Learning Relational Sum-Product Networks

Aniruddh Nath and **Pedro Domingos**
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350, U.S.A.
{*nath, pedrod*}*@cs.washington.edu*

Abstract

Sum-product networks (SPNs) are a recently-proposed deep architecture that guarantees tractable inference, even on certain high-treewidth models. SPNs are a propositional architecture, treating the instances as independent and identically distributed. In this paper, we introduce *Relational Sum-Product Networks* (RSPNs), a new tractable first-order probabilistic architecture. RSPNs generalize SPNs by modeling a set of instances jointly, allowing them to influence each other's probability distributions, as well as modeling probabilities of relations between objects. We also present LearnRSPN, the first algorithm for learning high-treewidth tractable statistical relational models. LearnRSPN is a recursive top-down structure learning algorithm for RSPNs, based on Gens and Domingos' LearnSPN algorithm for propositional SPN learning. We evaluate the algorithm on three datasets; the RSPN learning algorithm outperforms Markov Logic Networks in both running time and predictive accuracy.

Introduction

Graphical probabilistic models compactly represent a joint probability distribution among a set of variables. Unfortunately, inference in graphical models is intractable. In practice, using graphical models for most real-world applications requires either using approximate algorithms, or restricting oneself to a subset of graphical models on which inference is tractable. A common restriction that ensures tractability is to use models with low treewidth (Bach and Jordan 2001). However, not all tractable models have low treewidth. Poon and Domingos (2011) recently proposed *Sum-Product Networks (SPNs)*, a tractable probabilistic architecture that guarantees efficient exact inference. SPNs subsume most known tractable probabilistic models, and can compactly represent some high-treewidth distributions. SPNs can be seen as a deep architecture with alternating layers of sum nodes and product nodes. Since their introduction, several SPN learning algorithms have been proposed, and SPNs have been applied to a variety of problems (Delalleau and Bengio 2011; Dennis and Ventura 2012; Amer and Todorovic 2012; Peharz et al. 2013; Gens and Domingos 2013, etc.).

Besides intractability, one other key shortcoming of most widely used graphical models is their reliance on the i.i.d. assumption. In many real-world applications, instances are not truly independent, and can be better modeled if their interactions are taken into account. This is one of the key insights of the Statistical Relational Learning (SRL) community. SRL techniques have been applied to a wide variety of tasks, including collective classification, link prediction, natural language processing, etc. However, most SRL methods build on graphical models (e.g. Markov logic; Richardson and Domingos 2006), and suffer from the same computational difficulties; these are compounded by the additional problem of modeling interactions between instances.

In this paper, our goal is to combine these two lines of research: tractable probabilistic models and relational learning. One line of related work uses Naïve Bayes models in structured domains (Flach and Lachiche 2004; Landwehr et al. 2005; Davis et al. 2007). Although tractable, Naïve Bayes models are quite limited in expressiveness. PRISM is a probabilistic logic that supports efficient inference, but only under a very restrictive set of assumptions (Sato and Kameya 2008). PSL (Bröcheler et al. 2010) supports efficient inference, but uses fuzzy logic-based semantics instead of standard probabilistic semantics. TML (Domingos and Webb 2012) is a subset of Markov logic on which efficient inference can be guaranteed. TML is surprisingly expressive, subsuming most previous tractable models. However, a TML knowledge base determines the set of possible objects in the domain, and the relational structure among them. This limits the applicability of TML to learning; a TML knowledge base cannot be learned on one set of objects and applied to another mega-example with different size or structure.

To address this, we propose *Relational Sum-Product Networks* (RSPNs), a new tractable relational probabilistic architecture. RSPNs generalize SPNs by modeling a set of instances jointly, allowing them to influence each other's probability distributions, as well as modeling the probabilities of relations between objects. An RSPN can be trained on a set of mega-examples, and applied to a previously unseen mega-example with different structure (given a part decomposition as input). We also introduce *LearnRSPN*, the first algorithm for learning tractable statistical relational models. Intractable inference has historically been a major obstacle to the wider adoption of statistical relational methods; the de-

velopment of learning and inference algorithms for tractable relational models could go a long way towards making SRL more widely applicable.

Background

Sum-Product Networks

A sum-product network (SPN) is a rooted directed acyclic graph with univariate distributions at the leaves; the internal nodes are (weighted) sums and (unweighted) products.

Definition 1. (Gens and Domingos 2013)

1. A tractable univariate distribution is an SPN.
2. A product of SPNs with disjoint scopes is an SPN. (The scope of an SPN is the set of variables that appear in it.)
3. A weighted sum of SPNs with the same scope is an SPN, provided all weights are positive.
4. Nothing else is an SPN.

See fig. 1 for an example SPN.

A univariate distribution is tractable iff its partition function and mode can be computed in constant time.

Intuitively, an SPN can be thought of as an alternating set of mixtures (sums) and decompositions (products) of the leaf variables. If the values at the leaf nodes are set to the partition functions of the corresponding univariate distributions, then the value at the root is the partition function (i.e. the sum of the unnormalized probabilities of all possible assignments to the leaf variables). This allows the partition function to be computed in time linear in the size of the SPN.

If the values of some variables are known, the leaves corresponding to those variables' distributions are set to those values' probabilities, and the remainder are replaced by their (univariate) partition functions. This yields the unnormalized probability of the evidence, which can be divided by the partition function to obtain the normalized probability. The most probable state of the SPN, viewing sums as marginalized-out hidden variables, can also be computed in linear time.

