Programming with Personalized PageRank: A Locally Groundable First-Order Probabilistic Logic

William Yang Wang YWW@CS.CMU.EDU
Language Technology Institute, Carnegie Mellon University, Pittsburgh, PA 15213

Kathryn Mazaitis KRIVARD@CS.CMU.EDU
William W. Cohen WCOHEN@CS.CMU.EDU
Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

In many probabilistic first-order representation systems, inference is performed by "grounding"—i.e., mapping it to a propositional representation, and then performing propositional inference. With a large database of facts, groundings can be very large, making inference and learning computationally expensive. Here we present a first-order probabilistic language which is well-suited to approximate "local" grounding: every query Q can be approximately grounded with a small graph. The language is an extension of stochastic logic programs where inference is performed by a variant of personalized PageRank. Experimentally, we show that the approach performs well without weight learning on an entity resolution task; that supervised weight-learning improves accuracy; and that grounding time is independent of DB size. We also show how our approach can be used for joint inference in a statistical relational learning task.

1. INTRODUCTION

In many probabilistic first-order representation systems, including Markov Logic Networks (Richardson & Domingos, 2006) and Probabilistic Similarity Logic (Brocheler et al., 2012), inference is performed by mapping a first-order program to a propositional representation, and performing inference in that propositional representation. This mapping is often called *grounding*. For example, Figure 1 shows a simple MLN.[1] As is often the case, this MLN has two parts: the *rules* R_1, R_2, which are weighted first-order clauses; and the *database* DB, which consists of facts (unit clauses) of the form *links(a,b)* for constants a, b. The figure also shows the the grounded version of this MLN, which is an ordinary Markov network: the DB facts become constraints on node values, and the clauses become clique potentials.

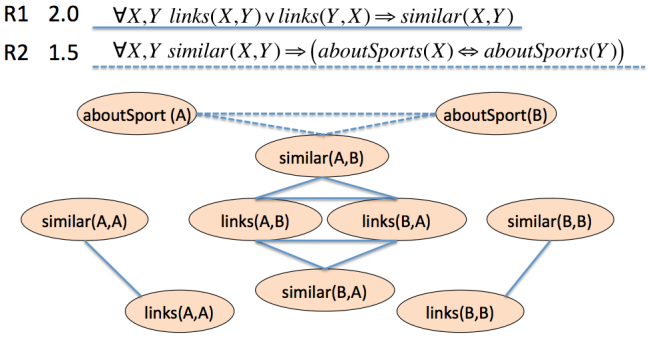

Figure 1. A Markov logic network program and its grounding. (Dotted lines are clique potentials associated with rule R1, solid lines with rule R2.)

Grounding a first-order program can be an expensive operation. For a realistic hyperlink graph, a Markov network with size even linear in the number of facts in the database, $|DB|$, is impractically large for inference. Superficially, it would seem that groundings must inherit be $o(|DB|)$ for some programs: in the example, for instance, the probability of *aboutSport(x)* must depends to some extent on the entire hyperlink graph (if it is fully connected). However, it also seems intuitive that if we are interested in inferring information about a specific page—say, the probability of

Presented at the International Conference on Machine Learning (ICML) workshop on *Inferning: Interactions between Inference and Learning*, Atlanta, Georgia, USA, 2013.

[1] This MLN does a very simple sort of label-propagation through hyperlinks.

aboutSport(d1)–then the parts of the network only distantly connected to *d1* are likely to have a small influence. This suggests that an *approximate* grounding strategy might be feasible, in which a query such as *aboutSport(d1)* would be grounded by constructing a small subgraph of the full network, followed by inference on this small "locally grounded" subgraph. Likewise, consider learning (e.g., from a set of queries Q with their desired truth values). Learning might proceed by locally-grounding every query goal, allowing learning to also take less than $O(|DB|)$ time.

