7-2011

Random Walk Inference and Learning in A Large Scale Knowledge Base

Ni Lao
Carnegie Mellon University

Tom Mitchell
Carnegie Mellon University, mitchell@andrew.cmu.edu

William W. Cohen
Carnegie Mellon University, wcohen@cs.cmu.edu

Published In
Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, 529-539.

Random Walk Inference and Learning in A Large Scale Knowledge Base

Ni Lao
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
nlao@cs.cmu.edu

Tom Mitchell
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
tom.mitchell@cs.cmu.edu

William W. Cohen
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
wcohen@cs.cmu.edu

Abstract

We consider the problem of performing learning and inference in a large scale knowledge base containing imperfect knowledge with incomplete coverage. We show that a soft inference procedure based on a combination of constrained, weighted, random walks through the knowledge base graph can be used to reliably infer new beliefs for the knowledge base. More specifically, we show that the system can learn to infer different target relations by tuning the weights associated with random walks that follow different paths through the graph, using a version of the Path Ranking Algorithm (Lao and Cohen, 2010b). We apply this approach to a knowledge base of approximately 500,000 beliefs extracted imperfectly from the web by NELL, a never-ending language learner (Carlson et al., 2010). This new system improves significantly over NELL's earlier Horn-clause learning and inference method: it obtains nearly double the precision at rank 100, and the new learning method is also applicable to many more inference tasks.

1 Introduction

Although there is a great deal of recent research on extracting knowledge from text (Agichtein and Gravano, 2000; Etzioni et al., 2005; Snow et al., 2006; Pantel and Pennacchiotti, 2006; Banko et al., 2007; Yates et al., 2007), much less progress has been made on the problem of drawing reliable inferences from this imperfectly extracted knowledge. In particular, traditional logical inference methods are too brittle to be used to make complex inferences from automatically-extracted knowledge, and probabilistic inference methods (Richardson and Domingos, 2006) suffer from scalability problems. This paper 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, 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 present a trainable inference method that learns 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 (Carlson et al., 2010).

1.1 The NELL Case Study

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 (Carlson et al., 2010). 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. 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*.

NELL is given as input an ontology that defines hundreds of categories (e.g., person, beverage, athlete, sport) and two-place typed relations among these categories (e.g., *atheletePlaysSport*(⟨athlete⟩, ⟨sport⟩)), which it must learn to extract from the web. It is also provided a set of 10 to 20 positive seed examples of each such category and relation, along with a downloaded collection of 500 million web pages from the ClueWeb2009 corpus (Callan and Hoy, 2009) as unlabeled data, and access to 100,000 queries each day to Google's search engine. Each day, NELL has two tasks: (1) to extract additional beliefs from the web to populate its growing knowledge base (KB) with instances of the categories and relations in its ontology, and (2) to learn to perform task 1 better today than it could yesterday. We can measure its learning competence by allowing it to consider the same text documents today as it did yesterday, and recording whether it extracts more beliefs, more accurately today.[1]

NELL uses a large-scale semi-supervised multi-task learning algorithm that couples the training of over 1500 different classifiers and extraction methods (see (Carlson et al., 2010)). Although many of the details of NELL's learning method are not central to this paper, two points should be noted. First, NELL is a multistrategy learning system, with components that learn from different "views" of the data (Blum and Mitchell, 1998): for instance, one view uses orthographic features of a potential entity name (like "contains capitalized words"), and another uses free-text contexts in which the noun phrase is found (e.g., "X frequently follows the bigram 'mayor of' "). Second, NELL is a bootstrapping system, which self-trains on its growing collection of confident beliefs.

