Noname manuscript No.
(will be inserted by the editor)

Lifted Graphical Models: A Survey

Angelika Kimmig · Lilyana Mihalkova · Lise Getoor

Received: date / Accepted: date

Abstract Lifted graphical models provide a language for expressing dependencies between different types of entities, their attributes, and their diverse relations, as well as techniques for probabilistic reasoning in such multi-relational domains. In this survey, we review a general form for a lifted graphical model, a par-factor graph, and show how a number of existing statistical relational representations map to this formalism. We discuss inference algorithms, including lifted inference algorithms, that efficiently compute the answers to probabilistic queries over such models. We also review work in learning lifted graphical models from data. There is a growing need for statistical relational models (whether they go by that name or another), as we are inundated with data which is a mix of structured and unstructured, with entities and relations extracted in a noisy manner from text, and with the need to reason effectively with this data. We hope that this synthesis of ideas from many different research groups will provide an accessible starting point for new researchers in this expanding field.

Keywords statistical relational learning · probabilistic reasoning · first-order probabilistic models · probabilistic programming · structured graphical models · collective classification · joint reasoning · par-factor graphs ·

A. Kimmig
Department of Computer Science
KU Leuven, Belgium
E-mail: angelika.kimmig@cs.kuleuven.be

L. Mihalkova
Google
340 Main Street, Los Angeles, CA
E-mail: lilyanam@alumni.cs.utexas.edu

L. Getoor
Computer Science Department
University of California, Santa Cruz
E-mail: getoor@soe.ucsc.edu

templated graphical models · Markov logic networks · probabilistic relational models · relational Markov networks · probabilistic soft logic · FACTORIE · Bayesian logic programs · relational Bayesian networks · BLOG · lifted inference · parameter estimation · structure learning

1 Motivation and Scope

Multi-relational data, in which entities of different types engage in a rich set of relations, is ubiquitous in many domains of current interest. For example, in social network analysis the entities are individuals who relate to one another via friendships, family ties, or collaborations; in computational biology, one is frequently interested in modeling how a set of chemical substances, the entities, interact with, inhibit, or catalyze one another; in web and social media applications, a set of users interact with each other and with a set of web pages or other online resources, which may themselves be related via hyperlinks; in natural language processing tasks, it is often necessary to reason about the relationships between documents, or words within a sentence or a document. There is thus a need for formalisms that can model such multi-relational data and for corresponding reasoning algorithms that allow one to infer additional information. Furthermore, regularities in these domains are often hard to identify manually, and methods that automatically learn them from data are thus desirable. Indeed, by incorporating such relational information into learning and reasoning, rather than relying solely on entity-specific attributes, it is usually possible to achieve higher predictive accuracy for an unobserved entity attribute. For example, exploiting hyperlinks between web pages can improve webpage classification accuracy, and taking into account both individual attributes of users and relationships between users can improve inference of demographic attributes in social networks. Developing algorithms and representations that can effectively deal with relational information is important also because in many cases it is necessary to predict the existence of a relation between the entities. For example, in an online social network application, one may be interested in predicting friendship relations between people in order to suggest new friends to the users; in molecular biology domains, researchers may be interested in predicting how newly-developed substances interact.

While multi-relational data has long been considered in relational learning, multi-relational data mining and inductive logic programming (De Raedt, 2008; Muggleton and De Raedt, 1994; Muggleton, 1991, 1992; De Raedt, 1996; Džeroski and Lavrač, 2001; Lavrač and Džeroski, 1993), these techniques do not address the inherent uncertainty present in many application domains. This limitation is overcome by explicitly modeling both relational and probabilistic aspects, an approach pursued by the field of statistical relational learning (SRL) (e.g., Dietterich et al, 2004; Fern et al, 2006; Getoor and Taskar, 2007; Domingos and Kersting, 2009; Kersting et al, 2010b; Kautz et al, 2012; Gogate et al, 2013), which has recently experienced significant growth. A closely related field that also relies on both relational data and probabilis-

tic information is structured prediction (Bakir et al, 2007; Lafferty et al, 2001; Tsochantaridis et al, 2004; Munoz et al, 2009; Weiss and Taskar, 2010), and especially collective classification (Jensen et al, 2004; Macskassy and Provost, 2007; Wu and Schölkopf, 2007; Sen et al, 2008b; Kuwadekar and Neville, 2011; London and Getoor, 2013).

