

PageRank, ProPPR and Stochastic Logic Programs

Dries Van Daele, Angelika Kimmig, and Luc De Raedt

Department of Computer Science, KU Leuven,
Celestijnenlaan 200a, 3001 Heverlee, Belgium
`{dries.vandaele, angelika.kimmig, luc.deraedt}@cs.kuleuven.be`

Abstract. ProPPR is a recently introduced probabilistic logic programming language inspired by stochastic logic programs that uses personalized PageRank for efficient inference. We clarify the link between these two frameworks by mapping the personalized PageRank distribution of a ProPPR program to an incomplete stochastic logic program and showing that the resulting programs induce the same probability distribution over queries.

Keywords: Stochastic Logic Programs; ProPPR; personalized PageRank; mapping probabilistic logic languages

1 Introduction

The PageRank algorithm was originally introduced as a method for objectively ranking the importance of web pages. It has found numerous other applications [1, 2, 9]. With regard to first-order probabilistic logics, the probabilistic Prolog ProPPR employs a variant of PageRank as the basis for an efficient inference method [8]. As indicated by [8], ProPPR is similar in spirit to stochastic logic programs (SLPs) [4, 5], but biased towards short derivations. The key contribution of this note is that we formalize the relationship among these two languages by providing a mapping from ProPPR’s personalized PageRank distribution to incomplete SLPs that preserves the probability distributions.

2 PageRank

The PageRank algorithm [7] assigns a weight to individual web pages, expressing their relative importance on the World Wide Web. Intuitively, the resulting PageRank distribution can be regarded as the likelihood that a ‘random surfer’ [3] arrives at each respective web page. By regarding the World Wide Web as a graph, where web pages are nodes and hyperlinks are edges, a ‘random surfer’ can be simulated by executing a modified random walk over the graph. In each step, an edge is followed by choosing uniformly between the outgoing edges. To ensure that the random walk can reach every part of the graph, there is a fixed probability of jumping to a random node in the graph. When the likelihood of jumping to a node is not uniformly distributed over all nodes, the algorithm is referred to as personalized PageRank.

3 Stochastic Logic Program

A stochastic logic program (SLP) [4, 5] is a definite clause program that has a probabilistic interpretation. It generalizes hidden Markov models, stochastic context-free grammars, and undirected Bayes' nets [6].

A definite clause program consists of a set of definite clauses. A definite clause is a clause that has exactly one positive literal, and any number of negative literals. We represent definite clauses using rules of the form $A \leftarrow B$, where A is the positive literal, called the head of the clause, and B is a conjunction of literals referred to as the clause body.

A pure SLP associates a probability label to each of its clauses. A complete SLP ensures that the probability labels attached to the clauses of each predicate sum up to one. An SLP that is both pure and complete then specifies for each query a probability distribution over its SLD derivations. These probability distributions are computed using a stochastic SLD procedure. The weight of a particular query q with answer substitution θ is defined under this stochastic SLD procedure as the sum of the probabilities of the SLD refutations of $\leftarrow q\theta$. The probability of a single SLD refutation is the product of the weighted probabilities of the clauses selected in its SLD derivation steps. There, the weighting involves dividing each of the probabilities by the sum of the probabilities of all clauses on which unification succeeds during their respective SLD derivation step. This is done to ensure that the outgoing probabilities from each node in the SLD tree add up to one.

Figure 1 shows an SLD tree along with the stochastic logic program from which it was derived. Since an SLP may contain failure nodes (as in this example), the actual probability distribution P_S over the solutions is obtained by dividing the weight associated with each solution by the total sum of all SLD refutations:

$$\begin{aligned}
 P(\text{sld_refutations}(\leftarrow q(a))) &= 0.3 * 0.6 * 0.3 + 0.7 * 0.1 \\
 P(\text{sld_refutations}(\leftarrow q(b))) &= 0.3 * 0.4 * 0.7 + 0.7 * 0.9 \\
 P_S(q(a)) &= \frac{P(\text{sld_refutations}(\leftarrow q(a)))}{P(\text{sld_refutations}(\leftarrow q(a))) + P(\text{sld_refutations}(\leftarrow q(b)))} = 0.148 \\
 P_S(q(b)) &= \frac{P(\text{sld_refutations}(\leftarrow q(b)))}{P(\text{sld_refutations}(\leftarrow q(a))) + P(\text{sld_refutations}(\leftarrow q(b)))} = 0.852
 \end{aligned}$$

