution $P(x)$, any number r in $[0, 1]$ points to exactly one x value, namely $x = \arg\min_j \sum_{m=1..j} P(m) \leq r$. Suppose we want to generate M samples from $P(x)$. LVS first generates a random number r in the interval $[0, M^{-1}]$. Then LVS repeatedly adds the fixed amount M^{-1} to r and chooses x values corresponding to the resulting numbers.

We also investigate the effect of low-variance sampling on the quality of prediction. Figure 5.4 compares independent and low variance sampling when applied to particle filtering [51]. The horizontal axis corresponds to the speedup of random walk compared with exact inference, and the vertical axis measures the quality of prediction by MRR with three fold cross validation. Low-variance sampling can improve prediction especially at high speedup range, where there are smaller number of walkers, and the random walk is more susceptible to variance. The numbers on the curves indicate the number of particles (or walkers). When using a large number of particles, the particle filtering methods converge to the exact inference. We noticed a similar improvement on retrieval tasks (Chapter 4), and conjectured that it is because the sampling inference imposes a regularization penalty on longer relation paths [51].

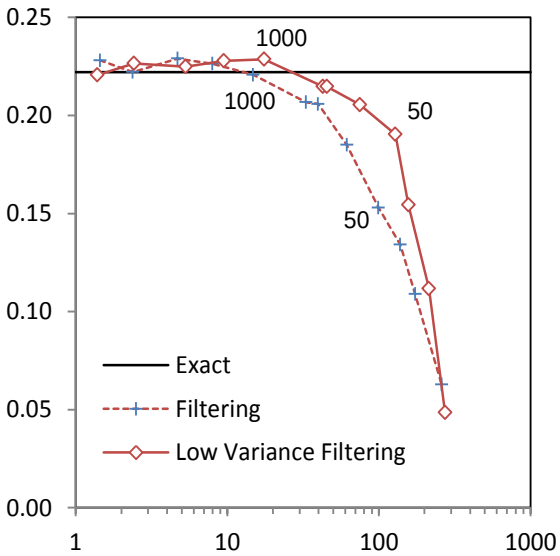


Figure 5.4: Comparing inference speed and quality over 8 scientific recommendation tasks (Chapter 4). The speedup is relative to exact inference

5.5 Summary

In this study, we evaluate the use of approximations to the computation of the path constrained random-walk distributions. We compared fingerprinting, particle filtering, and truncation strategies. In experiments on several recommendation and retrieval problems using two large scientific publication corpora, we show speedups of factors of 100 with little loss in retrieval accuracy. More interestingly, we found that in many cases moderately applying the sparsity strategies not only speeds up the query execution but also produces better retrieval quality. We also explore using low-variance sampling to further improve the random walk qualities.

Chapter 6

Case Study: Reading The Web with Learned Syntactic-Semantic Patterns

In Chapter 3 we discussed learning to extend a KB of facts extracted from the Web. Here we study how to extend a large knowledge base (Freebase) by using two source of information: other relations in FreeBase, and information from a large parsed text corpus.

Previous studies on extracting relational knowledge from text show the potential of syntactic patterns for extraction [60, 94, 96], but they do not exploit background knowledge of other relations in the knowledge base. We describe a distributed, Web-scale implementation of a path-constrained random walk model that learns syntactic-semantic inference rules for binary relations from a graph representation of the parsed text and the knowledge base. Experiments show significant accuracy improvements in binary relation prediction over methods that consider only text, or only the existing knowledge base [54].

6.1 Motivation

Manually-created knowledge bases (KBs) often lack basic information about some entities and their relationships, either because the information was missing in the initial sources used to create the KB, or because human curators were not confident about the status of some putative fact, and so they excluded it from the KB. For instance, as we will see in more detail later, many person entries in Freebase [11] lack nationality information. To fill those KB gaps, we might use general rules, ideally automatically learned, such as "if *person* was born in *town* and *town* is in *country* then the *person* is a national of the *country*." Of course, rules like this may be defeasible, in this case for example because of naturalization or political changes. Nevertheless, many such imperfect rules can be learned and combined to yield useful KB completions, as demonstrated in particular with PRA (see Chapter 3).

Alternatively, we may attempt to fill KB gaps by applying relation extraction rules to free text. For instance, Snow et al. [94] and Suchanek et al. [96] showed the value of syntactic patterns in extracting specific relations. In those approaches, KB tuples of the relation to be extracted serve as positive training examples for the extraction rule induction algorithm. However, the KB contains much more knowledge about other relations, which could potentially be helpful in improving relation extraction accuracy and coverage, but which is not used in such purely text-based approaches.

In this study, we test the hypothesis that PRA can be used to find useful "syntactic-semantic patterns" – that is, patterns that exploit both semantic and syntactic relationships, thereby using semantic knowledge as background in interpreting syntactic relationships. As shown in Figure 6.1, we extend the KB graph G with nodes and edges from text that has been syntactically analyzed with a dependency parser[1] and where pronouns and other anaphoric referring expressions have been clustered with their

[1] We use Stanford dependencies [22].

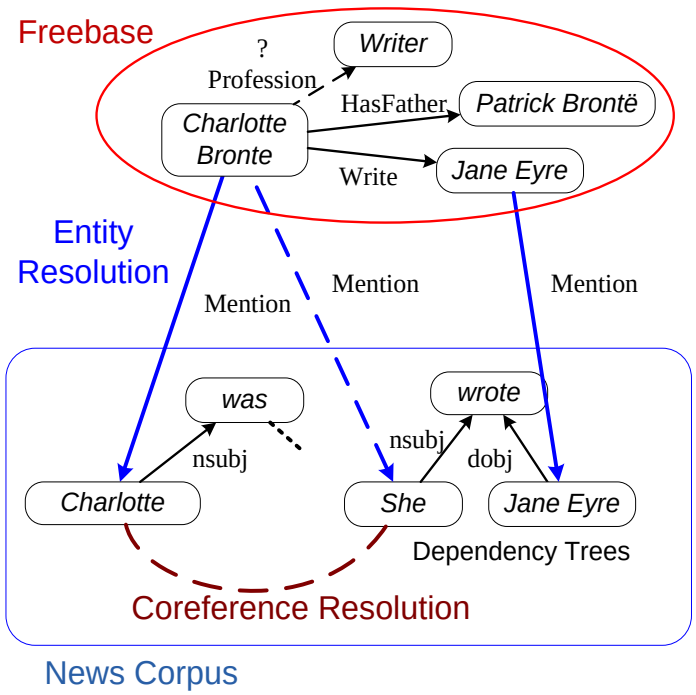


Figure 6.1: Knowledge base and parsed text as a labeled graph. For clarity, some word nodes are omitted. See Table 6.1 for the meaning of dependency edges

antecedents. The text nodes are word/phrase instances, and the edges are syntactic dependencies labeled by the corresponding dependency type. Mentions of entities in the text are linked to KB concepts by mention edges created by an entity resolution process.

Given, for instance, the query $Profession(CharlotteBronte, ?)$, PRA produces a ranked list of answers that may have the relation *Profession* with the query node *CharlotteBronte*. The features used to score answers are the random walk probabilities of reaching a certain profession node from the query node by paths with particular path types. PRA can learn path types that combine background knowledge in the database with syntactic patterns in the text corpus. We now exemplify some path types involving relations described in Table 6.4. Type $\langle M, conj, M^{-1}, Profession \rangle$ is *active* (matches paths) for professions of persons who are mentioned in conjunction with the query person as

Notation	Name	Description
appos	appositional modifier	An appositional modifier of an NP is an NP immediately to the right of it that serves to define or modify that NP. e.g., "Sam, my brother"
conj	conjunct	A conjunct is the relation between two elements connected by as "and", "or", etc. E.g., "Bill is big and honest", "They either ski or snowboard".
dobj	direct object	The direct object of a VP is the noun phrase which is the (accusative) object of the verb. E.g., "She gave me a raise" dobj(gave, raise).
nn	noun compound modifier	A noun compound modifier of an NP is any noun that modifies the head noun. (Note that many dependency parsers have no intelligent noun compound analysis. all nouns modify the rightmost noun of the NP. E.g., "Oil price futures" nn(futures, oil) nn(futures, price)
nsubj	nominal subject	A nominal subject is a noun phrase which is the syntactic subject of a clause. E.g., "Clinton defeated Dole" nsubj(defeated, Clinton)
poss	possession modifier	The possession modifier relation holds between the head of an NP and its possessive determiner. E.g., "Bill's clothes" poss(clothes, Bill)
tmod	temporal modifier	A temporal modifier is a bare noun phrase that modify the meaning of a constituent by specifying a time. E.g. "Last night, I swam in the pool" tmod(swam, night)

Table 6.1: List of dependency edges used in our examples

in "collaboration between McDougall and Simon Philips". For a somewhat subtler example, type $\langle M, TW, CW^{-1}, Profession^{-1}, Profession \rangle$ is active for persons who are mentioned by their titles as in "President Barack Obama". The type subsequence $\langle Profession^{-1}, Profession \rangle$ ensures that only profession concepts are activated. The features generated from these path types combine syntactic dependency relations (*conj*) and textual information relations (*TW* and *CW*) with semantic relations in the KB (*Profession*). Those relations are defined on Table 6.4.

Experiments on three Freebase relations (profession, nationality and parents) show that

exploiting existing background knowledge as path features can significantly improve the quality of extraction compared with using either Freebase or the text corpus alone.

6.2 Related Work

Much of the previous work on automatic relation extraction was based on certain lexico-syntactic patterns. Hearst [40] first noticed that patterns such as "NP and other NP" and "NP such as NP" often imply hyponym relations (NP refers to a noun phrase). However, such approaches are limited by the availability of domain knowledge. Later systems for extracting arbitrary relations from text mostly use shallow surface text patterns [30, 3, 81]. The idea of using sequences of dependency edges as features for relation extraction was explored by Snow et al. [94] and Suchanek et al. [96]. They define features to be shortest paths on dependency trees which connect pairs of NP candidates.

This study is most closely related to work of Mintz et al. [60], who also study the problem of extending Freebase by extraction from parsed text. As in our work, they use a logistic regression model with path features. However, their approach does not exploit existing knowledge in the KB. Furthermore, their path patterns are used as binary-valued features. We show experimentally that fractional-valued features generated by random walks provide much higher accuracy than binary-valued ones for this task.

Culotta et al. [19]'s work is similar to our approach in the sense of relation extraction by discovering relational patterns. However while they focus on identifying relation mentions in text, this work attempts to infer new tuples by gathering path evidence over the whole corpus. In addition, their work involves a few thousand examples, while we aim for Web-scale extraction.

Do and Roth [87] use a KB (YAGO) to aid the generation of features from free text. However their method is designed specifically for extracting hierarchical taxonomic structures, while our algorithm can discover relations for general graph-based KBs.