Learning SPNs

The first learning algorithms for sum-product networks used a fixed network structure, and only optimized the weights (Poon and Domingos 2011; Amer and Todorovic 2012; Gens and Domingos 2012). The network structure is domain-dependent; applying SPNs to a new problem required manually designing a suitable network structure.

More recently, several algorithms have been proposed that learn both the weights and the network structure, allowing SPNs to be applied out-of-the-box to new domains. Dennis and Ventura (2012) suggested an algorithm that builds an SPN based on a hierarchical clustering of variables. Gens and Domingos (2013) construct SPNs top-down, recursively partitioning instances and variables. Peharz et al. (2013) construct SPNs bottom-up, greedily merging SPNs into models of larger scope. These algorithms have been shown to perform well on a variety of domains, making more accurate predictions than conventional graphical models, while guaranteeing tractable inference.

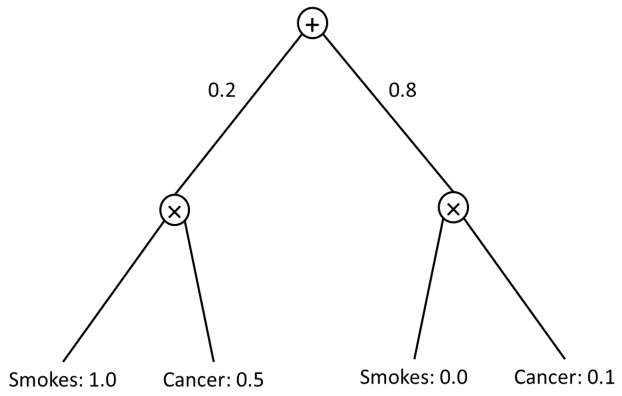

Figure 1: Example SPN over the variables 'Smokes' and 'Cancer'. The weights of the sum node are indicated next to the corresponding edges. At the leaves, each variable is modeled via a Bernoulli distribution; the numbers next to each leaf are the success probabilities of the corresponding Bernoulli distributions.

Statistical Relational Learning

SPNs are a propositional representation, modeling instances as independent and identically distributed (i.i.d.). Although the i.i.d. assumption is widely used in statistical machine learning, it is often an unrealistic assumption. In practice, objects usually interact with each other; Statistical Relational Learning algorithms (Getoor and Taskar 2007) can capture dependencies between objects, and make predictions about relationships between them.

Markov Logic Networks (MLNs; Richardson and Domingos 2006) are a widely-used representation for relational learning. An MLN is a first-order representation of the dependencies between objects in a domain. Given a *mega-example* (a set of related objects), an MLN can be *grounded* into a propositional graphical model representing a joint probability distribution over the attributes and relations among those objects. Unfortunately, the resulting graphical model is typically high-treewidth, and inference is intractable. In practice, users of SRL methods typically resort to approximate inference algorithms based on MCMC or loopy belief propagation, resulting in long runtimes and unreliable predictions. Like propositional graphical models, statistical relational models can be trivially restricted to the low-treewidth case, but this comes at great cost to the representational power of the model.

Tractable Markov Logic (TML; Domingos and Webb 2012; Webb and Domingos 2013) is a subset of Markov Logic that guarantees polynomial-time inference, even in certain cases where the ground propositional model would be high-treewidth. TML is expressive enough to capture several cases of interest, including junction trees, non-recursive PCFGs and hierarchical mixture models. A TML knowledge base is a generative model that decomposes the domain into parts, with each part drawn probabilistically from a class hierarchy. Each part is further

probabilistically decomposed into subparts (according to its class). The key limitation of TML is that the knowledge base fully specifies the set of possible objects in the domain, and their class and part structure. A TML knowledge base cannot generalize across mega-examples of varying size or structure.

This paper uses some first-order logic terminology. '*Predicate*' refers to a first-order logic predicate. (Our representation and algorithm support numeric attributes and relations as well, but for simplicity we focus on the Boolean case.) A *grounding* of a predicate (or a *ground atom*) is a replacement of all its variables by constants.

Relational Sum-Product Networks

Exchangeable Distribution Templates

Before we define RSPNs, we first define the notion of an *Exchangeable Distribution Template* (EDT).

Definition 2. Consider a finite set of variables $\{X_1, \ldots, X_n\}$ with joint probability distribution P. Let $S(n)$ be the set of all permutations on $\{1, \ldots, n\}$. $\{X_1, \ldots, X_n\}$ is a *finite exchangeable set* with respect to P if and only if $P(X_1, \ldots, X_n) = P(X_{\pi(1)}, \ldots, X_{\pi(n)})$ for all $\pi \in S(n)$. (Diaconis and Freedman 1980).

Note that finite exchangeability does not require independence: a set of variables can be exchangeable despite having strong dependencies. (For example, consider binary variables $X_1, \ldots, X_n$, with a uniform distribution over value assignments with an even number of non-zero variables.)

Definition 3. An *Exchangeable Distribution Template* (EDT) is a function that takes a set of variables $\{X_1, \ldots, X_n\}$ as input (n is unknown a priori), and returns a joint probability distribution P with respect to which $\{X_1, \ldots, X_n\}$ is exchangeable. We refer to the probability distribution P returned by the EDT for a given set of variables as an *instantiation* of that EDT.