This survey provides a detailed overview of developments in the field of SRL. We limit our discussion to *lifted graphical models* (also referred to as templated graphical models), that is, to formalisms that use relational languages to define graphical models, where we use Poole's par-factors (Poole, 2003) as the unifying language. As their propositional counterparts, lifted graphical models take advantage of independencies between random variables to compactly represent probability distributions by factorizing them. In the same way as first-order logic *lifts* propositional logic by making statements about all members of groups of objects represented by logical variables, lifted graphical models define random variables and their correlations on the level of groups of objects of the same type rather than for each individual object. Furthermore, all members of such a group use the same *tied* parameters in the graphical model, making it possible to define probabilistic models over flexible numbers of objects with a fixed number of parameters. Lifted graphical models thus exploit the structure of both the relational domain and the probability distribution when representing probabilistic models. Because of the great variety of existing SRL applications, we cannot do justice to all of them; therefore, the focus is on representations and techniques, and applications are mentioned in passing where they help illustrate our point.

By limiting the scope of the survey, we are able to provide a more focused and unified discussion of the representations that we do cover, but also omit several important SRL representations, such as stochastic logic programs (Muggleton, 1996) and ProbLog (De Raedt et al, 2007). These formalisms are representatives of the second main stream of SRL research that focuses on extending logic-based representations and techniques to take into account uncertainty. For more information on this type of representations, we refer the reader to De Raedt and Kersting (2003); De Raedt and Kersting (2004); De Raedt et al (2008); De Raedt and Kersting (2010). An overview of the development of first-order probabilistic models over time is provided by de Salvo Braz et al (2008). Among the many other approaches in machine learning and other fields that consider relational data and models and that we do not consider here are lifted (PO)MDPs and relational reinforcement learning (van Otterlo, 2009), probabilistic databases (Suciu et al, 2011), probabilistic programming (Roy et al, 2008; Mansinghka et al, 2012), (multi-)relational Gaussian processes (Chu et al, 2006; Xu et al, 2009), relational LDA (Chang and Blei, 2009), mixed membership models (Airoldi et al, 2008), relational and graph SVMs (Tsochantaridis et al, 2004; Gaudel et al, 2007) and relational PCA (Li et al, 2009).

This survey is structured as follows. In Section 2, we define SRL and introduce preliminaries. In Section 3, we describe several SRL representations that are based on lifting a graphical model. Our goal in this section is to establish

a unified view on the available representations by adopting a generic, or template, SRL model – a par-factor graph – and discussing how particular models implement its various aspects. In this way, we establish not just criteria for comparisons of the models, but also a common framework in which to discuss inference (Section 4), parameter learning (Section 5.1), and structure learning (Section 5.2) algorithms.

2 Preliminaries

This section summarizes the key characteristics of SRL and provides background on both graphical models and relational representations as relevant for the rest of the article.

2.1 What is SRL?

Statistical relational learning (SRL) studies knowledge representations and their accompanying inference and learning techniques that allow for efficient modeling and reasoning in noisy and uncertain multi-relational domains. In classical machine learning settings, the data consists of a single table of feature vectors, one for each entity in the data. A crucial assumption made is that the entities in the data represent *independent and identically distributed (i.i.d.)* samples from the general population. In contrast, multi-relational domains contain entities of potentially different types that engage in a variety of relations. Thus, a multi-relational domain can be seen as consisting of several tables: a set of attribute tables that contain feature-vector descriptions for entities of a certain type, and a set of relationship tables that establish relationships among two or more of the entities in the domain. Relations also allow one to model complex, structured objects. As a consequence of the relationships among the entities, they are no longer independent, and the i.i.d. assumption is violated. Figure 1 shows a small example with two types of entities and one relation in a publication domain. A further characteristic of multi-relational domains is that they are typically noisy or uncertain. For example, there frequently is uncertainty regarding the presence or absence of a relation between a particular pair of entities. Finally, aggregation functions are a useful concept in relational domains, as they allow one to consider properties of all entities participating in a certain relation, e.g., all authors of a given paper, without the need to make assumptions on the number of such entities.

To summarize, an effective SRL representation needs to support the following two essential aspects: a) it needs to provide a language for expressing dependencies between different types of entities, their attributes, and their diverse relations; and b) it needs to allow for probabilistic reasoning in a potentially noisy environment.