6.3 Extending PRA

As described in the previous section, the PRA model is trained on positive and negative queries generated from the KB. As Freebase contains millions of concepts and edges, training on all the generated queries is computationally challenging. Further, we extend the Freebase graph with parse paths of mentions of concepts in Freebase in millions of Web pages. Yet another issue is that the training queries generated using Freebase are inherently biased towards the distribution of concepts in Freebase and may not reflect the distribution of mentions of these concepts in text data. As one of the goals of our approach is to learn relation instances that are missing in Freebase, training on such a set, biased towards the distribution of concepts in Freebase, may not lead to good performance. In this section we explain how we modified the PRA algorithm to address those issues.

6.3.1 Scaling Up

Most relations in Freebase have a large set of existing triples. For example, for the *profession* relation, there are around 2 million persons in Freebase, and about 0.3 million of them have known professions. This results in more than 0.3 million training queries (persons), each with one or more positive answers (professions), and millions of negative answers, which make training computationally challenging. Generating all the paths for millions of queries over a graph with millions of concepts and edges further complicates the computational issues. Incorporating the parse path features from the text only exacerbates the matter. Finally once we have trained a PRA model for a given relation, say *profession*, we would like to infer the professions for all the 1.7 million persons whose professions are not known to Freebase (and possibly predict changes to the profession information of the 0.3 million people whose professions were given).

We use distributed computing to deal with the large number of training and prediction queries over a large graph. A key observation is that the different stages of the PRA

algorithm are based on independent computations involving individual queries. Therefore, we can use the MapReduce framework to distribute the computation [26]. For path discovery, we modify path finding approach to decouple the queries: instead of using one depth-first search that involves all the queries, we first find all paths up to certain length for each query node in the map stage, and then collect statistics for each path from all the query nodes in the reduce stage. We used a 500-machine, 8GB/machine cluster for these computations.

Another challenge associated with applying PRA to a graph constructed using large amounts of text is that we cannot load the entire graph on a single machine. To circumvent this problem, we first index all parsed sentences by the concepts that they mention. Therefore, to perform a random walk for a query concept s, we only load the sentences which mention s.

6.3.2 Sampling Training Data

Using the r-edges in the KB as positive examples distorts the training set. For example, for the *profession* relation, there are 0.3 million persons for whom Freebase has profession information, and amongst these 0.24 million are either politicians or actors. This may not reflect the distribution of professions of persons mentioned in Web data. Using all of these as training queries will most certainly bias the trained model towards these professions, as PRA is trained discriminatively. In other words, training directly with this data would lead to a model that is more likely to predict professions that are popular in Freebase. To avoid this distortion, we use stratified sampling. For each relation r and concept $t \in C$, we count the number of r edges pointing to t as $N_{r,t} = |\{(s, r, t) \in T\}|$. Given a training query (s, r, t) we sample it according to

$$P_{r,t} = \min\left(1, \frac{\sqrt{m + N_{r,t}}}{N_{r,t}}\right)$$

We fix $m = 100$ in our experiments. If we take the profession relation as an example, the above implies that for popular professions, we only sample about $\sqrt{N_{r,t}}$ out of the $N_{r,t}$ possible queries that end in t, whereas for the less popular professions we would use all the training queries.

6.3.3 Text Graph Construction

As we are processing Web text data (see following section for more detail), the number of mentions of a concept follows a somewhat heavy-tailed distribution: there are a small number of very popular concepts (head concepts) and a large number of not so popular concepts (tail concepts). For instance the concept *BarackObama* is mentioned about 8.9 million times in our text corpus. To prevent the text graph from being dominated by the head concepts, for each sentence that mentions concept $c \in C$, we accept it as part of the text graph with probability:

$$P_c = \min\left(1, \frac{\sqrt{k + S_c}}{S_c}\right)$$

where S_c is the number of sentences in which c is mentioned in the whole corpus. In our experiments we use $k = 10^5$. This means that if $S_c \gg k$, then we only sample about $\sqrt{S_c}$ of the sentences that contain a mention of the concept, while if $S_c \ll k$, then all mentions of that concept will likely be included.

6.4 Experiment Setup

We use Freebase [11] as our knowledge base for these experiments. Freebase data is harvested from many sources, including Wikipedia, AMG, and IMDB.[2] As of this writing, it contains more than 21 million concepts and 70 million labeled edges. For a large

[2] `www.wikipedia.org`, `www.allmusic.com`, `www.imdb.com`.

majority of concepts that appear both in Freebase and Wikipedia, Freebase maintains a link to the Wikipedia page of that concept.

We also collect a large Web corpus and identify 60 million pages that mention concepts relevant to this study. The free text on those pages are POS-tagged and dependency parsed with an accuracy comparable to that of the current Stanford dependency parser [47]. The parser produces a dependency tree for each sentence with each edge labeled with a standard dependency tag (see Figure 6.1).

In each of the parsed documents, we use POS tags and dependency edges to identify noun phrases (NPs). We then use a within-document coreference resolver comparable to that of [36] to group referring NPs into co-referring clusters. For each cluster that contains a proper-name mention, we find the Freebase concept or concepts, if any, with a name or alias that matches the mention. If a cluster has multiple possible matching Freebase concepts, we choose a single sense based on the following simple model. For each Freebase concept $c \in C$, we compute $N(c, m)$, the number of times the concept c is referred by mention m by using both the alias information in Freebase and the anchors of the corresponding Wikipedia page for that concept. Based on $N(c, m)$ we can calculate the empirical probability $p(c|m) = N(c, m)/\sum_{c'} N(c', m)$. If u is a cluster with mention set $M(u)$ in the document, and $C(m)$ the set of concepts in KB with name or alias m, we assign u to concept $c^* = \underset{c \in C(m), m \in M(u)}{\operatorname{argmax}} p(c|m)$, provided that there exists at least one $c \in C(m)$ and $m \in M(u)$ such that $p(c|m) > 0$. Note that here $M(c)$ only consists of the proper-name mentions in cluster c.

We use three relations *profession*, *nationality* and *parents* for our experiments. For each relation, we select its current set of triples in Freebase, and apply the stratified sampling (Section 6.3.2) to each of the three triple sets. The resulting triple sets are then randomly split into training (60% of the triples) and test (the remaining triples). However, the *parents* relation yields 350k triples after stratified sampling, so to reduce experimental effort we further randomly sub-sample 10% of that as input to the train-test split. Table 6.2

Table 6.2: Size of training and test sets for each relation.

Task	Training Set	Test Set
Profession	22,829	15,219
Nationality	14,431	9,620
Parents	21,232	14,155

shows the sizes of the training and test sets for each relation.

To encourage PRA to find paths involving the text corpus, we do not count relation M (which connects concepts to their mentions) or M^{-1} when calculating path lengths. We use L_1/L_2-regularized logistic regression to learn feature weights. The PRA hyperparameters (α and K as defined in Chapter 2) and regularizer hyperparameters are tuned by threefold cross validation (CV) on the training set. We average the models across all the folds and choose the model that gives the best performance on the training set for each relation.

We report results of two evaluations. First, we evaluate the performance of the PRA algorithm when trained on a subset of existing Freebase facts and tested on the rest. Second, we had human annotators verify facts proposed by PRA that are not in Freebase.

6.5 Evaluation with Closed-World Assumption

Previous work in relation extraction from parsed text [60] has mostly used binary features to indicate whether a pattern is present in the sentences where two concepts are mentioned.

To investigate the benefit of having fractional valued features generated by random walks (as in PRA), we also evaluate a *binarized PRA* approach, for which we use the same syntactic-semantic pattern features as PRA does, but binarize the feature values from PRA: if the original fractional feature value was zero, the feature value is set to zero (equivalent to not having the feature in that example), otherwise it is set to 1.

Table 6.3 shows a comparison of the results obtained using the PRA algorithm trained

Table 6.3: Mean Reciprocal Rank (MRR) for different approaches under closed-world assumption. Here KB, Text and KB+Text columns represent results obtained by training a PRA model with only the KB, only text, and both KB and text. KB+Text[b] is the binarized PRA approach trained on both KB and text. The best performing system (results shown in bold font) is significant at 0.0001 level over its nearest competitor according to a difference of proportions significance test.

Task	KB	Text	KB+Text	KB+Text[b]
Profession	0.532	0.516	**0.583**	0.453
Nationality	0.734	0.729	**0.812**	0.693
Parents	0.329	0.332	**0.392**	0.319

using only Freebase (**KB**), using only the text corpus graph (**Text**), trained with both Freebase and the text corpus (**KB+Text**) and the binarized PRA algorithm using both Freebase and the text corpus (**KB+Text[b]**). Comparing the results of first three columns we see that combining Freebase and text achieves significantly better results than using either Freebase or text alone. Further comparing the results of last two columns we also observe a significant drop in MRR for the binarized version of PRA. This clearly shows the importance of using the random walk probabilities. It can also be seen that the MRR for the parents relation is lower than those for other relations. This is mainly because there are larger number of potential answers for each query node of *Parent* relation than for each query node of the other two relations – all persons in Freebase versus all professions or nationalities. Finally, it is important to point out that our evaluations are actually *lower bounds* of actual performance, because, for instance, a person might have a profession besides the ones in Freebase and in such cases, this evaluation does not give any credit for predicting those professions — they are treated as errors. We try to address this issue with the manual evaluations in the next section.

Table 6.3 only reports results for the maximum path length $\ell = 4$ case. We found that shorter maximum path lengths give worse results: for instance, with $\ell = 3$ for the profession relation, MRR drops to 0.542, from 0.583 for $\ell = 4$ when using both Freebase and text. This difference is significant at the 0.0001 level according to a difference of proportions test. Further we find that using longer path length takes much longer time to

train and test, but does not lead to statistically significant improvements over the $\ell = 4$ case. For example, for profession, $\ell = 5$ gives a MRR of 0.589.

Table 6.4 shows the top weighted features that involve text edges for PRA models trained on both Freebase and the text corpus. To make them easier to understand, we group them based on their functionality. For the profession and nationality tasks, the conjunction dependency relation (in group 1,4) plays an important role: these features first find persons mentioned in conjunction with the query person, and then find their professions or nationalities. The features in group 2 capture the fact that sometimes people are mentioned by their professions. The subpath $\langle \textit{Profession}^{-1}, \textit{Profession} \rangle$ ensures that only profession related concepts are activated. Features in group 3 first find persons with similar names or mentioned in similar ways to the query person, and then aggregate the professions of their children, parents, or advisors. Features in group 6 can be seen as special versions of feature $\langle \textit{PlaceOfBirth}, \textit{ContainedBy} \rangle$ and $\langle \textit{PlaceOfDeath}, \textit{ContainedBy} \rangle$. The subpaths $\langle M, \textit{poss}, \textit{poss}^{-1}, M^{-1} \rangle$ and $\langle M, \textit{title}, \textit{title}^{-1}, M^{-1} \rangle$ return the random walks back to the query node only if the mentions of the query node have *poss* (stands for *possessive modifier*, e.g. "Bill's clothes") or *title* (stands for *person's title*, e.g. "President Obama") edges in text; otherwise these features are inactive. Therefore, these features are active only for specific subsets of queries. Features in group 8 generally find persons with similar names or mentioned in similar ways to the query person. However, they further expand or restrict this person set in various ways.