Example 1. The simplest family of EDTs simply returns a product of identical univariate distributions over each of $X_1, \ldots, X_n$. For example, if the variables are binary, then an EDT might model them as a product of Bernoulli distributions with some probability p.

Example 2. Consider an EDT over a set of binary variables $X_1, \ldots, X_n$, returning the following distribution: $P(X_1, \ldots, X_n) \propto \frac{\lambda^k}{k!} e^{-\lambda}$, where $k = \sum_{X_i} \mathbf{1}[X_i]$. λ is a parameter of the EDT. This is an EDT with a Poisson distribution over the number of variables in the set with value 1. (The probabilities must be renormalized, since the set of variables is finite.) Note that this EDT does not assume independence among variables.

Intuitively, EDTs can be thought of as probability distributions that depend only on aggregate statistics, and not on the values of individual variables in the set.

Relational Sum-Product Networks

Relational Sum-Product Networks (RSPNs) jointly model the attributes and relations among a set of objects. RSPNs inherit TML's notion of parts and classes. As in TML, each part of a class also belongs to some class. Unlike TML, an RSPN class's parts may be *unique* or *exchangeable*. An object's unique parts are those that play a special role, e.g. the commander of a platoon, the queen of a bee colony, or the hub of a social network. The exchangeable parts are those that behave interchangeably: soldiers in a platoon, worker bees in a colony, spokes in a network, and so on.

Definition 4. A *definition* for class C in an RSPN consists of:

- A set of *attributes*: unary predicates A applicable to individuals of C.
- A vector $U_C = (P_1, \ldots, P_n)$ specifying the classes of *unique parts*.
- A vector $E_C = (P_1, \ldots, P_n)$ of classes of *exchangeable parts*
- A set of *relations* between parts: predicates of the form $R_1(P_1, P_2)$ or $R_2(C, P_1)$, where P_1 and P_2 are either unique or exchangeable part classes of C. Predicates may be of any arity.
- A *class SPN* whose leaves fall into three categories:
  - L_A^C is a univariate distribution over attribute A of C;
  - L_R^C is an EDT over binary (or higher-order) predicate R involving C and/or its part types (e.g. formulas of the form $R_1(P_1, P_2)$ or $R_2(C, P_1)$, where P_1 and P_2 are part classes of C);
  - L_P^C is a sub-SPN for part class P (i.e. a valid class SPN for class P).

  All attributes, relations and parts of class C must be included in the class SPN. The class SPNs can have arbitrary internal structure.

Each P_k in U_C and E_C is an RSPN class. In principle, a class may occur multiple times in each part vector. In this case, each part may be uniquely identified by its index in the corresponding vector. However, to simplify this discussion, we assume that each class occurs at most once in (U_C, E_C).

Example 3. The following is a partial class specification for a simple political domain. A 'Region' consists of an arbitrary number of nations, and relationships between nations are modeled at this level. A 'Nation' has a unique government and an arbitrary number of people. National properties such as 'High GDP' are modeled here. The 'Supports' relation can capture a distribution over the number of people in the nation who support the government.

```
class Region:
  exchangeable part Nation
  relation Adjacent(Nation,Nation)
  relation Conflict(Nation,Nation)

class Nation:
  unique part Government
  exchangeable part Person
  attribute HighGDP
  relation Supports(Person,Government)
```

See fig. 2 for an example class SPN.

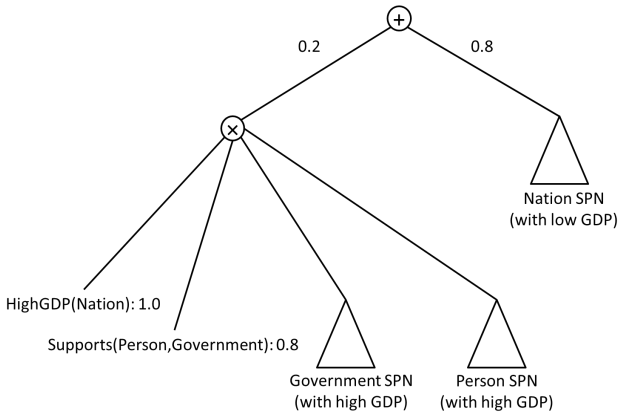

Figure 2: Partial SPN for the 'Nation' class (example 3). The sum node at the root represents a mixture model over two possible SPNs for 'Nation': one with high GDP (left), and the other with low GDP (right; omitted). The 'HighGDP' attribute is modeled by a Bernoulli distribution, and the 'Supports' relation is modeled by an EDT of the form described in example 1.

Grounding an RSPN

Like MLNs, RSPNs are templates for propositional models. To generate a ground SPN from an RSPN, we take as input a *part decomposition*:

Definition 5. For RSPN class C, a *C-rooted part decomposition* consists of:

- An object O of class C ('*root*');
- Exactly one P-rooted part decomposition for each unique part class P in U_C ('*unique child*');
- A (possibly empty) set of P-rooted decompositions for each exchangeable part class P in E_C ('*exchangeable children*').

The decomposition must be acyclic, i.e. an object may not be its own child or descendant. (This ensures that the ground SPN is acyclic even when the RSPN class structure contains cycles.)