4 PageRank over a Modified SLD Tree

The probabilistic logic programming language ProPPR [8] offers efficient approximate inference for a given query and logic program through the computation of the personalized PageRank distribution over a subgraph of a modified SLD tree for that query and logic program. Furthermore, ProPPR supports learning, and

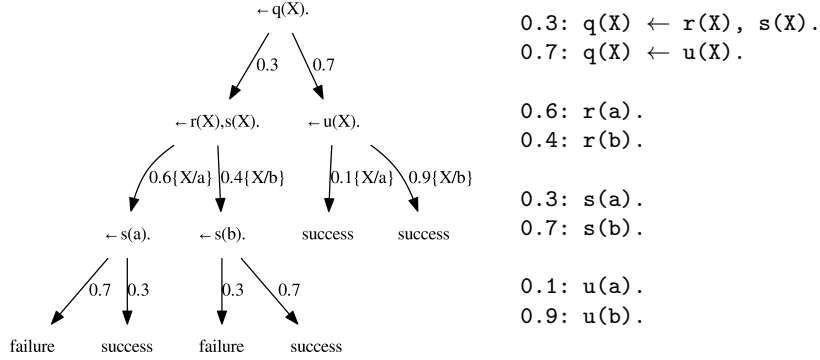


Fig. 1. An SLD Tree with probabilities annotating its edges and the stochastic logic program describing it

has shown to perform well on an entity resolution task, a classification task, and a joint inference task [8]. In this paper, we will only consider the exact inference case, where the personalized PageRank process is performed over the entire modified SLD tree.

Since personalized PageRank is used, the likelihood of jumping to a node will not be uniformly distributed over all nodes. Instead, this jump to a random node is replaced by a jump to the root of the SLD tree, as it would be illogical to jump from a particular node in the SLD tree to an unrelated random node. This can be regarded as simply restarting the walk for that query. Furthermore, instead of uniformly following outgoing edges, the probability of following a particular edge is specified by the probability label of the associated clause. Lastly, the SLD tree itself has to be slightly modified. Success nodes and failure nodes do not have any outgoing edges. In order to avoid these nodes acting as sinks that leak probability mass, the SLD tree is extended with an edge from each success and failure node to itself. The personalized PageRank process is applied on the resulting graph. Assuming an iterative computation method, it suffices to initialize the distribution in the first step by placing all the probability mass in the query node (i.e. the node that served as the root of the original SLD tree).

Note that we make use of the SLP syntax when writing programs that describe a PageRank distribution in order to facilitate comparison. Assuming that the program in Figure 1 describes a PageRank distribution, we can – given a particular restart probability (e.g. 0.2) – derive the graph in Figure 2. For ease of presentation, the restart edges have been omitted from the graph. This means that each node has an additional edge connecting it to the query node. To ensure that the probabilities of all outgoing edges sum to one, the original probabilities are normalized to sum to 1 minus the restart probability (0.8 in the example). An alternative, more explicit notation of this logic program is shown in Figure 3. It depicts the actual, rescaled weights of the clauses associated with each predicate – a consequence of accounting for the restart probability. Furthermore, by using the special symbol $\mathcal{R}$ for the jump to the query node, the extra-logical nature of

this restart step is highlighted. If inference is being performed on a goal such as $(r(X), s(X))$, and the first subgoal $r(X)$ triggers the restart clause, all subsequent subgoals (here just $s(X)$) are dropped, and all substitutions obtained so far forgotten. The sole remaining goal is then the query node (with fresh variables). So, as opposed to the cut operator in Prolog which only has a local effect on the clause calling it, the effect of the restart is far greater.