Typically, each trained model includes hundreds of paths with non-zero weights, so the bulk of classifications are based on on the combination of a large number of lower-precision high-recall or high-precision lower-recall rules.

Table 6.4: Top weighted path types involving text edges for each task grouped according to functionality. M connects each concept in knowledge base to its mentions in the corpus. TW connects each token in a sentence to the words in the text representation of this token. CW connects each concept in knowledge base to the words in the text representation of this concept. We use lower case names to denote dependency edges, and word capitalized names to denote KB edges. See Table 6.1 for the meaning of dependency edges

	Profession	**Comments**
1	$\langle M, conj, M^{-1}, Profession\rangle$ $\langle M, conj^{-1}, M^{-1}, Profession\rangle$	Professions of persons mentioned in conjunction with the query person: *"McDougall and Simon Phillips collaborated ..."*
2	$\langle M, TW, CW^{-1}, Profession^{-1}, Profession\rangle$	Active if a person is mentioned by his profession: *"The president said ..."*
3	$\langle M, TW, TW^{-1}, M^{-1}, Children, Profession\rangle$ $\langle M, TW, TW^{-1}, M^{-1}, Parents, Profession\rangle$ $\langle M, TW, TW^{-1}, M^{-1}, Advisors, Profession\rangle$	First find persons with similar names or mentioned in similar ways, then aggregate the professions of their children/parents/advisors: starting from the concept *BarackObama*, words such as *"Obama"*, *"leader"*, *"president"*, and *"he"* are reachable through path $\langle M, TW\rangle$
	Nationality	**Comments**
4	$\langle M, conj, TW, CW^{-1}, Nationality\rangle$ $\langle M, conj^{-1}, TW, CW^{-1}, Nationality\rangle$	The nationalities of persons mentioned in conjunction with the query person: *"McDougall and Simon Phillips collaborated ..."*
5	$\langle M, nc^{-1}, TW, CW^{-1}, Nationality\rangle$ $\langle M, tmod^{-1}, TW, CW^{-1}, Nationality\rangle$ $\langle M, nn, TW, CW^{-1}, Nationality\rangle$	The nationalities of persons mentioned close to the query person through other dependency relations.
6	$\langle M, poss, poss^{-1}, M^{-1}, PlaceOfBirth, ContainedBy\rangle$ $\langle M, title, title^{-1}, M^{-1}, PlaceOfDeath, ContainedBy\rangle$	The birth/death places of the query person with restrictions to different syntactic constructions.
	Parents	**Comments**
7	$\langle M, TW, CW^{-1}, Parents\rangle$	The parents of persons mentioned in similar ways: starting from the concept *CharlotteBronte* words such as *"Bronte"*, *"Charlotte"*, *"Patrick"*, and *"she"* are reachable through path $\langle M, TW\rangle$.
8	$\langle M, nsubj, nsubj^{-1}, TW, CW^{-1}\rangle$ $\langle M, nsubj, nsubj^{-1}, M^{-1}, CW, CW^{-1}\rangle$ $\langle M, nc^{-1}, nc, TW, CW^{-1}\rangle$ $\langle M, TW, CW^{-1}\rangle$ $\langle M, TW, TW^{-1}, TW, CW^{-1}\rangle$	Persons mentioned in similar ways to the query person with various restrictions or expansions. $\langle nsubj, nsubj^{-1}\rangle$ and $\langle nc^{-1}, nc\rangle$ require the query to be subject and noun compound respectively. $\langle TW^{-1}, TW\rangle$ expands further by word similarities.

6.6 Manual Evaluation

We performed two sets of manual evaluations. In each case, an annotator is presented with the triples predicted by PRA, and asked if they are correct. The annotator has access to the Freebase and Wikipedia pages for the concepts, and web search engines.

In the first evaluation, we compared the performance of two PRA models, one trained using the stratified sampled queries and another trained using a randomly sampled set of queries for the profession relation. For each model, we randomly sample 100 predictions from the top 1000 predictions (sorted by the scores returned by the model). We found that the PRA model trained with stratified sampled queries has 0.92 precision, while the other model has only 0.84 precision (significant at the 0.02 level). This shows that stratified sampling leads to improved performance.

We also evaluated the new beliefs proposed by the models trained for all the three relations using stratified sampled queries. We estimated precision for the top 100 predictions and randomly sampled 100 predictions each from the top 1,000 and 10,000 predictions. Here we use the PRA model trained using both KB and text. The results of this evaluation are shown in Table 6.5. It can be seen that the PRA model is able to produce very high precision predications even when one considers the top 10,000 predictions.

The model produces many new predictions. For instance, for the profession relation, we are able to predict professions for the around 2 million persons in Freebase. The top 1000 profession facts extracted by our system involve 970 distinct people, the top 10,000 facts involve 8,726 distinct people, and the top 100,000 facts involve 79,885 people.

Table 6.5: Human judgement for predicted new beliefs.

Task	p@100	p@1k	p@10k
Profession	0.97	0.92	0.84
Nationality	0.98	0.97	0.90
Parents	0.86	0.81	0.79

6.7 Summary

We have shown that path constrained random walk models can effectively infer new beliefs from a large scale parsed text corpus with background knowledge. Evaluation by human annotators shows that by combining syntactic patterns in parsed text with semantic patterns in the background knowledge, our model can propose new beliefs with high accuracy. Thus, the proposed random walk model can be an effective way to automate knowledge acquisition from the web.

Chapter 7

More Expressive Features

We have shown that PRA scales to large data sets. However, it has been limited to considering paths with no constants. In this chapter, we show that PRA can effectively model paths that correspond to first order rules with constants. In addition, we extend PRA to consider longer relational paths. We show that both types of paths can be efficiently discovered using a backward random walk techniques.

7.1 Motivation

Despite its efficiency, the expressibility of PRA is somewhat limited, compared with inductive logical programming approaches such as FOIL [79]. In particular, while methods such as FOIL learn first order rules (patterns) with constants, PRA restricts its features to generic edge type sequences. Learning features that include constants is generally desirable; for example, consider the rule

$$\textit{IsA}(t, \textit{SportsTeam}) \wedge \textit{AthleteEmployeedByAgent}(s, t) \rightarrow \textit{AthletePlaysForTeam}(s, t)$$

where *SportsTeam* is a *constant*, corresponding to a fixed graph node.

In this chapter we extend PRA to consider a larger set of possible paths, including paths with constants. To this end, we perform *backward* random walks to identify a set of candidate paths with constants, and estimate their quality at the same time. For instance, consider a given a training example, such as *AthletePlaysForTeam*(*HinesWard*, *Steelers*), stating that for the query node *HinesWard* and the relation *AhtletePlaysForTeam*, the node denoting *Steelers* is a correct answer. We perform a special type of *backward* random walks from the given correct answer *Steelers*. Any node z reachable through a path π indicates a potential useful random walk feature, originating from the constant node z and following the reversed path of π. A graph walk based goodness measure is then used to efficiently select the most promising features to be added to the model. We show that the paths with constants learned provide informative class priors to relational classification tasks. For example, a random walk that starts from the concept *Sports* following the path $\langle \textit{IsA}^{-1}, \textit{TeamPlaysSport}^{-1} \rangle$ provides a distribution over teams playing all types of sports, and thus serves as a useful prior for the task of predicting the *AthletePlaysForTeam* relationship.

In addition to learning paths with constants, we enhance PRA so that it efficiently learns patterns that consist of long predicate chains. While relational patterns often comprise of a short chain of relations between objects (such as the examples given so far), some phenomena may be represented by relatively long patterns. In order to efficiently find patterns containing long predicate chains, we introduce an random walk calculation method inspired by *bi-directional* search. More specifically, to calculate $P(s \rightarrow t; \pi_1\pi_2)$, the probability of reaching t from s following a long path type $\pi = \pi_1\pi_2$, we break the computation into two parts— calculating $P(s \rightarrow z; \pi_1)$ using forward random walks, and calculating $P(z \rightarrow t; \pi_2)$ using backward random walks. For paths of length $2M$, this reduces the time complexity of path finding from $O(|V|^{2M})$ to $O(|V|^{M})$, where $|V|$ is the number of relations types in a domain.

The main contribution of this chapter is to introduce and evaluate backward random

walks as a tool for efficiently learning complex relational features. Experiments on graph-based knowledge base inference and coordinate term extraction from parsed text show that these extensions allow PRA to efficiently explore larger feature spaces, significantly improving model quality over previous results in both domains. In particular, incorporating paths with constants significantly improves model quality across tasks, and the coordinate term extraction task benefits from learning long relational paths. In addition, we demonstrate the scalability of the proposed framework compared with traditional relational learning approaches, such as FOIL.

7.2 Related Work

The paths with constants features learned here are a generalization of the *query-independent experts* introduced in Chapter 4. There, query-independent experts constitute random walkers starting from a special node which has links to all nodes of a certain type. Thus, these experts form a general prior distribution over nodes (entities) of the entity type retrieved. In contrast, the paths with constants here can start from any node in the graph, corresponding to particular concepts that are task-specific (e.g., *sports*). Furthermore, by leveraging labeled samples, a relatively small number of these specialized paths are learned, as opposed to arbitrarily generating a large number of query-independent experts in an unsupervised fashion.

Previously, it has been proposed to pre-compute the graph walk distribution for a subset of 'hub' nodes in an entity-relation graph as means of accelerating the graph walk computation, where hub nodes were selected based on query log statistics [14]. Similarly, graph walks starting at these constant "hub" nodes are efficiently computed once and re-used. However, the set of paths with constants was selected here so as to maximize output nodes' relevancy.

Learning long relational paths is desirable as it allows capturing complex and specific

relationships between entities. In a recent work, the Markov Logic Network framework has been extended to model long paths, showing improvements in prediction performance in some cases [**?**]. To detect long useful paths, they analyze the graph to identify typical path segments (motifs) and perform search within and between these segments to reduce the search space size. In comparison, we show that extending PRA with simple bidirectional random walks, coupled with efficient sampling measures, allows to effectively learn long meaningful paths from the raw graph. Interestingly, we find that long paths can be viewed as an alternative to concatenating multiple path segments in a bootstrapping process [13]. This implies that incorporating long paths in relational inference can boost recall, as will be demonstrated in our experiments.

7.3 Approach

In this section, we define backward random walks (Sec. 7.3.1), which are used to efficiently apply two extensions to PRA—learning patterns with constants (Sec. 7.3.2), and learning patterns with long predicate chains (Sec. 7.3.3).

7.3.1 Backward Random Walks

We next define *backward random walk*, which will be used in evaluating paths with constants, and paths that describe long predicate chains.

Alternatively to Eq. (2.2.1), the probability of reaching a target node t from s, following path π, can be recursively defined in a backward fashion as

$$P(s \to t; \pi) = \sum_{z} P(s \to z; r) P(z \to t; \pi'), \tag{7.3.1}$$

where π' is the path that results from removing the first relation relation r in π. Using the sampling techniques described in [51], we can efficiently calculate $P(s \to t; \pi)$ for a given t and all possible source nodes s.

