Notes on ProPPR

***DRAFT v0.01 ***

Abstract

1 Introduction

[WMLC15] is an innovative attempt to combine machine learning ideas drawn from the literature on PageRank with the proof theoretic underpinnings of definite clause constraint logic programming (constrained SLD-resolution). This note is my attempt at understanding the ideas and laying out the underlying mathematical background clearly.

The basic idea is to view probabilistic SLD-resolution as a kind of graph traversal, and use ideas behind PageRank [PBMW99] to speed up traversal.

2 PageRank

First we cover some basics about PageRank, following [ACL06, ACL08]. Our interest is to develop the basics in such a way that they can directly apply to constraint-SLD graphs, which we will develop in the next section. Crucially transitions in such graphs are associated with probabilities, hence we wish to consider the setting of directed, weighted graphs, unlike [ACL06].

Let $G = (V,E)$ be a directed graph with vertex set V (of size n) and edge set E (of size m). Let $\mathbf{1}_v$ be the n-vector which takes on value 1 at v and is 0 elsewhere. Let $d(v)$ be the out-degree of vertex $v \in V$. A *distribution* over V is a non-negative vector over V. The *support* of a distribution p, $Supp(p)$ is $\{v \mid p(v) \neq 0\}$. The *volume* of a subset $S \subseteq V$, $vol(S)$ is the sum of the degrees of the vertices in S.

Let M be a Markov chain over G, i.e. an $n \times n$ matrix with positive elements, whose rows sum to 1 and whose non-zero elements M_{ij} (for $i, j \in 1 \ldots n$) are exactly (the transition probabilities of) the edges $(i, j) \in E$. Markov chains over G are our subject of interest.

Definition 2.1 *For a Markov chain M, the PageRank vector $\mathrm{pr}_M(\alpha, s)$ is the unique solution of the linear system*

$$\mathrm{pr}_M(\alpha, s) = \alpha s + (1-\alpha)\mathrm{pr}_M(\alpha, s)M \tag{1}$$

(Note that this definition is in line with [ACL08], but generalizes [ACL06] where it is given in an unweighted, undirected setting for the specific matrix $M = W = 1/2(I + D^{-1}A)$, where A is the adjacency matrix and D is the diagonal degree matrix.)

Proposition 2.2 *For any $\alpha \in [0,1)$ there is a linear transformation R_α s.t. $\mathrm{pr}_M(\alpha, s) = sR_\alpha$, where*

$$R_\alpha = \alpha\Sigma_{t=0}^{\infty}(1-\alpha)^t M^t = \alpha I + (1-\alpha)MR_\alpha$$

Proposition 2.3

$$\mathrm{pr}_M(\alpha, s) = \alpha s + (1-\alpha)\mathrm{pr}_M(\alpha, sM) \tag{2}$$

The proof is elementary:

$$\begin{array}{rcll} pr_M(\alpha, s) & = & sR_\alpha & \\ & = & \alpha s + (1-\alpha) sMR_\alpha & \text{(Proposition 2.2)} \\ & = & \alpha s + (1-\alpha) pr_M(\alpha, sM) & \text{(Proposition 2.2)} \end{array}$$

The following ("PageRank Nibble") algorithm is taken from [ACL06] with a slight variation (working with a Markov chain rather than an adjacency matrix). The key insight is to work with a pair of distributions (p, r) satisfying:

$$p + pr_M(\alpha, r) = pr_M(\alpha, s) \tag{3}$$

We shall think of p as an *approximate* PageRank vector, approximating $pr_M(\alpha, s)$ (from below) with *residual* vector r. Below we will use the notation $\text{apr}_M(\alpha, s, r)$ to stand for a vector p in the relationship given by Equation 3.

We now introduce the operation $push_u(p, r)$, generalizing [ACL06, Section 3] to the setting of Markov chains (and correcting some typos in [WMLC15, Table 2]):

1. Let $p' = p$ and $r' = r$ except for the following changes:
   (a) $p'(u) = p(u) + \alpha r(u)$
   (b) $r'(u) = (1-\alpha) r(u) M_{uu}$
   (c) For each v s.t. $(u, v) \in E$: $r'(v) = r(v) + (1-\alpha) r(u) M_{uv}$
2. Return (p', r').

It simulates one step of a random walk, at u.