In this paper, we present a first-order probabilistic language which is well-suited to approximate "local" grounding. We present an extension to *stochastic logic programs* (SLP) (Cussens, 2001) that is biased towards short derivations, and show that this is related to *personalized PageRank* (PPR) (Page et al., 1998; Chakrabarti, 2007) on a linearized version of the proof space. Based on the connection to PPR, we develop a proveably-correct approximate inference scheme, and an associated proveably-correct approximate grounding scheme: specifically, we show that it is possible to prove a query, or to build a graph which contains the information necessary for weight-learning, in time $O(\frac{1}{\alpha\epsilon})$, where α is a reset parameter associated with the bias towards short derivations, and ϵ is the worst-case approximation error across all intermediate stages of the proof. This means that both inference and learning can be approximated in time *independent of the size of the underlying database*—a surprising and important result.

The ability to locally ground queries has another important consequence: it is possible to decompose the problem of weight-learning to a number of moderate-size subtasks (in fact, tasks of size $O(\frac{1}{\alpha\epsilon})$ or less) which are weakly coupled. Based on this we outline a parallelization scheme, which in our initial implementation provides a order-of-magnitude speedup in learning time.

Below, we will first introduce our formalism, and then describe our weight-learning algorithm. We will then present experimental results on a prototypical inference task, and compare the scalability of our method to Markov logic networks. We finally discuss related work and conclude.

2. PROPPR

2.1. LOGIC PROGRAM INFERENCE AS GRAPH SEARCH

We will now describe our "locally groundable" first-order probabilistic language, which we call ProPPR.[2] Inference for ProPPR is based on a personalized PageRank process over the proof constructed by Prolog's Selective Linear Definite (SLD) theorem-prover. To define the semantics we will use notation from logic programming (Lloyd). Let LP be a program which contains a set of definite clauses $c_1, \ldots, c_n$, and consider a conjunctive query Q over the predicates appearing in LP. A traditional Prolog interpreter can be viewed as having the following actions. First, construct a "root vertex" v_0 which is a pair (Q, Q) and add it to an otherwise-empty graph $G'_{Q,LP}$. (For brevity, we will use drop the subscripts of G' where possible.) Then recursively add to G' new vertices and edges as follows: if u is a vertex of the form $(Q, (R_1, \ldots, R_k))$, and c is a clause in LP of the form $R' \leftarrow S'_1, \ldots, S'_\ell$, and R_1 and R' have a most general unifier $\theta = mgu(R_1, R')$, then add to G' a new edge $u \rightarrow v$ where $v = (Q\theta, (S'_1, \ldots, S'_\ell, R2, \ldots, R_k)\theta)$. Let us call $Q\theta$ the *transformed query* and $(S'_1, \ldots, S'_\ell, R2, \ldots, R_k)\theta$ the *associated subgoal list*.

Table 1. A simple program in ProPPR. See text for explanation.

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

G' is often large or infinite so it is not constructed explicitly. Instead Prolog performs a depth-first search on G' to find the first *solution vertex* v—i.e., a vertex with an empty subgoal list—and if one is found, returns the transformed query from v as an answer to Q. Table 1 and Figure 2 show a simple Prolog program and a proof graph for it.[3] Given the query Q = *about(a,Z)*, Prolog's depth-first search would return Q = *about(a,fashion)*.

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

2.2. FROM STOCHASTIC LOGIC PROGRAMS TO PROPPR

In *stochastic logic programs* (SLPs) (Cussens, 2001), one defines a randomized procedure for traversing the

[2] For Programming with Personalized PageRank.

[3] The annotations after the hashmarks and the edge labels in the proof graph will be described below. For conciseness, only $R_1, \ldots, R_k$ is shown in each node $u = (Q, (R_1, \ldots, R_k))$.