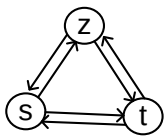


Figure 7.1: Six types of random walks which can generate relational features for link prediction tasks: $P(s \to t; \pi)$, $P(t \to s; \pi)$, $P(z \to s; \pi)$, $P(s \to z; \pi)$, $P(z \to t; \pi)$, $P(t \to z; \pi)$, where s is the query node, t is the target node, and z is a constant node.

As shown in Figure 7.1, there are six types of random walk probabilities that can be used as relational features. The original PRA algorithm only uses features of the form $P(s \to t; \pi)$. In this study we also consider graph walk features in the inverse direction of the form $P(t \to s; \pi)$. Therefore, we define generalized feature quality measures

$$acc(f) = \frac{1}{n} \sum_i \frac{\sum_{t \in G_i} f(s_i, t)}{\sum_t f(s_i, t)}, and \tag{7.3.2}$$

$$hits(f) = \sum_i I(\sum_{t \in G_i} f(s_i, t) > 0), \tag{7.3.3}$$

where $f(s, t)$ is either $P(s \to t; \pi)$ or $P(t \to s; \pi)$. The accuracy measure is in the range $[0, 1]$, as all features are random walk probabilities.

We also define the *support* of a path type π (either forward or backward) on a set of queries, as $sup(\pi)$ the number of queries for which this path type is active. Note that $his(\pi) \leq sup(\pi)$, and it will be useful for pruning the search space of useful paths with constants and long paths.

In the next section, we discuss how to identify useful constant nodes z using backward random walks, and efficiently compute $P(z \to t; \pi)$ and $P(t \to z; \pi)$ features. The number of useful constant nodes z is usually small, and, unlike the query-specific s and t node pairs, the set of constant nodes z can be stored and reused for different queries that are targeted at relation r. Random walk probabilities between s and constant nodes z, $P(z \to s; \pi)$ and $P(s \to z; \pi)$, do not directly affect the ranking of candidate answer

nodes, so we do not use them in this study. It is possible, however, to generate random walk features that combine these probabilities with random walks starting or ending with target t through conjunction.

7.3.2 Paths with Constants

First we consider efficiently finding paths with constants that are potentially useful for relational learning. The set of candidate constant nodes z ranges over all the nodes in graph, and it is impractical to perform random walks (forward or backward) from all of them. Instead, we propose to efficiently select the set of constant nodes z that are reached over backward random walks starting from correct target nodes in G_i.

Algorithm 2 describes the process of efficiently computing $P(z \rightarrow G_i; \pi)$ for all zs. The process for computing path probabilities in the inverse direction, $P(t \rightarrow z; \pi)$, is similar and is not shown here. Here B is the set of backward path types that are active in more than h queries (backward random walks start from good answer nodes G_i). We don't need to consider other backward paths, because their resulting path with constants have no chance of passing the hits threshold.

As described in Section 2.2.3, our goal is to find all combinations of constant nodes z and paths π for which hits and accuracy exceed thresholds h and a, respectively. The sparse matrices $A(z, \pi)$ and $H(z, \pi)$ represent the accuracies and hits for different combinations of z and π. For each query (s_i, G_i) and each backward path $\pi \in B$, the $P(z \rightarrow G_i; \pi)$ values of all zs are estimated as a sparse vector by a particle filtering-based random walk approach [51]. The accuracies and hits for the z nodes with non-zero probabilities are updated accordingly. Not all of these path types are added to the model; in our experiments, we only add to PRA the top informative paths with constants. Specifically, we elect the K highest scoring paths with constants found, ranked by their accuracy.

We have found in our experiment that many paths with constants are actually

Algorithm 2 Find Constant Paths

Input training queries $\{(s_i, G_i)\}, i = 1...n$
Input backward paths B
Input parameters a, h
Initialize sparse matrix $A(z, \pi) = 0$, $H(z, \pi) = 0$
for each query (s_i, G_i) **do**
  for each backward path $\pi \in B$ **do**
    for each node z s.t. $P(z \to G_i; \pi) \neq 0$ **do**
      $A(z, \pi) += P(z \to G_i; \pi)$
      $H(z, \pi) += 1$
    end for
  end for
end for
$A(z, \pi) /= n$
return (z, π) for which $A(z, \pi) \geq a$, $H(z, \pi) \geq h$

redundant. For instance, "the sports leagues participated by *Boise State* university teams" are very likely to overlap with"the sports leagues participated by *Boise Braves* university teams" (as shown in Table 7.2. If we consider combining this pattern to every university in KB, we end up with generating many paths with constants, which correspond to almost identical features. In this thesis, we take a simple approach to reduce such kind of redundancy—if several paths with constants have exactly the same accuracy value in the feature selection stage, then only (the first) one of them is added to the model.

7.3.3 Long Relational Paths

In order to efficiently find long paths, which are potentially useful for relational learning, we employ *bidirectional* random walks, combining forward and backward walks, as formulated in Section 7.3.1. Our goal is to efficiently find forward path type π_s (starting from the query node s) and backward path type π_t (starting from the target node t), such

Algorithm 3 Bidirectional Random Walks

Input training queries $\{(s_i, G_i)\}, i = 1...n$
Input forward paths F, backward paths B
Input parameters a, h
Initialize matrix $A(\pi_s, \pi_t) = 0$, $H(\pi_s, \pi_t) = 0$
for each query (s_i, G_i) **do**
  Initialize matrix $M(z, \pi_t) = 0$
  for each backward path $\pi_t \in B$ **do**
    for each node z s.t. $P(z \to G_i; \pi_t) \neq 0$ **do**
      $M(z, \pi_t) = P(z \to G_i; \pi_t)$
    end for
  end for
  Initialize matrix $H_0(\pi_s, \pi_t) = 0$
  for each forward path $\pi_s \in F$ **do**
    for each node z s.t. $P(s \to G_i; \pi_s) \neq 0$ **do**
      for each path π_t s.t. $M(z, \pi_t) \neq 0$ **do**
        $A(\pi_s, \pi_t) += P(s \to G_i; \pi_s) M(z, \pi_t)$
        $H_0(\pi_s, \pi_t) += 1$
      end for
    end for
  end for
  for each (π_s, π_t) s.t. $H_0(\pi_s, \pi_t) > 0$ **do**
    $H(\pi_s, \pi_t) += 1$
  end for
end for
$A(\pi_s, \pi_t) /= n$
return $\pi_s \pi_t$ for which $A(\pi_s, \pi_t) \geq a$, $H(z, \pi) \geq h$

that the concatenated path type $\pi_s \pi_t$ is characterized with hits and accuracy rates that exceed thresholds h and a, respectively. This means that in addition to finding long paths

π that connect a given node pair, it is required to efficiently compute $P(s \to t; \pi)$.

Algorithm 3 outlines the proposed process for computing long relational paths features $P(s \to t; \pi)$ using bidirectional graph walks. Here B is the set of backward paths with good support $\{\pi | sup(\pi) \geq h\}$, and F is the set of forward paths with good support $\{\pi | sup(\pi) \geq h\}$. The sparse matrices $A(\pi_s, \pi_t)$ and $H(\pi_s, \pi_t)$ represent the accuracies and hits for long path types, which are the results of different forward-backward path type concatenation. The queries are processed one at a time. For efficiency, we first index the backward random walk results $P(z \to G_i; \pi_t)$ into a sparse matrix M (indexed by z). For each forward path $\pi_s \in F$, each node z, and each backward path $\pi_t \in B$, the accuracy of the concatenated path type $\pi_s \pi_t$ is updated accordingly. To deal with the possibility that node s and t are connected through multiple instantiation of a path type $\pi_s \pi_t$, H is updated according to an intermediate counting matrix H_0. The process for finding long $P(t \to s; \pi)$ paths is similar and not shown here.

Both Algorithm 2 and 3 involve the task of collecting accuracies and hits statistics from individual queries. Therefore, we can employ multi-threaded computing to both of them in order to speedup the training process.

7.4 Experiments

This section reports the results of applying paths with constants and long paths found using bidirectional random walks to the tasks of knowledge base inference and coordinate term extraction. Given a set of labeled queries, we conduct 3-fold cross validation experiments. Since the number of correct answers per query is large in some cases, we report results in terms of Mean Average Precision (MAP), a measure that reflects both precision and recall[1]. All experiments were run on a machine with 16 core Intel Xeon 2.33GHz CPU and 24Gb of memory. We use multiple threads (8) to speedup different stages of PRA

[1] MAP is also used by previous work [58] that this work compares against

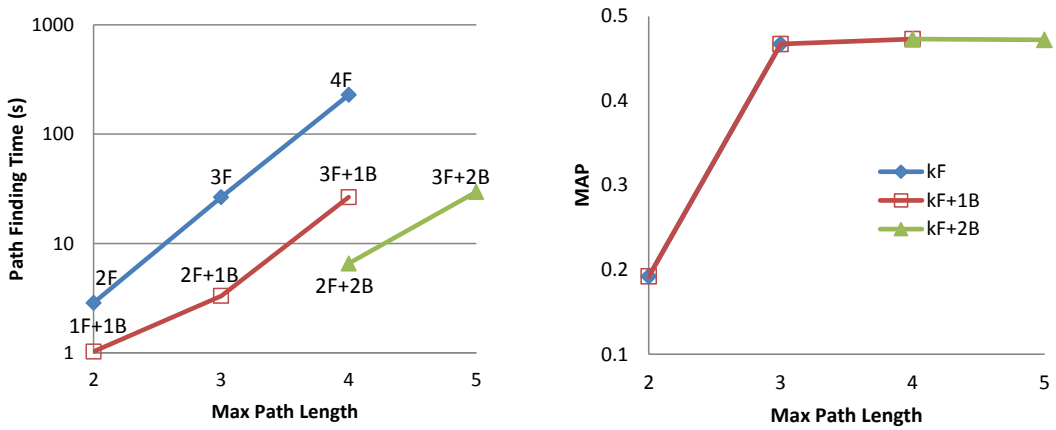
including path finding, random walks, and gradient calculation for parameter estimation. The parameters of *hits* and *accuracy* thresholds have been set to $h = 2$ and $a = 0.001$ for all our experiment, based on empirical tuning using a subset of the training data. Several variants of PRA are evaluated, in order to assess the contribution of including paths with constants, as well as long paths, in the PRA model.