To *ground* a class SPN is to instantiate the template for a specific set of objects. Given a class C and a part decomposition D rooted at object O, grounding C's SPN yields a propositional SPN whose leaf distributions are over attributes and relations involving the objects in D. This is done recursively as follows:

- For leaves of the form L_A^C: replace the univariate distribution over predicate A in the class SPN with a univariate distribution over $A(O)$ in the ground SPN.
- For leaves of the form L_R^C: replace the EDT over $R(X, Y)$ in the class SPN with an instantiation of that EDT over the groundings of R.
- For leaves of the form L_P^C: recursively ground P's class SPN over each type-P child of object O. Replace the leaf with a product over the resulting ground SPNs.

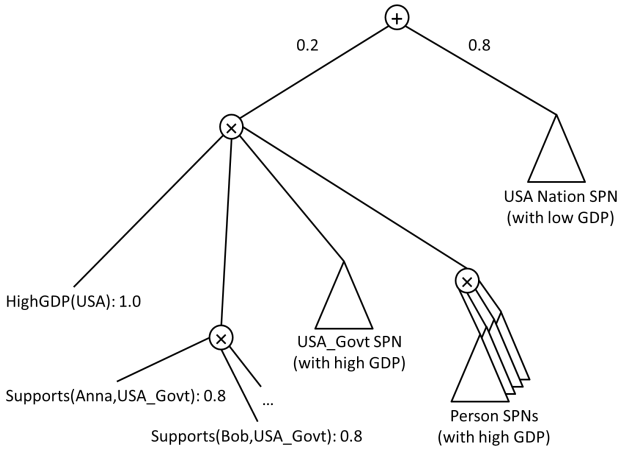

Figure 3: Example grounding of the 'Nation' class, with SPN from fig. 2.

Note that each attribute/relation/part may correspond to more than one leaf in the SPN (in different children of a sum node), potentially modeled by a different univariate distribution/EDT/class SPN. See fig. 3 for an example ground SPN.

Representational Power

The main limitation of the RSPN representation is that individual relational atoms are not modeled directly, but through aggregations. In this respect, it is similar to Probabilistic Relational Models (PRMs; Friedman et al. 1999)—though RSPNs guarantee tractable inference, unlike PRMs. Aggregations are extremely useful for capturing relational dependencies (Natarajan et al. 2012). For example, a person's smoking habits may depend on the number of friends she has who smoke, and not the smoking habits of each individual friend. Nevertheless, aggregations are not well-suited to capturing relational patterns that depend on specific paths of influence, such as Ising models.

It is important to note than RSPNs (and SPNs) are not simply tree-structured graphical models. The graph of an RSPN is not a conditional dependency graph, but a graphical representation of the computation of the partition function. A ground RSPN can be converted into an equivalent graphical model, but the resulting model may be high-treewidth, and computationally intractable as such.

In effect, RSPNs are a way to compactly represent context-specific independences in relational domains: different children of a sum node may have different variable decompositions in their product nodes. These context-specific independences are what give RSPNs more expressiveness than low-treewidth tractable models.

Learning RSPNs

The learning task for RSPNs is to determine the structure and parameters of all the class SPNs in the domain. In this work, we assume that the part relationships among classes are known, i.e. the user determines what types of unique and

Algorithm 1 LearnRSPN(C, T, V)

input: C, a class
T, a set of instances of C
V, a set of attributes, relation aggregates, and parts
output: an RSPN for class C
if $|V| = 1$ **then**
  if $v \in V$ is an attribute **then**
    return univariate estimated from v's values in T
  else if v is a relation aggregate **then**
    return EDT estimated from v's values in T
  else
    $C_{child} \leftarrow$ class of v *//v is a part*
    $T_{child} \leftarrow$ parts of $t \in T$ of type C_{child}
    $V_{child} \leftarrow$ attributes, relations and parts of C_{child}
    return LearnRSPN($C_{child}, T_{child}, V_{child}$)
  end if
else
  partition V into approximately independent subsets V_j
  if success **then**
    return $\prod_j$ LearnRSPN(C, T, V_j)
  else
    partition T into sets of similar subsets T_i
    return $\sum_i \frac{|T_i|}{|T|}$.LearnRSPN(C, T_i, V)
  end if
end if

exchangeable parts are allowed for each class. The input for the learning algorithm is a set of part decompositions, and an evidence database specifying the values of the attributes and relations among the objects in those decompositions.

Our learning algorithm is based on the top-down LearnSPN algorithm (Gens and Domingos 2013). LearnSPN(T, V) is a propositional SPN learning algorithm, and takes as input a set of training instances T and variables V. The algorithm attempts to decompose V into independent subsets $V_1, \ldots, V_k$ (using pairwise statistical independence tests); if such a decomposition exists, LearnSPN recurses over each set (calling LearnSPN(T, V_1),. . .,LearnSPN(T, V_k)), and returns a product node over the recursively learned sub-SPNs. If V does not decompose, LearnSPN instead clusters the instances T, recursively learns a sub-SPN over each subset $T_1, \ldots, T_k$, and returns a sum-node over the sub-SPNs, weighted by the mixture proportions. Under certain assumptions, LearnSPN can be seen as a greedy search maximizing the likelihood of the learned SPN.

Given an RSPN class C, LearnSPN could be used directly to learn an SPN over C's attributes. However, C's exchangeable parts pose a problem for LearnSPN: the number of leaf variables in the ground SPN can differ from one training instance to another, and between training instances and test instances. To address this, we propose the LearnRSPN algorithm (Alg. 1), a relational extension of LearnSPN. (For the purpose of this discussion, parts are assumed to be exchangeable; attributes of unique parts can be handled the same way as attributes of the parent part, and represented as separate leaves in the class SPN.)