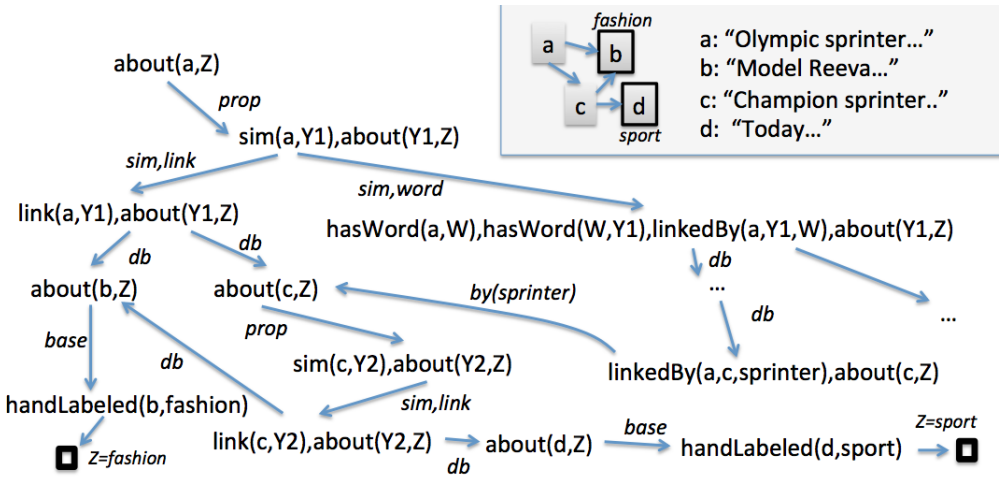

Figure 2. A partial proof graph for the query *about(a,Z)*. The upper right shows the link structure between documents a, b, c, and d, and some of the words in the documents. Restart links are not shown.

graph G' which thus defines a probability distribution over vertices v, and hence (by selecting only solution vertices) a distribution over transformed queries (i.e. answers) $Q\theta$. The randomized procedure thus produces a distribution over possible answers, which can be tuned by learning to upweight desired (correct) answers and downweight others.

In past work, the randomized traversal of G' was defined by a probabilistic choice, at each node, of which clause to apply, based on a weight for each clause. We propose two extensions. First, we will introduce a new way of computing clause weights, which allows for a potentially richer parameterization of the traversal process. We will associate with each edge $u \rightarrow v$ in the graph a *feature vector* $\phi_{u\rightarrow v}$. This edge is produced indirectly, by associating with every clause $c \in LP$ a function $\Phi_c(\theta)$, which produces the ϕ associated with an application of c using mgu θ. This feature vector is computed during theorem-proving, and used to annotate the edge $u \rightarrow v$ in G' created by applying c with mgu θ. Finally, an edge $u \rightarrow v$ will be traversed with probability $\Pr(v|u) \propto f(\mathbf{w}, \phi_{u\rightarrow v})$ where $\mathbf{w}$ is a parameter vector and where $f(\mathbf{w}, \phi)$ is a weighting function—e.g., $f(\mathbf{w}, \phi) = \exp(\mathbf{w}_i \cdot \phi)$. This weighting function now determines the probability of a transition, in theorem-proving, from u to v: specifically, $\Pr_{\mathbf{w}}(v|u) \propto f(\mathbf{w}, \phi_{u\rightarrow v})$. Weights in $\mathbf{w}$ default to 1.0, and learning consists of tuning these.

The second and more fundamental extension is to add edges in G' from every solution vertex to itself, and also add an edge from every vertex to the start vertex v_0. We will call this augmented graph $G_{Q,LP}$ below (or just G if the subscripts are clear from context). These links make SLP's graph traversal a *personalized PageRank* (PPR) procedure, sometimes known as *random-walk-with-restart* (Tong et al., 2006). These links are annotated by another feature vector function $\Phi_{restart}(R)$, which is a applied of the leftmost literal of the subgoal list for u to annotate the edge $u \rightarrow v$.

The results of *short proofs* are upweighted by these links back to the start vertex, due to the bias from the traversal of the proof graph. To see this, note that if the restart probability $P(v_0|u) = \alpha$ for every node u, then the probability of reaching any node at depth d is bounded by $(1-\alpha)^d$.

2.3. LOCALLY GROUNDING A QUERY

Note that this procedure both performs inference (by computing a distribution over literals $Q\theta$) and "grounds" the query, by constructing a graph G. ProPPR inference for this query can be re-done efficiently, by running an ordinary PPR process on G. This is useful for faster weight learning. Unfortunately, the grounding G can be very large: it need not include the entire database, but if T is the number of iterations until convergence for the sample program of Table 1 on the query $Q = about(d, Y)$, G will include a node for every page within T hyperlinks of d.