We compare PRA against *Random Walk with Restart* (RWR). We also compare PRA to a variant of the FOIL algorithm [79]. FOIL takes as input a set of positive and negative examples, and uses a "separate-and-conquer" strategy to learn a set of Horn clauses that fit the data well. Each Horn clause is learned by starting with a general rule and progressively specializing it, so that it still covers many positive examples but covers few negative examples. After a clause is learned, the examples covered by that clause are removed from the training set, and the process repeats until no positive examples remain. In order to evaluate FOIL by MAP, we modify the original FOIL implementation[2] to give confidence measures to each predicted new beliefs. To that end, we estimate a conditional probability $\hat{P}(conclusion|preconditions) = (N_+ + m \cdot prior)/(N_+ + N_- + m)$, where N_+ and N_- are the numbers of positive and negative instances matched by this rule in the FOIL training data. We set $m = 5$ and $prior = 0.2$. We tried several different scoring functions, and found that the following gives the best MRR performance— the score of an example is the sum of estimated probabilities $\hat{P}$ for all rules that are applicable to this example.

7.4.1 Case Study: Knowledge Base Inference

We study relational inference in the context of 16 NELL inference tasks defined in Chapter 3. The snapshot of the NELL knowledge base used here corresponds to a graph consisting of 1,661,245 edges, comprised of 353 relations, and 741,794 nodes (concepts). Following Lao et al. [53], we test our approach on 16 link prediction tasks as in Chapter 3. Given a relation r of interest and query node s, our model is evaluated on its ability to find

[2]http://www.rulequest.com/Personal/

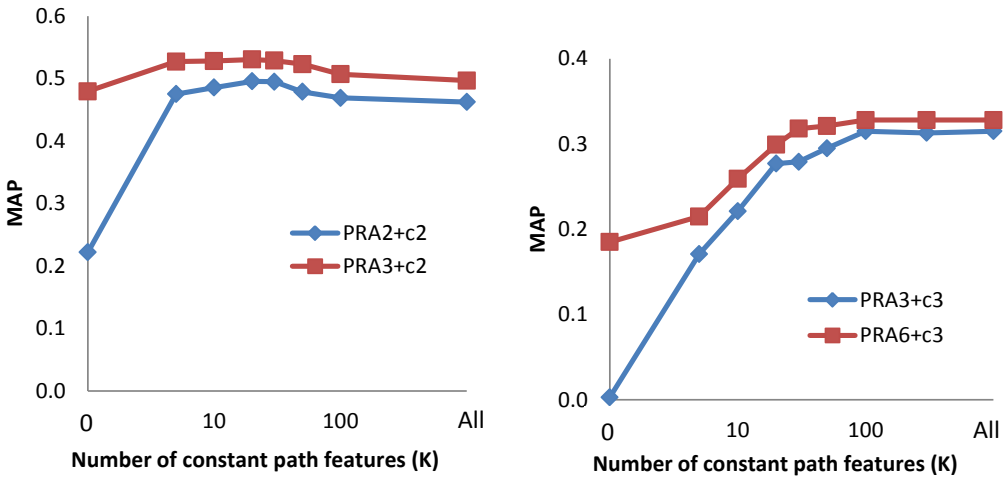


(a) (b)

Figure 7.2: Path finding time (a) and MAP (b) for the KB inference tasks. The markers iF+jB indicate the maximum path length for forward and backward random walks during feature selection. The markers iF indicate the baseline settings, where only forward random walk is applied.

nodes t for which $r(s, t)$ is true.

Figure 7.2 shows the effect of using bidirectional random walks on the knowledge base inference tasks. The curve denoted by kF shows baseline performances, where only forward random walk is applied to feature selection. We can clearly see that the time spent on path finding grows exponentially with the maximum path length. Forward path finding can be executed up to a maximal length of 4 steps due to memory limitation. The curves denoted by $kF + 1B$ and $kF + 2B$ show the results of adding backward random walks for up to one and two steps, respectively. As shown, the time needed for path search with bidirectional random walk is orders of magnitudes faster than the forward search, given the same total path length. In particular, the time spent on path finding is dominated by the longest path segment (either forward or backward)—e.g., the settings 3F, 3F+1B, 3F+2B have similar time complexity. The quality of the bidirectional models (MAP) is comparable to the forward search baseline on the evaluated tasks, indicating that the meaningful paths in this domain are mostly short.



(a) (b)

Figure 7.3: MAP vs. number of paths with constants features for the KB inference tasks (a) and the person name extract task (b). PRAi denotes PRA method with random walk features of length up to $\ell = i$. cj denotes paths with constants up to length j.

We next consider the contribution of learning paths with constants on performance. Figure 7.3a shows MAP results, varying the number of paths with constants modeled K. MAP performance peaks at roughly $K = 20$, and gradually decays as K increases. We therefore set $K = 20$ in this set of experiments.

Table 7.1 reports qualitative results and the time required for learning of PRA variants, comparing them against RWR and FOIL. By comparing PRA-iRW and PRA we can see that using backward random walk features increases training time by 60% percent, but only marginally improves MAP. The table includes the results of adding the top K paths with constants, up to length 2, to the baseline graph walk model. MAP results improve noticeably, when paths with constants are incorporated. As shown, PRA is significantly faster than FOIL, resulting in a competitive MAP performance.

Table 7.2 shows example paths with constants and their interpretations. We can see that they mainly provide informative class biases (priors) to different tasks.

Table 7.1: Compare approaches on the KB inference tasks. #Result is the average number of potential answers found per query. For PRA the maximum random walk length ℓ is 3. PRA-iRW is a variant of PRA without inverse random walk features. ci denotes paths with constants up to length i.

	#Results	Train Time(s)	MAP(SD)
RWR	2269.3	25.6	0.429 (0.005)
FOIL	69.3	18918.1	0.358 (0.005)
PRA-iRW	195.6	10.2	0.477 (0.005)
PRA	219.3	16.7	0.479 (0.005)
PRA+$c1$	219.3	23.3	0.524 (0.005)
PRA+$c2$	250.6	27.1	**0.530 (0.005)**

Table 7.2: Example paths with constants for the knowledge base inference tasks. (ϕ denotes an empty path.)

Paths with Constants	Interpretation
competesWith	
$P(\textit{google} \rightarrow t; \phi)$	Bias toward *Google*
$P(\textit{google} \rightarrow t;$ $\langle \textit{competesWith}, \textit{competesWith} \rangle)$	The companies that are competing with Google's competitors
athletePlaysInLeague	
$P(\textit{mlb} \rightarrow t; \phi)$	Bias toward *MLB*
$P(\textit{boston\_braves} \rightarrow t;$ $\langle athletePlaysForTeam^{-1},$ $\textit{athletePlaysInLeague} \rangle)$	The sports leagues participated by *Boston Braves* university team members
teamPlaysInLeague	
$P(\textit{ncaa} \rightarrow t; \phi)$	Bias toward *NCAA*
$P(\textit{boise\_state} \rightarrow t;$ $\langle teamPlaysInLeague \rangle)$	The sports leagues participated by *Boise State* university teams

7.4.2 Case Study: Coordinate Term Extraction

For the second case study, we follow closely the experimental settings introduced by Minkov and Cohen [58]. In this domain, an entity-relation graph is constructed from a corpus of parsed sentences [21], where nodes denote word types and word mentions, and syntactic relations between word mention pairs are represented by directed edges. A special relation W links each word mention to its corresponding word type. The graph

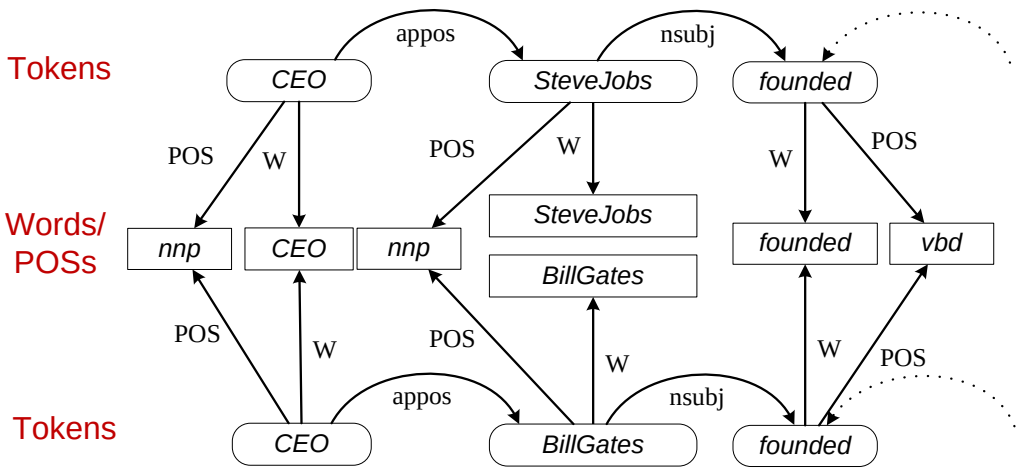
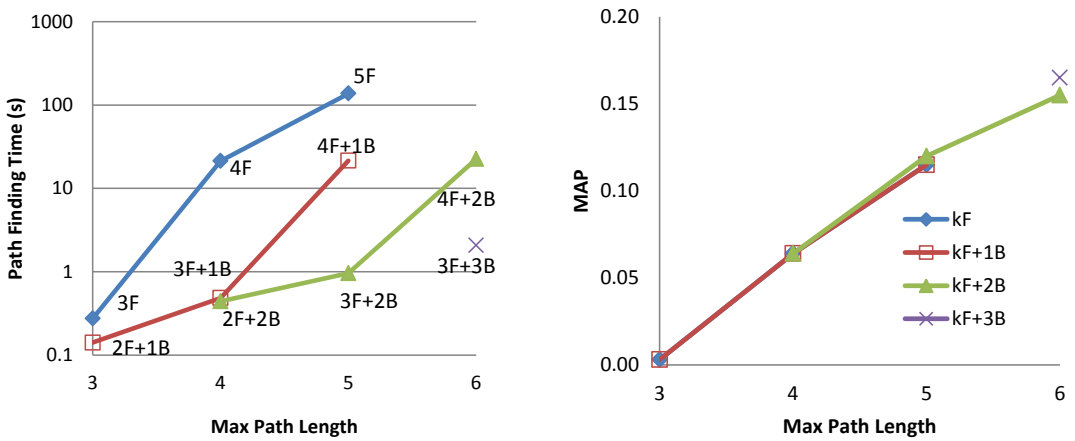


Figure 7.4: Two example sentences in the parsed MUC-6 data set. Relation *W* and *POS* connect tokens in sentences to their word representations and part of speeches. *nsubj* and *appos* are dependency relations in standard dependency parsers.

schema is illustrated in Figure 7.4. In this graph, given a small number of query nodes denoting *person names*, one expects that traversing meaningful syntactic paths through common neighboring words will retrieve a ranked list populated with many additional person names. In the experiments, we use the training set portion of the MUC-6 dataset [76], represented as a graph with 153k nodes and 748K edges. We generated 30 queries, each comprised of 4 person names selected randomly from the known person names, and evaluated performance using true named entity tags available.