1.2 Knowledge Base Inference: Horn Clauses

Although NELL has now grown a sizable knowledge base, its ability to perform inference over this knowledge base is currently very limited. At present 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

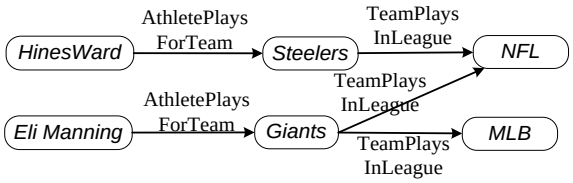

Figure 1: An example subgraph.

$$\begin{aligned} & \textit{AthletePlaysForTeam}(a,b) \\ \wedge \quad & \textit{TeamPlaysInLeague}(b,c) \\ \Rightarrow \quad & \textit{AthletePlaysInLeague(a,c)} \end{aligned} \tag{1}$$

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 1. NELL currently has 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 learns these Horn clause rules using a variant of the FOIL algorithm (Quinlan and Cameron-Jones, 1993), 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 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.

Learning first-order Horn clauses is computationally expensive—not only is the search space large, but some Horn clauses can be costly to evaluate (Cohen and Page, 1995). N-FOIL uses two tricks to improve its scalability. First, it assumes that the consequent predicate is functional—e.g., that

[1] NELL's current KB is available online at `http://rtw.ml.cmu.edu`.

each *Athlete* plays in at most one *League*. This means that explicit negative examples need not be provided (Zelle et al., 1995): 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" (Richards and Mooney, 1992) 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 (1). As in FOIL, 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

$$\hat{P} = (N_+ + m * prior)/(N_+ + N_- + m) \quad (2)$$

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.

1.3 Knowledge Base Inference: Graph Random Walks

In this paper, we consider an alternative approach, based on the Path Ranking Algorithm (PRA) of Lao and Cohen (2010b), described in detail below. PRA learns to *rank* graph nodes b relative to a query node a. PRA begins by enumerating a large set of bounded-length edge-labeled path types, similar to the initial clauses used in NELL's variant of FOIL. These path types are treated as ranking "experts", each performing a random walk through the graph, constrained to follow that sequence of edge types, and ranking nodes b by their weights in the resulting distribution. Finally, PRA combines the weights contributed by different "experts" using logistic regression to predict the probability that the relation $R(a, b)$ is satisfied.

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

$$\begin{aligned} & isa(a, c) \wedge isa^{-1}(c, a') \\ \wedge \quad & AthletePlaysInLeague(a', b) \\ \Rightarrow \quad & AthletePlaysInLeague(a, b) \end{aligned} \quad (3)$$

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 $\ell \in L$, let A_ℓ be the set of athletes in league ℓ: after three steps, the walk will have probability $|A_\ell|/|A|$ 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 players than hockey players over time, then the league rankings for the path of clause (3) 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"[2] as in Figure 1. Then the path for clause (1) above will give lower weight to $b =$ *NFL* for $a =$ *EliManning* than to $b =$ *NFL* for $a =$ *HinesWard*.

The main contribution of this paper is to 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:

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

- The evidence in support of inferring a relation instance $R(a, b)$ is based on many existing paths between a and b in the current KB, combined using 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 (since the paths used in the inference have these properties).
- 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 (Lao and Cohen, 2010a). The resulting inference is as fast as 10 milliseconds per query on average.

The Path Ranking Algorithm (PRA) we use is similar to that described elsewhere (Lao and Cohen, 2010b), except that to achieve efficient model learning, the paths between a and b are determined by the statistics from a population of training queries rather than enumerated completely. PRA uses random walks to generate relational features on graph data, and combine them with a logistic regression model. Compared to other relational models (e.g. FOIL, Markov Logic Networks), PRA is extremely efficient at link prediction or retrieval tasks, in which we are interested in identifying top links from a large number of candidates, instead of focusing on a particular node pair or joint inferences.

1.4 Related Work

The TextRunner system (Cafarella et al., 2006) 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 (Jain and Pantel, 2010) 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 (Schoenmackers et al., 2008) 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 (Richardson and Domingos, 2006) 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 (Lao and Cohen, 2010a).

Below we describe our approach in greater detail, provide experimental evidence of its value for performing inference in NELL's knowledge base, and discuss implications of this work and directions for future research.