To construct a more compact local grounding graph G, we adapt an approximate personalized PageRank method called PageRank-Nibble (Andersen et al., 2008). This method has been used for the problem of *local partitioning*: in local partitioning, the goal is to find a small, low-conductance[4] component of a large graph G that contains a given node v.

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

Following the proof technique of Andersen et al, it can be shown that after each push, $\mathbf{p} + \mathbf{r} = \mathbf{ppr}(v_0)$. It is also clear than when PageRank-Nibble terminates, then for any u, the error $\mathbf{ppr}(v_0)[u] - \mathbf{p}[u]$ is bounded by $\epsilon N(u)$: the number of edges in the graph $\hat{G}$ produced by PageRank-Nibble-Prove is no more than $\frac{1}{\alpha'\epsilon}$.

Importantly, the bound holds *independent of the size of the full database of facts*. The bound also holds regardless of the size or loopiness of the full proof graph, so this inference procedure will work for recursive logic programs.

To summarize, we have outlined an efficient approximate proof procedure, which is closely related to personalized PageRank. As a side-effect of inference for a query Q, this procedure will create a ground graph $\hat{G}_Q$ on which personalized PageRank can be run directly, without any (relatively expensive) manipulation of first-order theorem-proving constructs such as clauses or logical variables. As we will see, this "locally grounded" graph will be very useful in learning weights $\mathbf{w}$ to assign to the features of a ProPPR program.

As an illustration of the sorts of ProPPR programs that are possible, some small sample programs are shown in Figure 2. Clauses c_1 and c_2 are, together, a bag-of-words classifier: each proof of *predictedClass(D,Y)* adds some evidence for D having class Y, with the weight of this evidence depending on the weight given to c_2's use in establishing *related(w,y)*, where w and y are a specific word in D and y is a possible class label. In turn, c_2's weight depends on the weight assigned to the $r(w, y)$ feature by $\mathbf{w}$, relative to the weight of the restart link.[5] Adding c_3 and c_4 to this program implements label propagation, and adding c_5 and c_6 implements a sequential classifier.

In spite of its efficient inference procedure, and its limitation to only definite clauses, ProPPR appears to have much of the expressive power of MLNs (Domingos & Lowd, 2009), in that many useful heuristics can apparently be encoded.

2.4. LEARNING FOR PROPPR

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

In prior work, Backstrom and Leskovic (Backstrom et al., 2006) outlined a family of supervised learning procedures for this sort of annotated graph. To optimize this loss, we use stochastic gradient descent (SGD), rather than the quasi-Newton method of Backstrom and Leskovic.

We implemented SGD because it is fast and has been adapted to parallel learning tasks (Zinkevich et al., 2010; Niu et al., 2011b). Local grounding means that learning for ProPPR is quite well-suited to parallelization. The step of locally grounding each Q_i is "embarassingly" parallel, as every grounding can be done independently. To parallelize the weight-learning stage, we use multiple threads, each of which computes the gradient over a single grounding $\hat{G}_{Q^k}$, and all of which accesses a single shared parameter vector $\mathbf{w}$. Although the shared parameter vector is a potential bottleneck (Zinkevich et al., 2009), it is not a severe one, as the gradient computation dominates the learning cost.[6]

3. EXPERIMENTS

3.1. Entity Resolution

To evaluate this method, we use data from an entity resolution task previously studied as a test case for MLNs (Singla & Domingos, 2006a). The program we used is approximately the same as the MLN(B+T) approach from Singla and Domingos.[7] To evaluate accuracy, we use the CORA dataset, a collection of 1295 bibliography citations that refer to 132 distinct papers. Throughout the experiments, we set the regularization coefficient μ to 0.001, the total number of epochs to 5, and learning rate parameter η to 1. A standard log loss function was used in our objective function.

While the speedup in inference time is desirable, the more important advantages of the local grounding approach are that (1) grounding time, and hence inference, need not grow with the database size and (2) learning can be performed in parallel, by using multiple threads for parallel computations of gradients in SGD. Figure 3 illustrates the first of these points: the scalability of the PageRank-Nibble-Prove method as database size increases. For comparison, we also show the inference time for MLNs with three well-published inference methods: Gibbs refers to Gibbs sampling, and Lifted BP is the lifted belief propagation method. We also compare with the maximum a posteriori (MAP) inference approach, which does not

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

[6] This is not the case when learning a linear classifier, where gradient computations are much cheaper.