The key lemma satisfied by this definition is:

Lemma 2.4 *Let p', r' be the result of the operation* $\text{push}_u(p, r)$. *Then* $p' + \text{pr}_M(\alpha, r') = p + \text{pr}_M(\alpha, r) (= \text{pr}_M(\alpha, s))$.

In proof, following [ACL06, Appendix], note that after a $push_u(p, r)$ operation, the following is true:

$$\begin{aligned} p' &= p + \alpha r(u) \mathbf{1}_u \\ r' &= r - r(u) \mathbf{1}_u + (1-\alpha) r(u) \mathbf{1}_u M \end{aligned}$$

Now:

$$\begin{array}{rcll} p + pr_M(\alpha, r) & = & p + pr_M(\alpha, r - r(u)\mathbf{1}_u) + pr_M(\alpha, r(u)\mathbf{1}_u) & \text{(Linearity)} \\ & = & p + pr_M(\alpha, r - r(u)\mathbf{1}_u) + (\alpha r(u)\mathbf{1}_u + (1-\alpha) pr_M(\alpha, r(u)\mathbf{1}_u M) & (2) \\ & = & (p + (\alpha r(u)\mathbf{1}_u) + pr_M(\alpha, r - r(u)\mathbf{1}_u + (1-\alpha) r(u)\mathbf{1}_u M) & \text{(Linearity)} \\ & = & p' + pr_M(\alpha, r') & \end{array}$$

Now define the $\text{ApproxPageRank}(v, \alpha, e)$ algorithm as follows:

1. Let $p = \vec{0}$ and $r = \mathbf{1}_v$.
2. While there exists a vertex $u \in V : (r(u)/d(u)) \geq \varepsilon$, apply $push_u(p, r)$.
3. Return p.

The value returned is an $\text{apr}(\alpha, \mathbf{1}_v, r)$ s.t. for all $u \in V, r(u) < \varepsilon d(u)$.

The following are the main results, with proofs as in [ACL06, Appendix].

Lemma 2.5 *([ACL06, Lemma 2]) Let T be the total number of push operations performed by ApproxPageRank, and let d_i be the degree of the vertex u used in the i'th push. Then $\Sigma_{i=1}^{T} d_i \leq (1/\varepsilon\alpha)$.*

Theorem 2.6 *([ACL06, Theorem 1]) ApproxPageRank(v, α, ε) runs in time $O(1/(\varepsilon\alpha))$, and computes an approximate PageRank vector $p = apr_M(\alpha, \mathbf{1}_v, r)$ s.t.* $\text{vol}(\text{Supp}(p)) \leq 1/(\varepsilon\alpha)$ *and for all $u \in V$, $r(u) < \varepsilon d(u)$.*

3 Constrained SLD-resolution

Though [WMLC15] is presented for just definite clause programs, we shall follow the tradition of logic programming research and consider constraint logic programming, after [JL87]. This gives us significant generality and lets us avoid speaking of syntactic notions such as most general unifiers. Hence we assume an underlying constraint system $\mathcal{C}$ [Sar92], defined over a logical vocabulary. Atomic formulas in this vocabulary are called *constraints*. $\mathcal{C}$ specifies the notions of *consistency* of constraints and *entailment* between constraints. We assume for simplicity the existence of a vacuous constraint `true`.

We assume a fixed program P, consisting of a (finite) collection of (implicitly universally quantified) clauses $h \leftarrow c, b_1, \ldots, b_k$ (where h, b_i are atomic formulas and c is a constraint). Below, for a formula ϕ we will use the notation $var(\phi)$ to refer to the set of variables in ϕ. Given a set of variables Z and a formula ϕ by $\delta Z \phi$ we will mean the formula $\exists V \phi$ where $V = var(\phi) \setminus Z$.

We assume given an initial *goal* g (an atomic formula), with $Z = var(g)$. A *configuration* (or *state*) s is a pair $\langle a_1, \ldots, a_n; c \rangle$, with $n \geq 0$, c a constraint, and goals a_i. The *variables* of the state are $var(a_1 \wedge \ldots \wedge a_n \wedge c)$. s is said to be *successful* if $n = 0$, *consistent* if c is consistent and *failed* (or *inconsistent*) if c is inconsistent.