Publication			
paper	title	venue	year
p1	t1	v1	y1
p2	t2	v2	y1
p3	t3	v1	y2
p4	t4	v1	y2

Researcher		
person	name	affiliation
r1	n1	a1
r2	n2	a1
r3	n3	a2
r4	n4	a3

Author	
paper	person
p1	r1
p1	r3
p2	r3
p3	r2
p3	r3
p4	r1
p4	r4

Fig. 1: Example database describing a publication domain, with attribute tables for publications and researchers, and a relationship table connecting the two.

Table 1: Notation used throughout this survey

Concept	**Representation**
Parameterized random variable (par-RV)	Sans serif upper-case letters X, Y, ...
Vector of par-RVs	Bold sans serif upper-case letters $\mathbf{\mathsf{X}}$, $\mathbf{\mathsf{Y}}$, ...
Random variable (RV)	Upper-case letters X, Y, ...
Vector of RVs	Bold upper-case letters $\boldsymbol{X}$, $\boldsymbol{Y}$, ...
Value assigned to RV	Lower-case letters x, y, ...
Vector of values assigned to RVs	Bold lower-case letters $\boldsymbol{x}$, $\boldsymbol{y}$, ...
Logical variable	Typewriter upper-case letters $\mathtt{X}$, $\mathtt{Y}$, ...
Entity/constant	Typewriter lower-case letters $\mathtt{x}$, $\mathtt{y}$, ...
Factor-graph	$\langle \boldsymbol{X}, \boldsymbol{F} \rangle$
Par-factor	$\Phi = (\mathbf{A}, \phi, \boldsymbol{\mathcal{C}})$
Par-factor graph	$\boldsymbol{\mathcal{F}} = \{\Phi_1, \ldots, \Phi_n\}$
Set of constraints	$\boldsymbol{\mathcal{C}}$
Hypothesis space	$\boldsymbol{\mathcal{H}}$
Training data / set of training examples	$\boldsymbol{\mathcal{D}}$
Instances of par-factor $\Phi = (\mathbf{A}, \phi, \boldsymbol{\mathcal{C}})$	$\boldsymbol{\mathcal{I}}(\Phi) = \{\boldsymbol{A} \mid \boldsymbol{A}$ instance of $\mathbf{A}$ under $\boldsymbol{\mathcal{C}}\}$
Set of integer linear program variables	$\boldsymbol{\mathcal{V}}$

2.2 Background and Notation

Lifted graphical models combine ideas from graphical models and relational languages. We first summarize key concepts of graphical models and establish the notation and terminology to be used in the rest of this survey. Probability theory and first-order logic sometimes use the same term to describe different concepts. For example, the word "variable" could mean a random variable (RV), or a logical variable. To avoid confusion, we distinguish between different meanings using different fonts, as summarized in Table 1. Also, depending on context, the word "model" may denote a specification of a probability distribution in a generic sense (e.g., when talking about directed and undirected graphical models), a formalism to define such models (such as the languages discussed in Sections 3.2 and 3.3), or a specific encoding of a distribution in such a formalism.

2.2.1 Probabilistic Graphical Models

As lifted graphical models extend probabilistic graphical models, we first summarize basic concepts from that area. For a detailed introduction to graphical models, we refer the reader to Koller and Friedman (2009). We discuss *factor graphs*, which are the propositional counterpart of the par-factor graphs used as a unifying language in this survey, cf. Section 3.1, as well as *Markov networks* and *Bayesian networks* as representatives of *undirected* and *directed* graphical models, respectively, a distinction that will come back at the lifted level, cf. Sections 3.2 and 3.3.

In general, to describe a probability distribution on n binary RVs, one needs to store $2^n - 1$ parameters, one for each possible configuration of value assignments to the RVs. However, sets of RVs are often conditionally independent of one another, and thus, many of the parameters will be repeated. To avoid such redundancy of representation, several graphical models have been developed that explicitly represent conditional independencies. One of the most general representations is the factor graph (Kschischang et al, 2001). *A factor graph* consists of a tuple $\langle \boldsymbol{X}, \boldsymbol{F} \rangle$, where $\boldsymbol{X}$ is a set of RVs, and $\boldsymbol{F}$ is a set of factors, each of which is a function from the values of (a subset of) $\boldsymbol{X}$ to the non-negative real numbers. It is typically drawn as an undirected bipartite graph (cf. Figure 2b). The two partitions of vertices in the factor graph consist of the RVs X in $\boldsymbol{X}$ (drawn as circular nodes) and the factors ϕ in $\boldsymbol{F}$ (drawn as square nodes), respectively. There is an edge between an RV X and a factor ϕ if and only if X is necessary for the computation of ϕ (cf. Figure 2c); i.e., each factor is connected to its arguments. As a result, the structure of a factor graph defines conditional independencies between the variables. In particular, a variable is conditionally independent of all variables with which it does not share factors, given the variables with which it participates in common factors.