[7] The principle difference is that we do not include tests on the absence of words in a field in our clauses.

Table 2. Some more sample ProPPR programs. $LP = \{c_1, c_2\}$ is a bag-of-words classifier (see text). $LP = \{c_1, c_2, c_3, c_4\}$ is a recursive label-propagation scheme, in which predicted labels for one document are assigned to similar documents, with similarity being an (untrained) cosine distance-like measure. $LP = \{c_1, c_2, c_5, c_6\}$ is a sequential classifier for document sequences.

c_1: predictedClass(Doc,Y) :-
possibleClass(Y),
hasWord(Doc,W),
related(W,Y) # c1.
c_2: related(W,Y) :- true
relatedFeature(W,Y)

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

c_3: predictedClass(Doc,Y) :-
similar(Doc,OtherDoc),
predictedClass(OtherDoc,Y) # c3.
c_4 : similar(Doc1,Doc2) :-
hasWord(Doc1,W),
inDoc(W,Doc2) # c4.

c_5 : predictedClass(Doc,Y) :-
previous(Doc,OtherDoc),
predictedClass(OtherDoc,OtherY),
transition(OtherY,Y) # c5.
c_6: transition(Y1,Y2) :- true
transitionFeature(Y1,Y2)

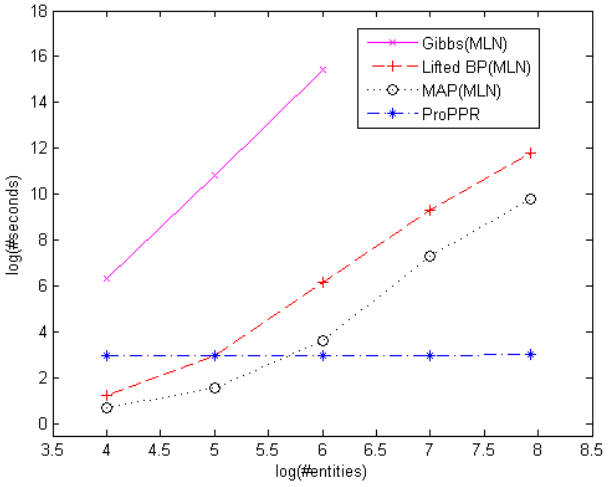

Figure 3. Run-time for inference in ProPPR (with a single thread) as a function of the number of entities in the database. The base of the log is 2.

return probabilistic estimates of the specified queries. In each case the performance task is inference over 16 test queries.

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

The results in Table 3 all use the same data and evaluation procedure, and the MLNs were trained with the state-of-the-art Alchemy system using the recommended commands for this data (which is distributed with Alchemy[8]). However, we should note that the MLN results reproduced here are not identical to previous-reported ones (Singla & Domingos, 2006a). Singla and Domingos used a number of complex heuristics that are difficult to reproduce—e.g., one of these was combining MLNs with a heuristic, TFIDF-based matching procedure based on canopies (McCallum et al., 2000). While the trained ProPPR model outperformed the reproduced MLN model in all prediction tasks, it outperforms the reported results from Singla and Domingos only on *venue*, and does less well than the reported results on *citation* and *author*[9].

Table 3. AUC results on CORA citation-matching.

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

3.2. Joint Inference for Link Prediction

In addition to the entity resolution task, we also investigate our approach for joint inference in a link (relation) prediction problem. To do this, we use a subset of 19,527 beliefs from a knowledge base, which is extracted imperfectly from the web by NELL, a never-ending language learner (Carlson et al., 2010). The training set contains 12,331 queries, and the test

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

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

Table 4. AUC results on the NELL link prediction task.