2 Approach

In this section, we first describe how we formulate link (relation) prediction on a knowledge base as a ranking task. Then we review the Path Ranking Algorithm (PRA) introduced by Lao and Cohen (2010b; 2010a). After that, we describe two improvements to the PRA method to make it more suitable for the task of link prediction in knowledge bases. The first improvement helps PRA deal with the large number of relations typical of large knowledge bases. The second improvement aims at improving the quality of inference by applying low variance sampling.

2.1 Learning with NELL's Knowledge Base

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.

2.2 Path Ranking Algorithm Review

We now review the Path Ranking Algorithm introduced by Lao and Cohen (2010b). A *relation path* P is defined as a sequence of relations $R_1 \dots R_\ell$, and in order to emphasize the types associated with each step, P can also be written as $T_0 \xrightarrow{R_1} \dots \xrightarrow{R_\ell} T_\ell$, where $T_i = range(R_i) = domain(R_{i+1})$, and we also define $domain(P) \equiv T_0$, $range(P) \equiv T_\ell$. In the experiments in this paper, there is only one type of node which we call a *concept*, which can be connected through different types of relations. In this notation, relations like "the team a certain player plays for", and "the league a certain player's team is in" can be expressed by the paths below (respectively):

$$P_1: \quad concept \xrightarrow{AtheletePlayesForTeam} concept$$

$$P_2: \quad concept \xrightarrow{AtheletePlayesForTeam} concept \xrightarrow{TeamPlaysInLeague} concept$$

For any relation path $P = R_1 \dots R_\ell$ and a seed node $s \in domain(P)$, a *path constrained random walk* defines a distribution $h_{s,P}$ recursively as follows. If P is the empty path, then define

$$h_{s,P}(e) = \begin{cases} 1, & \text{if } e = s \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

If $P = R_1 \dots R_\ell$ is nonempty, then let $P' = R_1 \dots R_{\ell-1}$, and define

$$h_{s,P}(e) = \sum_{e' \in range(P')} h_{s,P'}(e') \cdot P(e|e'; R_\ell), \tag{5}$$

where $P(e|e'; R_\ell) = \frac{R_\ell(e',e)}{|R_\ell(e',\cdot)|}$ is the probability of reaching node e from node e' with a one step random walk with edge type R_ℓ. $R(e', e)$ indicates whether there exists an edge with type R that connect e' to e.

More generally, given a set of paths $P_1, \dots, P_n$, one could treat each $h_{s,P_i}(e)$ as a *path feature* for the node e, and rank nodes by a linear model

$$\theta_1 h_{s,P_1}(e) + \theta_2 h_{s,P_2}(e) + \dots \theta_n h_{s,P_n}(e)$$

where θ_i are appropriate weights for the paths. This gives a ranking of nodes e related to the query node s by the following scoring function

$$score(e; s) = \sum_{P \in \mathbf{P}_\ell} h_{s,P}(e)\theta_P, \tag{6}$$

where $\mathbf{P}_\ell$ is the set of relation paths with length $\leq \ell$.

Given a relation R and a set of node pairs $\{(s_i, t_i)\}$ for which we know whether $R(s_i, t_i)$ is true or not, we can construct a training dataset $\mathbf{D} = \{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i$ is a vector of all the path features for the pair (s_i, t_i)—i.e., the j-th component of $\mathbf{x}_i$ is $h_{s_i,P_j}(t_i)$, and where y_i is a boolean variable indicating whether $R(s_i, t_i)$ is true. We then train a logistic function to predict the conditional probability $P(y|\mathbf{x}; \theta)$. The parameter vector θ is estimated by maximizing a regularized form of the conditional likelihood of y given $\mathbf{x}$. In particular, we maximize the objective function

$$O(\theta) = \sum_i o_i(\theta) - \lambda_1|\theta|_1 - \lambda_2|\theta|_2, \tag{7}$$

where λ_1 controls L_1-regularization to help structure selection, and λ_2 controls L_2-regularization to prevent overfitting. $o_i(\theta)$ is the per-instance weighted log conditional likelihood given by

$$o_i(\theta) = w_i[y_i \ln p_i + (1 - y_i) \ln(1 - p_i)], \tag{8}$$

where p_i is the predicted probability $p(y_i = 1|\mathbf{x}_i; \theta) = \frac{\exp(\theta^T \mathbf{x}_i)}{1+\exp(\theta^T \mathbf{x}_i)}$, and w_i is an importance weight to each example. A biased sampling procedure selects only a small subset of negative samples to be included in the objective (see (Lao and Cohen, 2010b) for detail).