Figure 7.5a shows the effect of using bidirectional random walks in the language domain. The curve denoted by kF corresponds to a forward search baseline. As shown, the time required for path finding grows exponentially with the maximal path length k. Forward path finding can process paths only up to length 5 due to the memory limitation of 24Gb of our machine. The curves $kF + 1B$, $kF + 2B$ and $kF + 3B$ show the result of adding backward search from the target nodes of up to 1, 2 or 3 steps, respectively. The results are consistent with our findings on the knowledge base inference tasks. In



(a) (b)

Figure 7.5: Path finding time (a) and MAP (b) for the person name extraction task. The markers iF+jB indicate the maximum path length for forward and backward random walks during feature selection. The markers iF indicate the baseline settings, where only forward random walk is applied.

particular, it is demonstrated that, by combining random walks in both directions, path finding is much faster. It is also shown that performance in terms of MAP (Figure 7.5b) mostly depends on the total path length explored, rather than on the search mode. In this domain, using bidirectional random walks enables us to exhaustively explore paths up to 6 steps long, which further improve the model performance[3].

Table 7.3 shows example long paths learned for the person name extraction task. These paths are similar to the top ranked paths found in previous work by Minkov and Cohen [58]. In comparison, their results on this dataset using paths of up to 6 steps long measured up to 0.09 in MAP. The results here reach roughly 0.15 in MAP with long relational paths.

Interestingly, by repeatedly applying the path concatenation procedure, a novel 9 step path can be found $P(s \rightarrow t; \langle W^{-1}, conj\_and^{-1}, W, W^{-1}, conj\_and^{-1}, W, W^{-1}, conj\_and, W \rangle)$. It corresponds to a chain of words linked by conjunctions,

[3]MAP values are low for this dataset, because there is a large number of correct answers, and many of them are not reachable from the seed nodes.

Table 7.3: Example long paths for person name extraction

Long Paths	Interpretation
$P(s \rightarrow t; \langle W^{-1}, conj\_and^{-1}, W, W^{-1}, conj\_and, W \rangle)$	Entities connected to the seeds by two conjunction relations, e.g. "Bill Gates and Steve Jobs collaborated ..."
$P(s \leftarrow t; \langle W^{-1}, nsubj^{-1}, W, W^{-1}, nsubj, W \rangle)$	Entities having the same action as the seeds e.g. "Bill Gates founded ..." (as in Figure 7.4)
$P(s \rightarrow t; \langle W^{-1}, appos, W, W^{-1}, appos^{-1}, W \rangle)$	Entities having the same appositive modifier as the seeds e.g. "Bill Gates, the CEO of ..." (as in Figure 7.4)

Table 7.4: Example paths with constants on the person name extraction task. #Result is the average number of potential answers found per query. For PRA the maximum random walk length ℓ is 6. PRA-iRW is a variant of PRA without inverse random walk features. ci denotes paths with constants up to length i.

	#Results	Train Time(s)	MAP(SD)
RWR	132,832.9	7,375.7	0.017 (0.01)
FOIL	683.1	366,558.6	0.167 (0.02)
PRA-iRW	15,880.4	277.8	0.107 (0.02)
PRA	15,925.9	449.4	0.167 (0.02)
PRA+$c2$	15,944.1	556.2	0.186 (0.02)
PRA+$c3$	16,034.0	643.6	**0.316 (0.03)**

mimicking a bootstrapping process, and improves MAP over the length-6 model. However, a study of repeated path concatenation is beyond the scope of this thesis.

Figure 7.3b shows MAP results, varying K, the number of paths with constants modeled. As shown, MAP performance improves as these paths are added to the model, indicating their usefulness. Performance picks (and stabilizes) at roughly $K = 100$. We therefore set $K = 100$.

Finally, Table 7.4 shows the impact of adding paths with constants on this dataset, comparing against other PRA variants, RWR and FOIL in terms of qualitative and learning time performance. PRA is much faster than RWR or FOIL methods, and gives competitive MAP performance to FOIL. The RWR approach is ineffective on this task, because similarity in this domain is represented by a relatively small set of long paths, rather than

Table 7.5: Example paths with constants for person name extraction. The relation POS links tokens in sentences to their part of speeches.

Paths with Constants	**Interpretation**
$P(t \rightarrow said; \langle W^{-1}, nsubj^{-1}, W\rangle)$	The subjects of "said"
$P(t \rightarrow says; \langle W^{-1}, nsubj^{-1}, W\rangle)$	The subjects of "says"
$P(t \rightarrow vbg; \langle W^{-1}, nsubj^{-1}, POS\rangle)$	The subjects of verbs
$P(t \rightarrow nnp; \langle W^{-1}, POS\rangle)$	Entities having *noun compound* POS
$P(t \rightarrow nn; \langle W^{-1}, appos, POS\rangle)$	Entities having *appositive* constructions e.g. "Bill Gates, the CEO of ..."
$P(t \rightarrow nn; \langle W^{-1}, poss^{-1}, POS\rangle)$	Entities having *possessive* constructions e.g. "Bill Gates' view of ..."

general node association in the graph. As shown by the MAP results, adding features of paths with constants boosts performance. In this domain, these features mainly provide class priors for different part of speech categories. By comparing PRA-iRW and PRA we can see that backward random walk features although noticeably increases training time, but also noticeably improves MAP.

Table 7.5 shows examples of the top weighted paths with constants learned. They indicate that the subjects of verbs (especially "said" and "says") and nouns (especially those with apposition and possessive constructions) are likely to be person names. Compared to PRA, FOIL generates fewer rules, which are characterized with low recall and high precisions, such as

$$W(B, A) \wedge POS(B, nnp) \wedge nsubj(D, B) \wedge W(D, said) \wedge appos(B, F) \rightarrow person(A).$$

This indicates an interesting future work of using products of basic random walk features to express their conjunctions. Such conjunctions can be represented as features that include random walks between constant node z and query node s.

7.5 Summary

We have shown that it is feasible to efficiently learn random walk features which involve long paths or constant nodes. Towards the first goal, we use backward random walks from target nodes to efficiently evaluate constant path types. Towards the second goal, we combine forward and backward random walks from both query and target nodes to evaluate the goodness of long path patterns. Knowledge base inference and person name extraction experiments show significant improvement in training time and model quality.

Chapter 8

Conclusion and Future Work

8.1 Summary of Contributions

This thesis introduces *random walk inference*, which combines the expressiveness of logic, the robustness of statistical learning, and the scalability of random walk models. It also introduces the path ranking algorithm (PRA), a method for learning random walk inference for a family of restricted first order rules. A key to PRA's efficiency and robustness is the stochastic approach to feature instantiation. As we have described, simple but effective metrics can be used to explore the large feature space (Chapter 2 and 7), and the instantiation cost of such features can be computationally bounded using sampling based random walk approaches (Chapter 5). Experiments show that we are able to apply relational learning at scales not possible before.

Compared to existing ILP methods such as FOIL, the key characteristics of this new inference method are as follows:

- The evidence in support of inferring a relation instance $r(s, t)$ is based on many existing paths between s and t in the KB, combined by a learned logistic function.
- The confidence in an inference is sensitive to the current state of the knowledge base,

and the specific entities being queried.

- Experimentally, the inference method yields many more moderately-confident inferences than the Horn clauses learned by FOIL, and is more accurate than random walk with restart.
- The learning and inference are more efficient than FOIL, in part because we can exploit efficient approximation schemes for random walks.

Although link prediction, the task performed by PRA, is a special case of logical inference, it has may applications. We applied PRA to several applications including inference in a large scale knowledge base (Chapter 3), recommendation problems in the biological literature (Chapter 4), relation extraction from parsed text with knowledge base (Chapter 6), and coordinate term extraction (Chapter 7). We have shown that PRA has very competitive speed and accuracy compared to existing approaches such as random walks with restart and FOIL.

8.2 Limitations

Because of its efficiency, PRA can potentially be applied to a wide range of tasks, for which a knowledge base might help. However, the presented framework has its limitations in several aspects, which point to possible avenues of future research.

- **PRA Implementation**—the current PRA implementation cannot effectively handle graphs with large numbers of virtual nodes. It also suffers from an overfitting problem when facing a large number of correlated path features.
- **Model Expressiveness**—compared to logic programs, PRA's language for defining features is very restricted, which prevents it from effectively encode certain type of semantics.
- **Multitask Learning**—the current framework learns models of different relations in a KB separately. The models can neither share useful prior information about which

relations (or combinations of relations) are more likely to be useful, nor can they make decisions based on each other's decisions.

In the following, we describe these limitations in detail, and discuss possible solutions.

8.3 PRA Implementation Extensions

There are two directions in which the current PRA implementation can be improved.

Unsupervised Learning to Find Groups of Related paths

As we have discussed in Chapter 7, many paths with constants are actually redundant. For instance, "the sports leagues participated in by *University of Michigan*" are very likely to overlap with "the sports leagues participated in by *University of Wisconsin*". If we consider applying this rule to every university in KB, we end up with generating many paths with constants, which correspond to almost identical features. Using all of them as features for logistic regression training would lead to overfitting as shown in the knowledge base inference task.

In this thesis, we take a simple approach to reduce this kind of redundancy—if several paths with constants have exactly the same accuracy value in the feature selection stage, then only (the first) one of them is added to the model. One interesting research direction is to employ more robust feature selection or dimensionality reduction methods to reduce the redundancy of the random walk features.

Dynamic Graph Instantiation

This thesis assumes that the graph representation of knowledge base is given before PRA does any random walk or learning. However, for domains with large number of nodes or even an infinite number of nodes, maybe only a small subset of nodes are relevant to PRA's queries. In this case it is more practical to generate parts of PRA's graph during

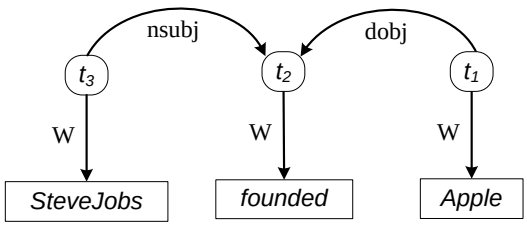


Figure 8.1: Path conjunction

random walk. For example, social computing tasks may require pairs of persons to be represented as nodes in the knowledge base. However, it is unnecessary to instantiate possible pairs among all persons, since most of them are not related to each other anyway. For text processing tasks, we might want to conceptually treat any character sequence as a node in the graph representation, but only instantiate the ones generated by string manipulation operators such as string concatenation, or a spell checker. Similarly, for natural language processing tasks, we might want to conceptually treat any sub-span of a word sequence (i.e. a sentence) as a node in the graph representation, but only instantiate the ones relevant to the queries.

8.4 Path Language vs. Logic Programs

Although PRA is very efficient, it uses a very restricted language to express rules compared to logic programs such as FOIL.

Existential Quantifier

PRA learns rules that involve source, target, and constant nodes. In comparison, most logical programs can include rules that involve unbounded nodes such as y in $\mathit{IsA}(x, \mathit{writer}) \leftarrow \mathit{Write}(x, y)$. As we can see, such rules can compactly express certain semantics. The incorporation of such rules to PRA is straight forward—we can defined the probability of reaching any node from a bounded node, through a particular path.