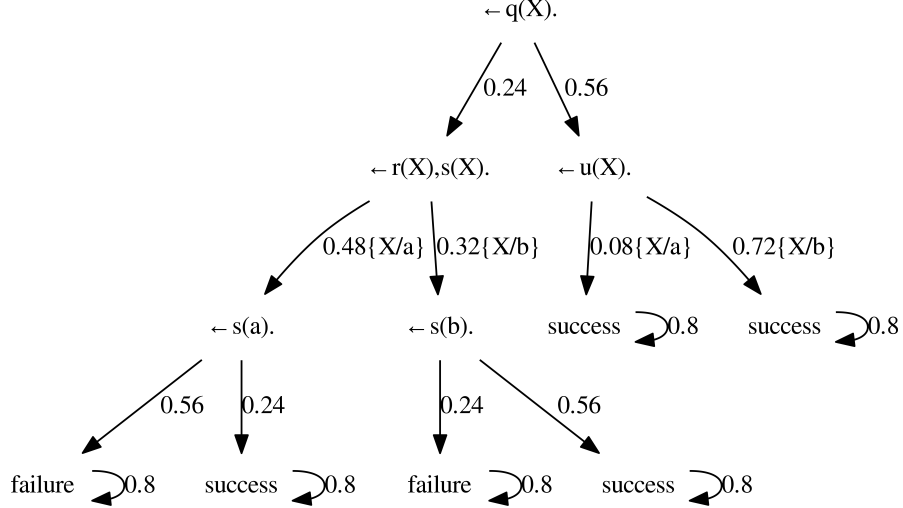


Fig. 2. The modified SLD tree (restart edges omitted) for the SLP in Figure 1.

When applying personalized PageRank over this graph, we retrieve a probability distribution over all its nodes. In particular, we are interested in the probabilities of the success nodes. From these probabilities we can determine the normalized probabilities of the answers to the query $\leftarrow q(X)$. As for SLPs, we sum the probabilities of the success nodes associated with each SLD refutation for a particular query answer, and normalize by dividing each value by the sum of all success node probabilities:

$$P_{\mathcal{P}}(q(a)) = \frac{0.027648 + 0.0448}{0.027648 + 0.0448 + 0.4032 + 0.043008} = 0.1397$$

$$P_{\mathcal{P}}(q(b)) = \frac{0.4032 + 0.043008}{0.027648 + 0.0448 + 0.4032 + 0.043008} = 0.8603$$

Though similar in magnitude, these differ from the ones of the SLP in Figure 1. We can however express the distribution over the success nodes specified by this PageRank distribution using an incomplete SLP, as we show next.

0.24: $q(X) \leftarrow r(X), s(X).$	0.24: $q(X) \leftarrow r(X), s(X).$
0.56: $q(X) \leftarrow u(X).$	0.56: $q(X) \leftarrow u(X).$
0.20: $q(X) \leftarrow \mathcal{R}.$	
0.48: $r(a).$	0.48: $r(a).$
0.32: $r(b).$	0.32: $r(b).$
0.20: $r(X) \leftarrow \mathcal{R}.$	
0.24: $s(a).$	0.24: $s(a).$
0.56: $s(b).$	0.56: $s(b).$
0.20: $s(X) \leftarrow \mathcal{R}.$	
0.08: $u(a).$	0.08: $u(a).$
0.72: $u(b).$	0.72: $u(b).$
0.20: $u(X) \leftarrow \mathcal{R}.$	

Fig. 3. An explicit notation for the SLP in Figure 1 under the PageRank interpretation with restart probability 0.2 (left) and the corresponding incomplete SLP (right).

5 PageRank-SLP Semantics

The main characterizing feature of the PageRank distribution is the emphasis on solutions found through short proofs [8]. This is a direct result of the probability of jumping back to the query node. In the special case where all the success nodes occur at the same height in the SLD tree, the normalized probability of the answers will be the same as when the logic program is interpreted as an SLP. When success nodes occur at different heights, their probabilities are the probabilities of their respective SLD-refutations in the SLP scaled by a factor $(1 - \alpha)^n$, where α is the restart probability (i.e., $(1 - \alpha)$ the probability that a restart edge is not followed), and n the number of SLD-resolution steps in the given SLD-refutation. Alternatively, this can be regarded as the probability distribution of an incomplete SLP of which the probability labels are derived by rescaling the probability labels of the original, complete SLP with $(1 - \alpha)$. As illustrated in Figure 3, this corresponds to the incomplete SLP obtained by dropping the restart clauses from the explicit representation of the PageRank distribution.