	AUC
ProPPR(Non-recursive PRA rules)	0.858
ProPPR(Recursive PRA rules)	**0.916**

set contains 1,185 queries. In contrast to a previous approach (Lao et al., 2011) for link prediction, we combine the learned top-ranked paths (Lao & Cohen, 2010) from the NELL relational network, and tranform these paths as ProPPR programs, then perform joint inference to predict the link between entities. In this experiment, our goal is to answer the following question: can we use ProPPR to perform joint inference to improve the performance of a link prediction task on a relational knowledge base?

The results on the NELL link-prediction task is shown in Table 4. We observe that when not performing joint inference with the learned top-ranked paths, ProPPR obtains an AUC of 0.858. However, when using these rules for joint learning, we observe an AUC of 0.916 when using the ProPPR. The total time for joint inference with 797 rules takes only 13 minutes.

4. RELATED WORK

Although we have chosen here to compare experimentally to MLNs (Richardson & Domingos, 2006; Singla & Domingos, 2006a), ProPPR represents a rather different philosophy toward language design: rather than beginning with a highly-expressive but intractible logical core, we begin with a limited logical inference scheme and add to it a minimal set of extensions that allow probabilistic reasoning, while maintaining stable, efficient inference and learning. While ProPPR is less expressive than MLNs (for instance, it is limited to definite clause theories) it is also much more efficient. This philosophy is similar to that illustrated by probabilistic similarity logic (PSL) (Brocheler et al., 2012); however, unlike ProPPR, PSL does not include a "local" grounding procedure, which leads to small inference problems, even for large databases.

Technically, ProPPR is most similar to stochastic logic programs (SLPs) (Cussens, 2001). The key innovation is the integration of a restart into the random-walk process, which, as we have seen, leads to very different computational properties. More broadly speaking, ProPPR also relates to the prior work on probabilistic theorem proving (e.g. (Gogate & Domingos, 2011)).

There has been some prior work on reducing the cost of grounding probabilistic logics: noteably, Shavlik et al (Shavlik & Natarajan, 2009) describe a preprocessing algorithm called FROG that uses various heuristics to greatly reduce grounding size and inference cost, and Niu et al (Niu et al., 2011a) describe a more efficient bottom-up grounding procedure that uses an RDBMS. Other methods that reduce grounding cost and memory usage include "lifted" inference methods (e.g., (Singla & Domingos, 2008; Van den Broeck et al., 2011; 2012; Venugopal & Gogate, 2012)) and "lazy" inference methods (e.g., (Singla & Domingos, 2006b)); in fact, the LazySAT inference scheme for Markov networks is broadly similar algorithmically to PageRank-Nibble-Prove, in that it incrementally extends a network in the course of theorem-proving. However, there is no theoretical analysis of the complexity of these methods, and experiments with FROG and LazySAT suggest that they still lead to a groundings that grow with DB size, albeit more slowly. Our work also aligns with the lifted personalized PageRank approach (Ahmadi et al., 2011), which constructs a network of lifted supernodes and superpotentials of indistinguishable nodes to improve the efficiency of inference in graphical models.

ProPPR is also closely related to the Path Ranking Algorithm (PRA), learning algorithm for link prediction (Lao & Cohen, 2010). Like ProPPR, PRA uses random-walk methods to approximate logical inference. However, the set of "inference rules" learned by PRA corresponds roughly to a logic program in a particular form—namely, the form

$$
\begin{aligned}
&p(S,T) \leftarrow r_{1,1}S, X_1), \ldots, r_{1,k_1}(X_{k_1-1}, T).\\
&p(S,T) \leftarrow r_{2,1}(S, X_1), \ldots, r_{2,k_2}(X_{k_2-1}, T).\\
&\vdots
\end{aligned}
$$

ProPPR allows much more general logic programs. However, unlike PRA, we do not consider the task of searching for new logic program clauses.

5. CONCLUSIONS

We described a new probabilistic first-order language which is designed with the goal of highly efficient inference and rapid learning. ProPPR takes Prolog's SLD theorem-proving, extends it with a probabilistic proof procedure, and then limits this procedure further, by including a "restart" step which biases the system to short proofs. This means that ProPPR has a simple polynomial-time proof procedure, based on the well-studied personalized PageRank (PPR) method.