2.3 Data-Driven Path Finding

In prior work with PRA, $\mathbf{P}_\ell$ was defined as all relation paths of length at most ℓ. When the number of edge types is small, one can generate $\mathbf{P}_\ell$ by

Table 1: Number of paths in PRA models of maximum path length 3 and 4. Averaged over 96 tasks.

	ℓ=3	ℓ=4
all paths up to length L	$15,376$	$1,906,624$
+query support$\geq \alpha = 0.01$	522	5016
+ever reach a target entity	136	792
+L_1 regularization	63	271

Table 2: Comparing PRA with RWR models. MRRs and training times are averaged over 96 tasks.

	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

enumeration; however, for domains with a large number of edge types (e.g., a knowledge base), it is impractical to enumerate all possible relation paths even for small ℓ. For instance, if the number of edge types related to each node type is 100, even the number of length three paths types easily reaches millions. For other domains like parsed natural language sentences, useful relation paths can be as long as ten relations (Minkov and Cohen, 2008). In this case, even with smaller number of possible edge types, the total number of relation paths is still too large for systematic enumeration.

In order to apply PRA to these domains, we modify the path generation procedure in PRA to produce only relation paths which are potentially useful for the task. 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 be supported by at least a fraction α of the training queries s_i, as well as being of length no more than ℓ (In the experiments, we set $\alpha = 0.01$) We also require that in order for a relation path to be included in the PRA model, it must retrieve at least one target entity t_i in the training set. As we can see from Table 1, 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.

The idea of finding paths that connects nodes in a graph is not new. It has been embodied previously in first-order learning systems (Richards and Mooney, 1992) as well as N-FOIL, and relational database searching systems (Bhalotia et al., 2002). These approaches consider a single query during path finding. In comparison, the data-driven path finding method we described here uses statistics from a population of queries, and therefore can potentially determine the importance of a path more reliably.

2.4 Low-Variance Sampling

Lao and Cohen (2010a) previously showed 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) (Thrun et al., 2005), 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.

3 Results

This section reports 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.

3.1 Cross Validation on the Training Queries

Random Walk with Restart (RWR) (also called personalized PageRank (Haveliwala, 2002)) is a general-purpose graph proximity measure which has been shown to be fairly successful for many types of tasks. We compare PRA to two versions of RWR on the 96 tasks of link prediction with NELL's knowledge base. The two baseline methods are an untrained RWR model and a trained RWR model as described by Lao and Cohen (2010b). (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 a loss function, such as Equation 7. In the untrained model, edge weights are uniform.) We explored a range of values for the regularization parameters L_1 and L_2 using cross validation on the training data, and we fix both L_1 and L_2 parameters to 0.001 for all tasks. The maximum path length is fixed to 3.[3]

Table 2 compares the three methods using 5-fold cross validation and the Mean Reciprocal Rank (MRR)[4] measure, which is defined as the inverse rank of the highest ranked relevant result in a set of results. If the the first returned result is relevant, then MRR is 1.0, otherwise, it is smaller than 1.0. Supervised training can significantly improve retrieval quality (p-value=9×10^{-8} comparing untrained and trained RWR), and leveraging path information can produce further improvement (p-value=4×10^{-4} comparing trained RWR with PRA). The average training time for a predicate is only a few seconds.