LearnRSPN is a top-down algorithm similar to LearnSPN; it attempts to find independent subsets from among the object's set of attributes, relations and parts; if multiple subsets exist, the algorithm learns a sub-SPN over each subset, and returns a product over the sub-SPNs. If independent subsets cannot be found, LearnRSPN instead clusters the instances, returning a sum node over the components, weighted by the mixture proportions.

LearnRSPN exploits the fact that predicates involving exchangeable parts are grounded into finite sets of exchangeable variables. Instead of treating each ground atom as a separate leaf in the SPN, LearnRSPN summarizes a set of exchangeable variables with an aggregate statistic (in our experiments, we used the fraction of true variables in the set, though other statistics can be used). This summary statistic is treated as a single variable in the decomposition stage of RSPN. Thus, attributes that are highly predictive of the statistics of the groundings of a relation will be grouped with that relation. Parts are similarly summarized by the statistics of their attribute and relation predicates.

The base case of LearnRSPN (when $|V| = 1$) varies depending on what v is. When v is an attribute, the RSPN to be returned is simply a univariate distribution over the attribute, as in the propositional version of LearnSPN. When v is an aggregate over an exchangeable relation, the RSPN to be returned is an EDT over the relation. The final base case is when v is a part of class C_{child}. In this case, LearnRSPN returns an SPN for class C_{child}. Crucially, different SPNs are learned for C_{child} in different children of a sum node in parent class C (since the recursive call is made with a different set of instances).

Like LearnSPN, LearnRSPN can be seen as an algorithm schema rather than a single algorithm; the user is free to choose a clustering algorithm for instances, a dependency test for variable splitting, an aggregate statistic, and a family of EDTs for exchangeable relations. Note that different families of EDTs may require different aggregate statistics for parameter estimation. The fraction of true groundings is sufficient for the two EDTs described in examples 1 and 2.

Evaluation

Methodology

We compared a Python implementation of LearnRSPN to two MLN structure learning algorithms:

- MSL (Kok and Domingos 2005), as implemented in the widely-used ALCHEMY system (Kok et al. 2008);
- LSM (Kok and Domingos 2010), a state-of-the-art MLN learning method.

We also evaluated a simple baseline (BL) that simply predicts each atom according to the global marginal probability of its predicate.

To cluster instances in LearnRSPN, we used the EM implementation in SCIKIT-LEARN (Pedregosa et al. 2011), with two clusters. To test independence, we fit a Gaussian distribution (for aggregate variables) or Bernoulli distribution (for binary attributes), and computed the pairwise mutual information (MI) between the variables. The test statistic

```
class Area:
  exchangeable part Group

class Group:
  unique part Professor
  exchangeable part Student
  exchangeable part GroupPaper
  exchangeable part NonGroupPaper
  relation Author(Professor, GroupPaper)
  relation Author(Professor, NonGroupPaper)
  relation Author(Student, GroupPaper)
  relation Author(Student, NonGroupPaper)

class Professor:
  attribute Position_Faculty
  attribute Position_Adjunct
  attribute Position_Affiliate

class Student:
  attribute InPhase_PreQuals
  attribute InPhase_PostQuals
  attribute InPhase_PostGenerals
```

Figure 4: Part structure for UW-CSE domain.

used was $G = 2N \times MI$ (N being the number of samples), which in the discrete case is equivalent to the G-test used by Gens and Domingos (2013). In this case, G's distribution is approximately chi-square. To discourage excessively fine-grained decomposition during structure learning, we used a high threshold of 0.5 for the one-tailed p-value. For EDTs, we used the independent Bernoulli form, as described in example 1 in the main paper. All Bernoulli distributions were smoothed with a pseudocount of 0.1.

For MLN inference, we used the MC-SAT algorithm, the default choice in ALCHEMY 2.0, with the default parameters. For LSM, we used the example parameters in the implementation ($N_{walks} = 10,000$, $\pi = 0.1$; remaining parameters as specified by Kok and Domingos 2010).

We report results in terms of area under the precision-recall curve (AUC; Davis and Goadrich 2006) and the average conditional marginal log-likelihood (CMLL) of test atoms. AUC is a prediction quality measure that is insensitive to the fraction of true negative atoms. CMLL directly measures the quality of the probability estimates. For LearnRSPN, we also report test set log-likelihood (LL) normalized by the number of queries, as an alternate measure of prediction quality. (ALCHEMY does not compute this quantity, since it is intractable for MLNs.) Unlike CMLL, LL captures the joint likelihood, rather than just the individual marginal likelihoods.

UW-CSE

The UW-CSE database (Richardson and Domingos 2006) has been used to evaluate a variety of statistical relational learning algorithms. The dataset describes the University of Washington Computer Science & Engineering department, and includes advising relationships, paper authorships, etc. The database is divided into five non-overlapping mega-examples, by research area.

To generate a part structure for this domain (fig. 4), we separated the people into one research group per faculty member, with students determined using the $AdvisedBy$ and $TempAdvisedBy$ predicates (breaking ties by number of coauthored papers). Publications are also divided among groups: each paper is assigned to the group of the professor who wrote it, voting by the number of student authors in the group in the event of a tie. The prediction tasks are to infer the roles of faculty (Professor, Associate Professor or Assistant Professor) and students (Pre-Quals, Post-Quals, Post-Generals), as well as paper authorships. The part structure is also made available to ALCHEMY in the form of predicates $Has(Area, Group)$, $Has(Group, Professor)$, $Has(Group, Student)$, $Has\_Group(Group, Paper)$, and $Has\_NonGroup(Group, Paper)$.