Theorem 1 (reduction of ProPPR programs to incomplete SLPs). *For a ProPPR program $\mathcal{P}$ in the form of a pure complete SLP and a restart probability $\alpha \in [0, 1]$, let $\mathcal{S}$ be the incomplete SLP obtained by replacing each probability*

label p in $\mathcal{P}$ by $p \cdot (1 - \alpha)$. Then, for all queries Q and answer substitutions θ , we have $P_S(Q\theta) = P_P(Q\theta)$.

Proof Sketch. We assume that the PageRank vector of the ProPPR program is computed using an iterative method. Initially, we place all the probability mass in the query node, and then let it flow through the graph. In each iteration, the nodes distribute their current probability mass among their respective children.

Since the modified SLD tree has no sinks that could leak probability mass, and considering that each node passes an α share of its probability mass back to the query node, we can conclude that – after the first iteration – the query node will stabilize at the probability value α . This also means that the probability mass distributed by a node after stabilization will be an α portion of the volume distributed in the step before stabilization.

With the exception of the query node, and the success and failure nodes, all nodes in the modified SLD tree receive their probability from nodes that are at a strictly smaller depth in the original SLD tree. Given that the probability of the parent of a node becomes stable at iteration i , we can conclude that the node itself converges at iteration $i + 1$. This happens because the probability mass it receives from its parent only gets weighted by the probability label of the clause used in its SLD-derivation from its parent. Since that probability label is static, the resulting value will converge once the parent probability is fixed.

The success and failure nodes initially receive their probability mass from their respective parent. At each iteration, the success and failure nodes retain a $1 - \alpha$ share of their probability mass, and pass the rest back to the query node. This outflow will be cancelled out by the inflow produced by the parent nodes. This ensures that the moment of convergence of the success and failure nodes coincides with that of their respective parents.

Since the probability mass of the nodes on a path to a success node will each converge in turn, their final probability when reaching the success node will be the product of their associated rescaled probability labels (i.e. the probability of the SLD refutation as computed by an incomplete SLP). Since the success nodes simply aggregate these incoming probabilities, it can be concluded that their final values will be equal to those specified by the associated incomplete SLP. $\square$

6 Conclusions

We have shown that the probability distribution defined by a ProPPR program, based on its personalized PageRank distribution, corresponds to the probability distribution over answer substitutions given by an incomplete SLP with labels computed from the ProPPR program and the PageRank restart probability. This clarifies the link between the two languages and opens up possibilities for using ProPPR's efficient inference approaches for SLPs.

References

1. AC Aguiar and GC Beroza. Pagerank for low frequency earthquake detection. In *AGU Fall Meeting Abstracts*, volume 1, page 2440, 2013.
2. Stefano Allesina and Mercedes Pascual. Googling food webs: can an eigenvector measure species’ importance for coextinctions? *PLoS computational biology*, 5(9):e1000494, 2009.
3. Prasad Chebolu and Páll Melsted. Pagerank and the random surfer model. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1010–1018, 2008.
4. James Cussens. Stochastic logic programs: Sampling, inference and applications. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 115–122, 2000.
5. Stephen Muggleton. Stochastic logic programs. *Advances in inductive logic programming*, 32:254–264, 1996.
6. Stephen Muggleton. Semantics and derivation for stochastic logic programs. In *UAI Workshop on Fusion of Domain Knowledge with Data for Decision Support*, 2000.
7. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.
8. William Yang Wang, Kathryn Mazaitis, and William W Cohen. Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2129–2138, 2013.
9. Christof Winter, Glen Kristiansen, Stephan Kersting, Janine Roy, Daniela Aust, Thomas Knsel, Petra Rmmele, Beatrix Jahnke, Vera Hentrich, Felix Rckert, Marco Niedergethmann, Wilko Weichert, Marcus Bahra, Hans J. Schlitt, Utz Settmacher, Helmut Friess, Markus Bhler, Hans-Detlev Saeger, Michael Schroeder, Christian Pilarsky, and Robert Grtzmnn. Google Goes Cancer: Improving Outcome Prediction for Cancer Patients by Network-Based Ranking of Marker Genes. *PLOS Computational Biology*, 8, 2012.