Following prior work on PPR-like methods, we designed a local grounding procedure for ProPPR, based on local partitioning methods (Andersen et al., 2008), which leads to an inference scheme that is an order of magnitude faster that the conventional power-iteration approach to computing PPR, takes time $O(\frac{1}{\epsilon\alpha'})$, independent of database size. This ability to "locally ground" a query also makes it possible to partition the weight learning task into many separate gradient computations, one for each training example, leading

to a weight-learning method that can be easily parallelized. Experimentally, we showed that ProPPR performs well, even without weight learning, on an entity resolution task, and that supervised weight-learning improves accuracy. We also investigated how ProPPR can be used to perform joint inference and improve the performance in a link prediction task. A longer version of this extended abstract can be found on arXiv (Wang et al., 2013).

References

Ahmadi, Babak, Kersting, Kristian, and Sanner, Scott. Multi-evidence lifted message passing, with application to pagerank and the kalman filter. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, pp. 1152–1158. AAAI Press, 2011.

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

Backstrom, Lars, Huttenlocher, Dan, Kleinberg, Jon, and Lan, Xiangyang. Group formation in large social networks: membership, growth, and evolution. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 44–54, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5. doi: http://doi.acm.org/10.1145/1150402.1150412.

Brocheler, Matthias, Mihalkova, Lilyana, and Getoor, Lise. Probabilistic similarity logic. *arXiv preprint arXiv:1203.3469*, 2012.

Carlson, Andrew, Betteridge, Justin, Kisiel, Bryan, Settles, Burr, Jr., Estevam R. Hruschka, and Mitchell, Tom M. Toward an architecture for never-ending language learning. In Fox, Maria and Poole, David (eds.), *AAAI*. AAAI Press, 2010.

Chakrabarti, Soumen. Dynamic personalized PageRank in entity-relation graphs. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pp. 571–580, New York, NY, USA, 2007. ACM Press. ISBN 978-1-59593-654-7. doi: http://doi.acm.org/10.1145/1242572.1242650.

Cussens, James. Parameter estimation in stochastic logic programs. *Machine Learning*, 44(3):245–271, 2001.

Domingos, Pedro and Lowd, Daniel. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009.

Gogate, Vibhav and Domingos, Pedro. Probabilistic theorem proving. *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2011.

Lao, Ni and Cohen, William W. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 81(1):53–67, 2010.

Lao, Ni, Mitchell, Tom M., and Cohen, William W. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, pp. 529–539. ACL, 2011. ISBN 978-1-937284-11-4.

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

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

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

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

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

Richardson, Matthew and Domingos, Pedro. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, 2006. ISSN 0885-6125. doi: http://dx.doi.org/10.1007/s10994-006-5833-1.

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

Singla, Parag and Domingos, Pedro. Entity resolution with markov logic. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pp. 572–582. IEEE, 2006a.

Singla, Parag and Domingos, Pedro. Memory-efficient inference in relational domains. In *Proceedings of the national conference on Artificial intelligence*, volume 21, pp. 488. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006b.

Singla, Parag and Domingos, Pedro. Lifted first-order belief propagation. In *Proceedings of the 23rd national conference on Artificial intelligence*, volume 2, pp. 1094–1099, 2008.

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

Van den Broeck, Guy, Taghipour, Nima, Meert, Wannes, Davis, Jesse, and De Raedt, Luc. Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pp. 2178–2185. AAAI Press, 2011.

Van den Broeck, Guy, Choi, Arthur, and Darwiche, Adnan. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2012.

Venugopal, Deepak and Gogate, Vibhav. On lifting the gibbs sampling algorithm. In *Advances in Neural Information Processing Systems 25*, pp. 1664–1672, 2012.

Wang, William Yang, Mazaitis, Kathryn, and Cohen, William W. Programming with personalized pagerank: A locally groundable first-order probabilistic logic. *arXiv preprint arXiv:1305.2254*, 2013.

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

Zinkevich, Martin, Weimer, Markus, Smola, Alex, and Li, Lihong. Parallelized stochastic gradient descent. *Advances in Neural Information Processing Systems*, 23(23):1–9, 2010.