We performed leave-one-out testing by area, testing on each area in turn using the model trained from the remaining four. 80% of the groundings of the query predicates were provided as evidence, and the task was to predict the remaining atoms. Table 1 shows the results on all five areas, and the average. The RSPN approach is orders of magnitude faster than the other systems, and significantly more accurate. Average training times in this domain are 9s (RSPN), 14,094s (MSL) and 1,620s (LSM).

Social Network Link Prediction

Link prediction is a challenging statistical relational learning problem (Popescul and Ungar 2003; Taskar et al. 2003). The task is to predict missing links in a partially observed graph, taking into account observed attributes of the nodes.

We generated artificial social networks in the Friends-and-Smokers domain (Singla and Domingos 2008), using a generalization of the Barabási-Albert preferential attachment model (Barabási and Albert 1999). A network with N nodes is generated as follows:

- For some fraction $p_{smokes} = 0.3$ of nodes x, set $Smokes(x)$ to 'true', and set the remainder to 'false'.
- For each node, sample another node and create an undirected edge. In the basic Barabási-Albert model, the probability of choosing a node is proportional to its degree. To encourage homophily, we multiply the unnormalized probability of an edge by a factor of $h_{smokes,smoker} = 100$ for smoker-smoker edges, and $h_{nonsmoker,nonsmoker} = 10$ for edges between non-smokers.
- Iterate, creating more edges for each node using the above distribution. We generate 2 or 3 edges (with equal probability) for each smoker node, and 1 or 2 edges for each non-smoker node.

This procedure results in graphs with small, dense communities of smokers, sparser communities of non-smokers, and relatively few links between smokers and non-smokers (fig. 5).

For training data, we generated five 100-person graphs. We tested on graphs of size ranging from 100 to 400 nodes

Table 1: UW-CSE results. $|Q|$ is the number of query atoms.

Area	$\|Q\|$	Inference time (s)			AUC-PR				CMLL				LL/$\|Q\|$
		RSPN	MSL	LSM	RSPN	MSL	LSM	BL	RSPN	MSL	LSM	BL	RSPN
AI	1,414	0.10	7.18	5.32	0.71	0.36	0.29	0.36	-0.10	-0.16	-0.17	-0.16	-0.09
Graphics	171	0.03	0.82	0.50	0.80	0.59	0.38	0.35	-0.13	-0.28	-0.31	-0.26	-0.13
PL	17	0.01	0.05	0.05	0.81	0.84	0.69	0.42	-0.46	-0.81	-0.84	-0.80	-0.45
Systems	1,120	0.01	8.81	4.33	0.75	0.76	0.25	0.28	-0.07	-0.08	-0.14	-0.14	-0.07
Theory	308	0.04	1.01	0.94	0.79	0.51	0.37	0.32	-0.11	-0.29	-0.21	-0.20	-0.10
Average	606	**0.05**	3.57	2.22	0.77	0.61	0.39	0.34	**-0.17**	-0.32	-0.33	-0.31	-0.17

Table 2: Friends & Smokers link prediction results. N is the number of people in the network.

N	$\|Q\|$	Inference time (s)			AUC-PR				CMLL				LL/$\|Q\|$
		RSPN	MSL	LSM	RSPN	MSL	LSM	BL	RSPN	MSL	LSM	BL	RSPN
100	2,000	**0.10**	25.31	8.11	**0.22**	0.08	0.02	0.02	**-0.09**	-0.13	-0.13	-0.12	-0.09
200	8,000	**0.34**	202.14	51.29	**0.16**	0.03	0.01	0.01	**-0.05**	-0.54	-0.06	-0.07	-0.05
300	18,000	**0.75**	740.64	150.13	**0.13**	0.01	0.00	0.00	**-0.04**	-3.24	-0.04	-0.06	-0.03
400	32,000	**1.29**	1753.14	330.92	**0.13**	0.00	0.00	0.00	**-0.03**	-6.10	-0.04	-0.05	-0.02

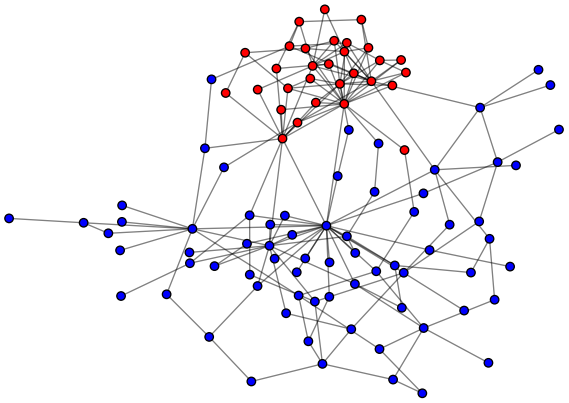

Figure 5: Sample 100-node social network. Red nodes are smokers.

(10,000 to 160,000 possible edges), to evaluate how well the structures learned by LearnRSPN generalize to mega-examples of different size.

At test time, all the $Smoker$ labels and 80% of the $Friendship$ edges are known; the prediction task is to infer the marginal probabilities of the remaining 20% of the graph edges.