We also investigate the effect of low-variance sampling on the quality of prediction. Figure 2 compares independent and low variance sampling when applied to finger printing and particle filtering (Lao and Cohen, 2010a). 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 on the training query set. Low-variance

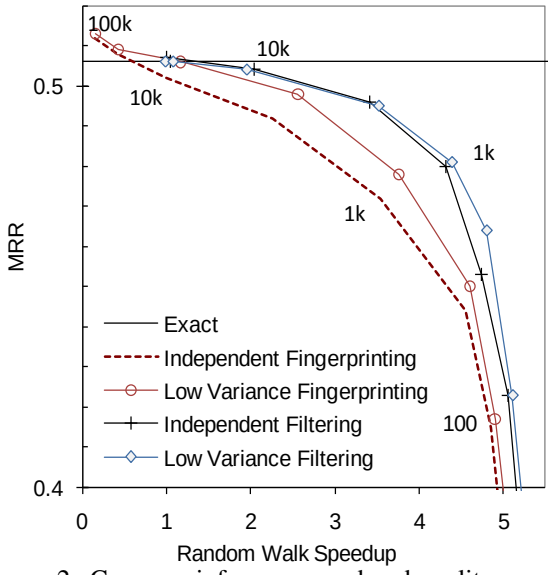

Figure 2: Compare inference speed and quality over 96 tasks. The speedup is relative to exact inference, which is on average 23ms per query.

sampling can improve prediction for both finger printing and particle filtering. 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. Interestingly, when using a large number of walkers, the finger printing methods produce even better prediction quality than exact inference. Lao and Cohen noticed a similar improvement on retrieval tasks, and conjectured that it is because the sampling inference imposes a regularization penalty on longer relation paths (2010a).

3.2 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 compare the quality of PRA to N-FOIL on these 8 predicates only. Among all the 72 non-functional predicates—which

[3] Results with maximum length 4 are not reported here. Generally models with length 4 paths produce slightly better results, but are 4-5 times slower to train

[4] For a set of queries Q, $MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank of the first correct answer for } q}$

Table 3: The top two weighted PRA paths for tasks on which N-FOIL discovers confident rules. c stands for *concept*.

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

Table 4: 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	$P_{majority}$	**#Paths**	**p@10**	**p@100**	**p@1000**	**#Rules**	**#Query**	**p@10**	**p@100**	**p@1000**
athletePlaysForTeam	0.07	125	0.4	0.46	0.66	1(+1)	7	0.6	0.08	0.01
athletePlaysInLeague	0.60	15	1.0	0.84	0.80	3(+30)	332	0.9	0.80	0.24
athletePlaysSport	0.73	34	1.0	0.78	0.70	2(+30)	224	1.0	0.82	0.18
stadiumLocatedInCity	0.05	18	0.9	0.62	0.54	1(+0)	25	0.7	0.16	0.00
teamHomeStadium	0.02	66	0.3	0.48	0.34	1(+0)	2	0.2	0.02	0.00
teamPlaysInCity	0.10	29	1.0	0.86	0.62	1(+0)	60	0.9	0.56	0.06
teamPlaysInLeague	0.26	36	1.0	0.70	0.64	4(+151)	30	0.9	0.18	0.02
teamPlaysSport	0.42	21	0.7	0.60	0.62	4(+86)	48	0.9	0.42	0.02
average	0.28	43	0.79	0.668	0.615		91	0.76	0.38	0.07
teamMember	0.01	203	0.8	0.64	0.48					
companiesHeadquarteredIn	0.05	42	0.6	0.54	0.60					
publicationJournalist	0.02	25	0.7	0.70	0.64					
producedBy	0.19	13	0.5	0.58	0.68	N-FOIL does not produce results				
competesWith	0.19	74	0.6	0.56	0.72	for non-functional predicates				
hasOfficeInCity	0.03	262	0.9	0.84	0.60					
teamWonTrophy	0.24	56	0.5	0.50	0.46					
worksFor	0.13	62	0.6	0.60	0.74					
average	0.11	92	0.650	0.620	0.615					

N-FOIL cannot be applied to—PRA exhibits a wide range of performance in cross-validation. The 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.

Table 3 shows the top two weighted PRA features for each task on which N-FOIL can successfully learn rules. These PRA rules can be categorized into broad coverage rules which behave like priors over correct 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) 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.