Conjunctions of Paths

The random walk features in PRA correspond to edge sequences. However, it is desirable, for many applications, to have features that correspond to more complex structures. For example, Mintz et al. [60] discovered, from their relation extraction experiment, that it is useful to define features that are conjunctions of dependency chains and the word/POS representations of tokens along these chains. For example, the rule $\langle W(s, t_1), dobj(t_1, t_2), W(t_2, founded), nsubj(t_3, t_2), W(t_3, t)\rangle$ would be useful in predicting the CEO of a company, given the fragment of parse tree shown in Figure 8.1. Here W relation links tokens in sentences to their word representations. In PRA's terminology, each of these features corresponds to a tree T with constant nodes, source nodes, and the target node as its leaves, and KB relations as edges.

In order to use such tree structured rules in the random walk inference framework, we need to define random walk processes on tree structures instead of on paths. Note that the random walk probabilities $P(s \rightarrow t; \pi)$ defined in this thesis can be interpreted as the probability that an instantiation of path type π ends at node t, if started from s, and choosing possible node instantiations with equal probabilities at each step. Here is a possible way to generalize this definition to tree shaped rules.

Let's assume that $[s_1, ..., s_d]$ is the set of source nodes and constant nodes, and treat the target node t as as the root of instantiation. Then we can define a *forward random walk probability* $P([s_1, ..., s_d] \rightarrow t; \pi)$ as the chance of starting d walkers from the leaf nodes and instantiating a tree structure of rule π, which ends at node t. Here we require that wherever π has a merging structure such as $\left\langle s \xrightarrow{r_1} x, s \xrightarrow{r_2} x \right\rangle$ the walkers that follow this pattern meet at exactly the same node otherwise the join random walk fails, and, once they meet, they walk together as if they are a single walker. By setting t as the root, we can efficiently calculate such probabilities for all possible t using dynamic programming. A set of tree rules, as well as their random walk results, can be compactly represented as a relation network (as oppose to a relation tree [58, 52].

Similarly, we can define a *backward random walk probability* $P(t \rightarrow [s_1, ..., s_d]; \pi)$ as the chance of starting one walker from the root t and instantiating a tree structure of rule π, which ends at nodes $[s_1, ..., s_d]$. Here we require that, wherever π has a diverging structure such as $\left\langle x \xrightarrow{r_1} s_1, x \xrightarrow{r_2} s_2 \right\rangle$, the walker that follows this pattern splits into two or more walkers (one for each branch). The result walkers walk independently thereafter following different branches of the rule π.

Representing K-ary Relations

The graph representation of PRA only has binary relations. However, many types of knowledge, such as events, are naturally represented as K-ary relations. For example, "Alice met Bob in 1999" expresses a ternary relation *MeetInYear*(*Alice*, *Bob*, *1999*). FOIL represents these K-ary relations as K-ary predicates.

Learning rules that contain K-ary relations leads to a much larger hypothesis space than the one PRA considers. The hypothesis space of PRA (sequences of binary relations) can be compactly represented as a path tree. The size of such a tree is generally $O(|R|^{\ell})$, where ℓ is the number of relations allowed in a rule, and $|R|$ is the number relations in a KB. If K-ary relations are considered, the hypothesis space can be represented as a directed acyclic graph (DAG), where each node can have one or more parents. More specifically, a node generated by a K-ary relation has $K - 1$ parents. The size of such a DAG is generally double exponential $O((|R| \cdot K)^{(K-1)^{\ell}})$.

Instead of exhaustive search, FOIL relies on local searches based on refinement operators. A similar strategy can potentially be applied to random walk inference.

Learning K-ary Target Relations

This thesis started from a link prediction task: given a directed edge-labeled graph KB representing background knowledge, a relation r, and a source node s (also called a query), find the set of nodes G s.t. $r(s, t)$ for each $t \in G$. Then Chapter 4 extended this definition: instead of having a single source node s, each query has d sets of source

nodes $[S_1, ..., S_d]$, $d \geq 1$, where each set contains source nodes of a certain type. It is an interesting direction to generalize this task definition even further to situations where there is no target node (classification tasks), or where there are more than one target node (e.g. the query *PersonPlaysSportAtLocation*(*Bob*, a, b)).

The extension to classification tasks is straight forward: instead of defining random walks between source nodes and target nodes, we just define random walks between source nodes and constant nodes, or among source nodes themselves.

The extension to more than one target node is more involved. One possible solution [55] is to conceptually treat each pair of nodes in a KB as a new node of "composition type". These composite nodes are connected to regular nodes through composition relations, e.g. *ComposeOf*($[Bob, Alice], Bob, Alice$), where $[Bob, Alice]$ is a composite node. This allow the positive and negative samples of a link prediction task to still be individual nodes.

8.5 Joint Learning of Multiple Relations

The rules defined in PRA are based on existing edges in a KB. However, it can be beneficial for a model to make decision based on other models' decisions—i.e. to impute missing edges, or correct erroneous edges in a KB. One can imagine training several PRA models for different relations at the same time, where the path types of these PRA models are defined as either existing edges in the KB or other PRA models. For instance, a PRA model that predicts the hobby of a person, may contain a path $\langle PRA_{HasFriend}, \textit{HasHobby} \rangle$, which invokes another PRA model $PRA_{HasFriend}$, which predicts the friend of a person.

Since the models are dependent on each other, it is natural to consider training them in a single coherent process. The objective function for learning these models collectively can be the sum of all their individual objective functions. To optimize such a combined objective function, we can either do coordinate descent, or joint update. In the former

case, we update the parameters of only one model at a time, while fixing the parameters for all other model. This approach is easy to implement, but may converge slowly. In the later case, we update the parameters of all model at the same time. This approach has good convergence speed, but involves calculating gradient of each model with respect to parameters of other models.

One interesting question arises in this multitask learning setting—since the models of different target relations may share the same rules or rule fragments, it can potentially useful for them to share the knowledge about which rule (or rule fragments) are more likely to lead to good model. One possible way to achieve this, is to learn "macro" relations, which are combinations of several existing relations. For example, the macro relation $\textit{Grandpa}(x, y)$ can be defined as $\textit{Father}(x, z) \wedge \textit{Father}(z, y)$. A more powerful way to share knowledge between models is predicate invention. For example, a new relation $\textit{Grandpa}(x, y)$ may be discovered from existing relations in a KB, and a corresponding model is learnt, which involves rules such as $\textit{Grandpa}(x, y) \leftarrow \textit{Father}(x, z) \wedge \textit{Father}(z, y)$ and $\textit{Grandpa}(x, y) \leftarrow \textit{Mother}(x, z) \wedge \textit{Father}(z, y)$.

Another interesting question arises, if we allow the definition of PRA models to be recursive (e.g. a PRA model relies on itself, or two PRA models rely on each other). Some kind of computation complexity measure or merit measure is needed to prevent infinite calls among PRA models when they make decisions.

References

[1] G Adomavicius and A Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. In *IEEE Transactions on Knowledge and Data Engineering*, volume 17, pages 734–749. Department of Molecular and Cell Biology, University of California, Berkeley 94720., IEEE Educational Activities Department, 2005.

[2] Alekh Agarwal, Soumen Chakrabarti, and Sunny Aggarwal. Learning to rank networked entities. In *KDD*, 2006.

[3] Eugene Agichtein and Luis Gravano. Snowball: extracting relations from large plain-text collections. In *ACM conference on Digital libraries - DL*, pages 85–94, New York, New York, USA, 2000. ACM Press.

[4] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. *ICDE*, 2002.

[5] Galen Andrew and Jianfeng Gao. Scalable training of l^1-regularized log-linear models. *ICML*, pages 33–40, 2007.

[6] Andrew Arnold and William W. Cohen. Information extraction as link prediction: Using curated citation networks to improve gene detection. *WASA*, pages 541–550, 2009.

[7] Javed A. Aslam, Evangelos Kanoulas, Virgiliu Pavlu, Stefan Savev, and Emine Yilmaz. Document selection methodologies for efficient and effective learning-to-rank. *SIGIR*, pages 468–475, 2009.

[8] Marko Balabanovic and Yoav Shoham. Fab : content-based , collaborative recommendation .(Special Section : Recommender Systems). *Communications of the ACM*, 66(March):1–7, 1997.

[9] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open Information Extraction from the Web. In *IJCAI*, pages 2670–2676, 2007.

[10] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S Sudarshan. Keyword Searching and Browsing in Databases using BANKS. *ICDE*, 2002.

[11] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1247–1250, New York, NY, USA, 2008. ACM.

[12] MJ Cafarella, M Banko, and O Etzioni. Relational Web Search. In *www*, 2006.

[13] 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.

[14] Soumen Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. *Proceedings of the 16th international conference on World Wide Web WWW 07*, 10(3):571, 2007.

[15] Soumen Chakrabarti and Alekh Agarwal. Learning Parameters in Entity Relationship Graphs from Ranking Preferences. *PKDD*, 2006.

[16] Brian Cohen and Claude Sammut. Object recognition and concept learning with CONFUCIUS. *Pattern Recognition*, 15(4):309–316, January 1982.

[17] William W. Cohen and C. David Page Jr. Polynomial Learnability and Inductive Logic Programming: Methods and Results. In *New Generation Comput.*, volume 13, pages 369–409, 1995.

[18] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[19] Aron Culotta, Andrew McCallum, and Jonathan Betz. Integrating probabilistic extraction models and data mining to discover relations and patterns in text. *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, (June):296–303, 2006.

[20] Bhavana Bharat Dalvi, Meghana Kshirsagar, and S Sudarshan. Keyword search on external memory data graphs. In *Proc VLDB Endow*, volume 1, pages 1189–1204. VLDB Endowment, 2008.

[21] Marie-Catherine De Marneffe, Bill MacCartney, and Christopher D Manning. Generating Typed Dependency Parses from Phrase Structure Trees. In *Proceedings LREC2006*, pages 449–454, 2006.

[22] Marie-Catherine de Marneffe and Chris Manning. Stanford dependencies. `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=citeURL`, 2008.

[23] Luc De Raedt. *Logical and Relational Learning*, volume 38 of *Cognitive Technologies*. Springer, 2008.

[24] Luc De Raedt and Luc Dehaspe. Clausal Discovery. *Machine Learning*, 26(2-3):99–146, 1997.

[25] Rodrigo De Salvo Braz, Eyal Amir, and Dan Roth. Lifted First-Order Probabilistic Inference. In F Giunchiglia and L Pack Kaelbling, editors, *Introduction to Statistical Relational Learning*, pages 1125–1319. AAAI Press, 2007.

[26] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[27] Michelangelo Diligenti, Marco Gori, and Marco Maggini. Learning Web Page Scores by Error Back-Propagation. In *IJCAI*, 2005.

[28] Zhicheng Dou, Ruihua Song, and Ji-Rong Wen. A large-scale evaluation and analysis of personalized search strategies. *WWW*, pages 581–590, 2007.

[29] Saso Dzeroski. Relational Data Mining. In *The Data Mining and Knowledge Discovery Handbook*, pages 869–898. 2005.

[30] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in knowitall. In *WWW*, page 100, New York, New York, USA, 2004. ACM Press.