Two states $\langle a_1, \ldots, a_n; c \rangle$ and $\langle b_1, \ldots, b_k; d \rangle$ are equivalent if $\vdash \delta Z(a_1 \wedge \ldots \wedge a_n \wedge c) \Leftrightarrow \delta Z(b_1 \wedge \ldots \wedge b_k \wedge d)$ (where the $\vdash$ represents the entailment relation of the underlying logic, including the constraint entailment relation). Note that any two inconsistent states are equivalent, per this definition.

We now consider the transitions between states. For simplicity, we shall confine ourselves to a fixed *selection rule*. Given the sequence of goals in a state, a selection rule chooses one of those goals for execution. Logically, any goal can be chosen (e.g. Prolog chooses the first goal).

A clause is said to be *renamed apart* from a state if it has no variables in common with the state. If $g = p(s_1, \ldots, s_k)$ and $h = p(t_1, \ldots, t_k)$ are two atomic formulas with the same predicate p and arity k, then $g = h$ stands for the collection of equalities $s_1 = t_1, \ldots, s_k = t_k$. We say that a state $s = \langle a_1, \ldots, a_n; c \rangle$ *can transition to* a state $\langle a_1, \ldots, a_{i-1}, b_1, \ldots, b_k, a_{i+1}, \ldots a_n; c, d, (a_i = h) \rangle$ provided that (a) s is consistent, (b) a_i is chosen by the selection rule, (c) there is a clause $C = h \leftarrow d, b_1, \ldots, b_k$ in P, renamed apart from s s.t. h and a_i are atomic formulas with the same predicate name and arity. We say that a_i is the *selected goal* (for the transition) and C the *selected clause*.

Note that the current state will have at most k states it can transition to, if the program has k clauses with the predicate and arity of the selected goal. (It will have fewer than k if resulting states are equivalent.) Of course, not all resulting configurations may be consistent. Finally, note that a state can never transit to itself (since the selected goal exists in the source state of the transition but not the target state.)[1]

Constrained SLD-resolution starts with a state $\langle g; \texttt{true} \rangle$ and transitions to successive states, until a state is reached which is successful or failed.

We are interested in *stochastic logic programs*. For the purposes of this note, these are programs that supply with each transition a *probability* for the transition (a non-negative number bounded by 1) in such a way that the sum of the probabilities across all transitions from this state is 1. The probabilities may depend on the current state. In stochastic logic programs as described in [Mug96] the probability is a number directly associated with the clause (and independent of the state). [WMLC15] describes a more elaborate setting: a clause specifies "features" (dependent on the current state) which are combined with a (learnt) matrix of weights to produce the probability. For the purposes of this note we shall not be concerned with the specific mechanism.

Note that multiple transitions from a state s (each using a different clause) may lead to the same (equivalent) state t. In such cases we consider that there is only one transition $s \rightarrow t$, and its associated probability is the sum of the probabilities across all clauses contributing to the transition.

[1] Note that in theory a clause has an infinite number of variants that are renamed apart from a given state. It can be shown that only one of them needs to be considered for selection, the results for all other choices can be obtained by merely renaming the results for this choice.

In general, we will only be concerned with goals that have a finite derivation graph. This can be guaranteed by placing restrictions on the expressiveness of programs (e.g. by requiring that programs satisfy the Datalog condition), but we shall not make such further requirements.

4 Applying PageRank to constraint-SLD graphs

Acknowledgements.

References

[ACL06] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, pages 475–486, Washington, DC, USA, 2006. IEEE Computer Society.

[ACL08] Reid Andersen, Fan Chung, and Kevin Lang. Local partitioning for directed graphs using pagerank. *Internet Math.*, 5(1-2):3–22, 2008.

[JL87] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '87, pages 111–119, New York, NY, USA, 1987. ACM.

[Mug96] Stephen Muggleton. Stochastic logic programs. In *New Generation Computing*. Academic Press, 1996.

[PBMW99] 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. Previous number = SIDL-WP-1999-0120.

[Sar92] Vijay A. Saraswat. The category of constraint systems is cartesian-closed. In *LICS*, 1992.

[WMLC15] William Yang Wang, Kathryn Mazaitis, Ni Lao, and William W Cohen. Efficient inference and learning in a large knowledge base. *Machine Learning*, 100(1):101–126, 2015.