An RSPN was trained with two classes: $Network$ and $Person$. $Network$ has both classes as exchangeable subparts. The part decompositions were generated using the Louvain method[1] (Blondel et al. 2008), and the resulting community structure was also made available to ALCHEMY in the form of $Has(Network, Network)$ and $Has(Network, Person)$ predicates.

Table 2 shows the inference time and accuracy of the three systems. Results are averaged over five runs. Figures in bold are statistically significant improvements over all other systems (using the sign test, with a p-value of 0.05). Average training times for the three systems are 168s (RSPN), 31,083s (MSL) and 105,018s (LSM).

This is an extremely challenging link prediction task, due to the sparsity of the domain, and the relatively weak dependence between a node's attributes and links. MSL's greedy structure learning fails to find formulas that either exploit the provided community structure or capture the relationship between a person's friendships and smoking habits. Additionally, the weights learned by MSL on 100-node networks become increasingly inappropriate as the network size changes. On larger graphs, MLNs become prone to predicting that everybody is friends, a known pathology in social network models. Nevertheless, MSL outperforms the baseline in the AUC metric, which corrects for sparsity. LSM avoids the above pathology, learning a model similar to the baseline. As in the UW-CSE domain, RSPNs greatly outperform the other systems in both speed and accuracy.

Automated Debugging

We applied RSPNs to a fault localization problem. The task is to predict the location of the bug in a faulty program.

The test corpus consists of four short Python programming assignments from MIT edX introductory programming course (6.00x) (Singh, Gulwani, and Solar-Lezama 2013): `oddTuples`, `derivatives`, `isWordGuessed` and `newtons_method`. We developed a suite of ten unit tests for each assignment, and identified ten buggy but syntactically valid responses to each question. (We filtered out submissions with multiple bugs, non-terminating loops, etc.) We manually annotated the location of the bug in each of the 40 programs in the corpus.

The corpus included several other programs, which were unusable for the following reasons:

- `ate1` and `biggest` had a single example each.
- `polynomials` and `getAvailableWords` predominantly contained syntactic errors; we did not find examples that met the constraints above.

[1] http://perso.crans.org/aynaud/communities/

Table 3: Fault localization results.

Program	Avg. LOC	Inference time (s)		Fraction skipped				CMLL				LL/$\lvert Q \rvert$
		RSPN	MSL	RSPN	MSL	TAR	BL	RSPN	MSL	TAR	BL	RSPN
oddTuples	10.9	0.004	0.030	0.66	0.63	0.80	0.58	-0.65	-3.65	-1.47	-0.74	-0.64
derivatives	15.3	0.005	0.055	0.66	0.54	0.53	0.47	-0.36	-4.31	-0.62	-0.42	-0.35
isWordGuessed	15.8	0.004	0.056	0.70	0.58	0.63	0.42	-0.49	-3.35	-0.57	-0.40	-0.41
newtons_method	22.1	0.007	0.077	0.54	0.74	0.51	0.47	-0.41	-2.69	-0.80	-0.37	-0.41
Average	16.5	0.005	0.055	0.64	0.62	0.62	0.48	-0.60	-3.50	-0.86	-0.48	-0.45

```
class Program:
  exchangeable part IfStmt
  exchangeable part LoopStmt
  exchangeable part AtomicStmt

class IfStmt:
  unique part Program
  attribute Buggy

class LoopStmt:
  unique part Program
  attribute Buggy

class AtomicStmt:
  attribute Buggy
```

Figure 6: Part structure for debugging domain.

- `hangman` and `simple_hangman` were interactive programs, incompatible with our automated testing environment.
- The predominant cause of failure in `getGuessedWord` was that the output string was formatted differently from our reference implementation.

The program parse tree provides the part structure. The parse tree is mapped to the part structure in fig. 6; this part structure is also provided to ALCHEMY as evidence. The aggregation used by LearnRSPN is simply a discrete variable indicating which class of statement is most common in the subprogram (`IfStmt`, `LoopStmt`, or `AtomicStmt`). This information is also provided as evidence to ALCHEMY. The systems were trained on three programs and tested on the fourth; reported results are averaged over the 10 buggy versions of each program.

In addition to MSL, we compared RSPNs to TARANTULA (TAR; Jones and Harrold 2005), a well-established fault localization approach. A common evaluation metric for fault localization systems is the fraction of program lines ranked lower than the buggy line. Higher scores indicate better localization. Ties in the line ranking are broken randomly. TARANTULA scores are computed in closed form from the coverage matrix. We also report the CMLL; for TARANTULA, this was computed by treating the suspiciousness score (which falls between 0 and 1) as a probability. As seen in table 3, RSPNs are competitive with both MSL and TARANTULA in ranking quality, and outperform them in CMLL. Average training times are 0.37s (RSPN) and 188s (MSL).

Discussion

SRL algorithms have been successfully applied to several problems, but the difficulty, cost and unreliability of approximate inference has limited their wider adoption. In practice, applying SRL methods to a new domain requires substantial engineering effort in choosing and configuring the approximate learning and inference algorithms. The expressiveness of languages like Markov logic is both a boon and a curse: although these languages can compactly represent sophisticated probabilistic models, they also make it easy for practitioners to unintentionally design models too complex even for state-of-the-art inference algorithms.

In the propositional setting, several approaches have been recently proposed for learning high-treewidth tractable models (Lowd and Domingos 2008, Gogate et al. 2010, Poon and Domingos 2011). To our knowledge, LearnRSPN is the first algorithm for learning high-treewidth tractable relational models. Empirically, LearnRSPN outperforms conventional statistical relational methods in accuracy, inference time and training time.