[31] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-maria Popescu, Tal Shaked, Stephen Soderl, Daniel S. Weld, and Er Yates. Unsupervised Named-Entity Extraction from the Web: An Experimental Study. *Artificial Intelligence*, 165:91–134, 2005.

[32] D. Fogaras and B. Rácz. Towards fully personalizing PageRank. In *Proceedings of the 3rd Workshop on Algorithms and Models for the Web-Graph (WAW2004), in conjunction with FOCS 2004.*, 2004.

[33] N Friedman, L Getoor, D Koller, and A Pfeffer. Learning Probabilistic Relational Models. In *IJCAI*, volume 16, pages 1300–1309. Citeseer, 1999.

[34] Lisa Getoor and Ben Taskar. Introduction to statistical relational learning. In *The MIT Press*, 2007.

[35] David Goldberg, David A. Nichols, Brian M. Oki, and Douglas B. Terry. Using Collaborative Filtering to Weave an Information Tapestry. *Commun. ACM*, 35(12):61–70, 1992.

[36] Aria Haghighi and Dan Klein. Simple coreference resolution with rich syntactic and semantic features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1152–1161, Singapore, August 2009. Association for Computational Linguistics.

[37] Taher H Haveliwala. Topic-sensitive PageRank. *Proceedings of the eleventh international conference on World Wide Web WWW 02*, 20(650):517, 2002.

[38] Hao He, Haixun Wang, Jun Yang, and Philip S Yu. BLINKS : Ranked Keyword Searches on Graphs. In *Most*, SIGMOD '07, pages 305–316. ACM, 2007.

[39] Qi He, Jian Pei, Daniel Kifer, Prasenjit Mitra, and C. Lee Giles. Context-aware citation recommendation. 2010.

[40] Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Computational linguistics*, volume II of *COLING '92*, pages 539–545. Association for Computational Linguistics Morristown, NJ, USA, Association for Computational Linguistics, 1992.

[41] William C. Hill, Larry Stead, Mark Rosenstein, and George W. Furnas. Recommending and Evaluating Choices in a Virtual Community of Use. In *CHI*, pages 194–201, 1995.

[42] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, 1982.

[43] Vagelis Hristidis and Yannis Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. *VLDB*, 2002.

[44] A Jaimovich, O Meshi, and N Friedman. Template-based inference in symmetric relational Markov random fields. In *23rd UAI*, 2007.

[45] Alpa Jain and Patrick Pantel. Factrank: Random walks on a web of facts. In *COLING*, pages 501–509, 2010.

[46] Kristian Kersting and Luc De Raedt. Bayesian Logic Programs. *CoRR*, cs.AI/0111, 2001.

[47] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In Erhard Hinrichs and Dan Roth, editors, *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 423–430. Association for Computational Linguistics, July 2003.

[48] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. GroupLens: applying collaborative filgering to Usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[49] Ken Lang. NewsWeeder : Learning to Filter Netnews (To appear in ML 95). *Forbes*, 1995.

[50] Pat Langley, Herbert A. Simon, Gary L. Bradshaw, and Jan M. Zytkow. Scientific discovery: computational explorations of the creative process. *MIT Press*, April 1987.

[51] Ni Lao and William W. Cohen. Fast query execution for retrieval models based on path-constrained random walks. In *KDD*, page 881, New York, New York, USA, 2010. ACM Press.

[52] Ni Lao and William W. Cohen. Relational retrieval using a combination of path-constrained random walks. In *Machine Learning*, volume 81, pages 53–67, July 2010.

[53] 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, 2011.

[54] Ni Lao, Amarnag Subramanya, Fernando Pereira, and William W. Cohen. Reading The Web with Learned Syntactic-Semantic Inference Rules. In *EMNLP*, page (to appear), 2012.

[55] Ni Lao, Jun Zhu, Liu Liu, Yandong Liu, and William W. Cohen. Efficient relational learning with hidden variable detection. *NIPS*, 2010.

[56] N Lavrač, S Džeroski, and M Grobelnik. Learning nonrecursive definitions of relations with linus. In Yves Kodratoff, editor, *Proc 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 265–281. Springer-Verlag, 1991.

[57] Mihai Lupu, Florina Piroi, Xiangji Huang, Jianhan Zhu, and John Tait. Overview of the trec 2009 chemical ir track. *TREC-18*, 2009.

[58] Einat Minkov and William W Cohen. Learning Graph Walk Based Similarity Measures for Parsed Text. *EMNLP*, 2008.

[59] Einat Minkov, William W Cohen, and Andrew Y Ng. Contextual search and name disambiguation in email using graphs. *SIGIR*, 2006.

[60] Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL/AFNLP*, pages 1003–1011, 2009.

[61] Raymond J. Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries - DL '00*, pages 195–204, New York, New York, USA, 2000. ACM Press.

[62] Stephen Muggleton. Inductive Logic Programming. *New Generation Comput.*, 8(4):295–, 1991.

[63] Stephen Muggleton. Inverse Entailment and Progol. In *New Generation Comput.*, volume 13, pages 245–286, 1995.

[64] Stephen Muggleton. Stochastic Logic Programs. *NEW GENERATION COMPUTING*, 1996.

[65] Zaiqing Nie, Yuanzhi Zhang, Ji-Rong Wen, and Wei-Ying Ma. Object-level ranking: bringing order to Web objects. *WWW*, 2005.

[66] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. In *World Wide Web Internet And Web Information Systems*, number 1999-66, pages 1–17. Stanford Digital Library Technologies Project, Technical report, Stanford Digital Library Technologies Project, 1998, 1998.

[67] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Using pagerank to characterize web structure. In *COCOON*, pages 330–339, 2002.

[68] Patrick Pantel and Marco Pennacchiotti. Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations. In *ACL*, 2006.

[69] Virgil Pavlu. Large scale ir evaluation. *PhD thesis, Northeastern University, College of Computer and Information Science*, 2008.

[70] Michael Pazzani and Daniel Billsus. Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning*, 27(3):313–331, 1997.

[71] Michael Pazzani, Cliff Brunk, and Glenn Silverstein. A Knowledge-Intensive Approach to Learning Relational Concepts. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 432–436. Morgan Kaufmann, 1991.

[72] Simon Perkins, Kevin Lacker, and James Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. In *JMLR*, 2003.

[73] D Poole. Logic Programming, Abduction and Probability. *New Generation Computing*, 11(3):377–400, 1993.

[74] David Poole. First-order probabilistic inference. In *IJCAI*, volume 18, pages 985–991. University of Illinois at Urbana-Champaign, Citeseer, 2003.

[75] Alexandrin Popescul and L H Ungar. Dynamic feature generation for relational learning. In *Workshop on MultiRelational Mining MRDM2004 at ACM SIGKDD*, volume 173. Citeseer, 2004.

[76] Muc Proceedings. *Proceedings of the Sixth Message Understanding Conference (MUC-6)*. Morgan Kaufmann, 1995.

[77] J R Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[78] J. Ross Quinlan. Learning Logical Definitions from Relations. In *Machine Learning*, volume 5, pages 239–266, 1990.

[79] J. Ross Quinlan and R. Mike Cameron-Jones. FOIL: A Midterm Report. In *ECML*, pages 3–20, 1993.

[80] Sriram Raghavan and Hector Garcia-Molina. Representing Web graphs. In Umeshwar Dayal, Krithi Ramamritham, and T M Vijayaraman, editors, *Proceedings 19th International Conference on Data Engineering Cat No03CH37405*, pages 405–416. Stanford University Database Group, Ieee, 2003.

[81] Deepak Ravichandran and Eduard Hovy. Learning surface text patterns for a Question Answering system. *ACL*, 2(July):41–47, 2002.

[82] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work - CSCW '94*, pages 175–186, New York, New York, USA, 1994. ACM Press.

[83] Elaine Rich. User modeling via stereotypes. *Cognitive Science*, 3(4):329–354, 1979.

[84] B L Richards and R J Mooney. First-Order Theory Revision. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 447–451. Morgan Kaufmann, 1991.

[85] Bradley L. Richards and Raymond J. Mooney. Learning Relations by Pathfinding. In *AAAI*, pages 50–55, 1992.

[86] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 2006.

[87] Dan Roth. Constraints based Taxonomic Relation Classification. *Computational Linguistics*, (October):1099–1109, 2010.

[88] C Sammut and R Banerji. Learning Concepts by Asking Questions. In R S Michalski, J G Carbonell, and T M Mitchell, editors, *Machine Learning An Artificial Intelligence Approach*, volume 2, pages 167–192. Morgan Kaufmann, 1986.

[89] Taisuke Sato and Yoshitaka Kameya. PRISM: A Language for Symbolic-statistical Modeling. In *In Proceedings of the 15th International Joint Conference on Artificial Intelligence IJCAI97*, pages 1330–1335, 1997.

[90] Stefan Schoenmackers, Oren Etzioni, and Daniel S. Weld. Scaling Textual Inference to the Web. In *EMNLP*, pages 79–88, 2008.

[91] E Y Shapiro. *Algorithmic Program Debugging*, volume 43 of *ACM Distinguished Disertation Series*. MIT Press, 1983.

[92] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". In *CHI*, pages 210–217, 1995.

[93] Parag Singla and Pedro Domingos. Lifted First-Order Belief Propagation. In *AAAI*, pages 1094–1099, 2008.

[94] Rion Snow, Daniel Jurafsky, and Andrew Y Ng. Learning syntactic patterns for automatic hypernym discovery. In Lawrence K Saul, Yair Weiss, and Léon Bottou, editors, *NIPS*, volume 17, pages 1297–1304. Stanford University, Citeseer, 2005.

[95] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. Semantic Taxonomy Induction from Heterogenous Evidence. In *ACL*, 2006.

[96] Fabian M. Suchanek, Georgiana Ifrim, and Gerhard Weikum. Combining linguistic and statistical analysis to extract relations from web documents. In *KDD*, page 712, New York, New York, USA, 2006. ACM Press.

[97] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[98] Hanghang Tong, Christos Faloutsos, and Jia-yu Pan. Fast Random Walk with Restart and Its Applications. In Christopher Wade Clifton and Et Al., editors, *Sixth International Conference on Data Mining ICDM06*, volume 7, pages 613–622. Ieee, 2006.

[99] Ryen W. White, Mikhail Bilenko, and Silviu Cucerzan. Studying the use of popular destinations to enhance web search interaction. *SIGIR*, pages 159–166, 2007.

[100] Alexander Yates, Michele Banko, Matthew Broadhead, Michael J. Cafarella, Oren Etzioni, and Stephen Soderland. TextRunner: Open Information Extraction on the Web. In *HLT-NAACL (Demonstrations)*, pages 25–26, 2007.

[101] John M. Zelle, Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney. Inducing logic programs without explicit negative examples. In *Proceedings of the Fifth International Workshop on Inductive Logic Programming (ILP-95)*, pages 403–416, Leuven, Belgium, 1995.

[102] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal Of The Royal Statistical Society Series B*, 67(2):301–320, 2005.