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. (6). For N-FOIL, the candidates are sorted by the estimated accuracies of the rules as in Eq. (2) (which generate the candidates). 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 first sort the queries s for each predicate R by the scores of their top ranked candidate t in descending order, and then calculate precisions at top 10, 100 and 1000 positions for the list of result $R(s^{R,1}, t_1^{R,1}), R(s^{R,2}, t_1^{R,2}), ...$, where $s^{R,1}$ is the first query for predicate R, $t_1^{R,1}$ is its first candidate, $s^{R,2}$ is the second query for predicate R, $t_1^{R,2}$ is its first candidate, so on and so forth. To reduce the labeling load, we judge all top 10 queries 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 remove some workers' judgments which are obviously incorrect[5]. We sampled 100 beliefs, and compared their voted result to gold-standard labels produced by one author of this paper. Table 5 shows that 74% of the time the workers' voted result agrees with our judgement.

Table 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%

Table 4 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. Predicting the functional predicates is generally easier predicting the non-functional predicates. The #Query column shows the number of queries on which N-FOIL is able to match any of its rules, and hence produce a candidate belief. For most predicates, N-FOIL is only able to produce results for at most a few hundred queries. 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 10 and at 1000 (p@100 and p@1000) are much lower[6].

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

[5]Certain workers label all the questions with the same answer

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

specialized rules. Generally, specialized rules have much smaller 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. However, learning more accurate specialized paths is part of our future work.

A significant advantage of PRA over N-FOIL is that it can be applied to non-functional predicates. The last eight rows of Table 4 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.

4 Conclusions and Future Work

We have shown that a soft inference procedure based on a combination of constrained, weighted, random walks through the knowledge base graph 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.

There are several prominent directions for future work. First, inference starting from both the query nodes and target nodes (Richards and Mooney, 1992) can be much more efficient in discovering long paths than just inference from the query nodes. Second, inference starting from the target nodes of training queries is a potential way to discover specialized paths (with grounded nodes). Third, generalizing inference paths to inference trees or graphs can produce more expressive random walk inference models. Overall, we believe that random walk is a promising way to scale up relational learning to domains with very large data sets.

Acknowledgments

This work was supported by NIH under grant R01GM081293, by NSF under grant IIS0811562, by DARPA under awards FA8750-08-1-0009 and AF8750-09-C-0179, and by a gift from Google. We thank Geoffrey J. Gordon for the suggestion of applying low variance sampling to random walk inference. We also thank Bryan Kisiel for help with the NELL system.

References

Eugene Agichtein and Luis Gravano. 2000. Snowball: extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries - DL '00*, pages 85–94, New York, New York, USA. ACM Press.

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

Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. 2002. Keyword searching and browsing in databases using banks. *ICDE*, pages 431–440.

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory - COLT' 98*, pages 92–100, New York, New York, USA. ACM Press.

MJ Cafarella, M Banko, and O Etzioni. 2006. Relational Web Search. In *WWW*.

Jamie Callan and Mark Hoy. 2009. Clueweb09 data set. *http://boston.lti.cs.cmu.edu/Data/clueweb09/*.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M.

Mitchell. 2010. Toward an Architecture for Never-Ending Language Learning. In *AAAI*.

William W. Cohen and David Page. 1995. Polynomial learnability and inductive logic programming: Methods and results. *New Generation Comput.*, 13(3&4):369–409.

Oren Etzioni, Michael Cafarella, Doug Downey, Anamaria Popescu, Tal Shaked, Stephen Soderl, Daniel S. Weld, and Er Yates. 2005. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165:91–134.

Taher H. Haveliwala. 2002. Topic-sensitive pagerank. In *WWW*, pages 517–526.

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

Ni Lao and William W. Cohen. 2010a. Fast query execution for retrieval models based on path-constrained random walks. *KDD*.

Ni Lao and William W. Cohen. 2010b. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*.

Einat Minkov and William W. Cohen. 2008. Learning graph walk based similarity measures for parsed text. *EMNLP*, pages 907–916.

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

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

Bradley L. Richards and Raymond J. Mooney. 1992. Learning relations by pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 50–55, San Jose, CA, July.

Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Machine Learning*.

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

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

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

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

John M. Zelle, Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney. 1995. 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.