A limitation of RSPNs is that they require a known, fixed part decomposition for all training and test mega-examples. Applying RSPNs to a new domain does require the user to specify the part decomposition; this is analogous to specifying the relational structure in PRMs. Many domains have a natural part structure that can be exploited (like the UW-CSE and debugging domains); in other cases, part structure can be created using existing graph-cut or community detection algorithms (as in our link prediction experiments). An important direction for future work is to develop an efficient, principled method of finding part structure in a database.

Acknowledgments

This research was partly funded by ARO grant W911NF-08-1-0242, ONR grants N00014-13-1-0720 and N00014-12-1-0312, and AFRL contract FA8750-13-2-0019. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO, ONR, AFRL, or the United States Government.

References

Amer, M. R., and Todorovic, S. 2012. Sum-product networks for modeling activities with stochastic structure. In *Proceedings of CVPR*.

Bach, F., and Jordan, M. I. 2001. Thin junction trees. In *Advances in NIPS*.

Barabási, A. L., and Albert, R. 1999. Emergence of scaling in random networks. *Science* 286:509–512.

Blondel, V. D.; Guillaume, J.; Lambiotte, R.; and Lefebvre, E. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 10:10008–10019.

Bröcheler, M.; Mihalkova, L.; and Getoor, L. 2010. Probabilistic similarity logic. In *Proceedings of UAI*.

Davis, J., and Goadrich, M. 2006. The relationship between precision-recall and ROC curves. In *Proceedings of ICML*.

Davis, J.; Ong, I.; Struyf, J.; Burnside, E.; Page, D.; and Costa, V. S. 2007. Change of representation for statistical relational learning. In *Proceedings of IJCAI*.

Delalleau, O., and Bengio, Y. 2011. Shallow vs. deep sum-product networks. In *Advances in NIPS*.

Dennis, A., and Ventura, D. 2012. Learning the architecture of sum-product networks using clustering on variables. In *Advances in NIPS*.

Diaconis, P., and Freedman, D. 1980. De Finetti's generalizations of exchangeability. *In Studies in Inductive Logic and Probability* 2:235–250.

Domingos, P., and Webb, A. 2012. A tractable first-order probabilistic logic. In *Proceedings of AAAI*.

Flach, P., and Lachiche, N. 2004. Naive Bayesian classification of structured data. *Machine Learning* 57(3):233–269.

Friedman, N.; Getoor, L.; Koller, D.; and Pfeffer, A. 1999. Learning probabilistic relational models. In *Proceedings of IJCAI*.

Gens, R., and Domingos, P. 2012. Discriminative learning of sum-product networks. In *Advances in NIPS*.

Gens, R., and Domingos, P. 2013. Learning the structure of sum-product networks. In *Proceedings of ICML*.

Getoor, L., and Taskar, B., eds. 2007. *Introduction to Statistical Relational Learning*. MIT Press.

Gogate, V.; Webb, W. A.; and Domingos, P. 2010. Learning efficient Markov networks. In *Advances in NIPS*.

Jones, J. A., and Harrold, M. J. 2005. Empirical evaluation of the TARANTULA automatic fault-localization technique. In *Proceedings of ASE*.

Kok, S., and Domingos, P. 2005. Learning the structure of Markov logic networks. In *Proceedings of ICML*.

Kok, S., and Domingos, P. 2010. Learning markov logic networks using structural motifs. In *Proceedings of ICML*.

Kok, S.; Sumner, M.; Richardson, M.; Singla, P.; Poon, H.; Lowd, D.; Wang, J.; and Domingos, P. 2008. The Alchemy system for statistical relational AI. Technical report, University of Washington. http://alchemy.cs.washington.edu.

Landwehr, N.; Kersting, K.; and De Raedt, L. 2005. nFOIL: Integrating Naive Bayes and FOIL. In *Proceedings of AAAI*.

Lowd, D., and Domingos, P. 2008. Learning arithmetic circuits. In *Proceedings of UAI*.

Natarajan, S.; Khot, T.; Kersting, K.; Gutmann, B.; and Shavlik, J. 2012. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning* 86(1):25–56.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Peharz, R.; Geiger, B. C.; and Pernkopf, F. 2013. Greedy part-wise learning of sum-product networks. In *Proceedings of ECML-PKDD*.

Poon, H., and Domingos, P. 2011. Sum-product networks: A new deep architecture. In *Proceedings of UAI*.

Popescul, A., and Ungar, L. H. 2003. Structural logistic regression for link analysis. In *Proceedings of 2nd International Workshop on Multi-Relational Data Mining*.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62:107–136.

Sato, T., and Kameya, Y. 2008. New advances in logic-based probabilistic modeling by PRISM. In *Proceedings of PILP*.

Singh, R.; Gulwani, S.; and Solar-Lezama. 2013. Automated feedback generation for introductory programming assignments. In *Proceedings of SIGPLAN*.

Singla, P., and Domingos, P. 2008. Lifted first-order belief propagation. In *Proceedings of AAAI*.

Taskar, B.; Wong, M. F.; Abbeel, P.; and Koller, D. 2003. Link prediction in relational data. In *Advances in NIPS*.

Webb, W. A., and Domingos, P. 2013. Tractable probabilistic knowledge bases with existence uncertainty. In *StaR-AI*.