A factor graph $\langle \boldsymbol{X}, \boldsymbol{F} \rangle$ defines a probability distribution over $\boldsymbol{X}$ as follows. Let $\boldsymbol{x}$ be a particular assignment of values to $\boldsymbol{X}$. Then,

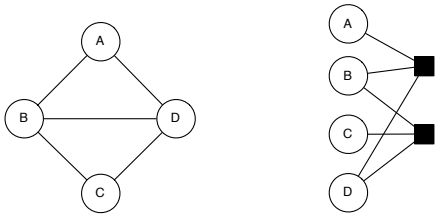
$$P(\boldsymbol{X} = \boldsymbol{x}) = \frac{1}{Z} \prod_{\phi \in \boldsymbol{F}} \phi(\boldsymbol{x}_\phi). \tag{1}$$

Above, $\boldsymbol{x}_\phi$ represents the values of those variables in $\boldsymbol{X}$ that are necessary for computing ϕ's value. Z is a normalizing constant that sums over all possible value assignments $\boldsymbol{x'}$ to $\boldsymbol{X}$, and is given by:

$$Z = \sum_{\boldsymbol{x'}} \prod_{\phi \in \boldsymbol{F}} \phi(\boldsymbol{x}'_\phi). \tag{2}$$

As before, $\boldsymbol{x}'_\phi$ represents the values of only those variables in $\boldsymbol{X}$ that are necessary to compute ϕ.

Factor graphs are a general representation for graphical models that subsumes both Markov networks and Bayesian networks, two very common types



(a) (b)

A	B	D	$\phi_1(A,B,D)$
T	T	T	5
T	T	F	1
T	F	T	1
T	F	F	5
F	T	T	1
F	T	F	5
F	F	T	5
F	F	F	1

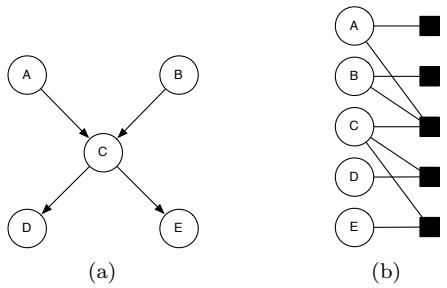
B	C	D	$\phi_2(B,C,D)$
T	T	T	4
T	T	F	2
T	F	T	3
T	F	F	1
F	T	T	2
F	T	F	4
F	F	T	1
F	F	F	3

(c)

Fig. 2: Example of (a) Markov network structure, (b) corresponding factor graph, and (c) potential functions (all random variables are Boolean). Circular nodes correspond to variables, whereas square nodes correspond to factors.

of graphical models whose graphical representations use RVs as nodes only, and implicitly provide the factors through the graph structure.

A *Markov network* (Pearl, 1988) is an undirected graphical model whose nodes correspond to the random variables in $\boldsymbol{X}$. It computes the probability distribution over $\boldsymbol{X}$ as a product of strictly positive potential functions defined over cliques in the graph, i.e., for any set of variables that are connected in a maximal clique, there is a potential function that takes them as arguments. For instance, the Markov network in Figure 2a has two cliques, and thus two potential functions, one over variables A, B and D, the second over variables B, C and D, which are given in tabular form in Figure 2c. Alternatively, potential functions are often represented as a log-linear model, in which each potential function $\phi(X_1 \dots X_n)$ of n variables $X_1 \dots X_n$ is represented as an exponentiated product $\exp(\lambda \cdot f(X_1 \dots X_n))$. In this expression, $\lambda \in \mathbb{R}$ is a learnable parameter, and f is a feature that captures characteristics of the variables and can evaluate to any value in $\mathbb{R}$. In general, there may be more than one potential function defined over a clique. In this way, a variety of feature functions, each with its own learnable parameter λ, can be defined for the same set of variables. There is a direct mapping from Markov networks to factor graphs. To convert a Markov network to a factor graph, for each maximal clique in the Markov network, we include a factor that evaluates to the product of potentials defined over that clique. The factor graph corresponding to the Markov network in Figure 2a is shown in Figure 2b.



(a)

(b)

A=T	B=T
0.8	0.4

A	B	C=T
T	T	0.2
T	F	0.3
F	T	0.7
F	F	0.1

C	D=T
T	0.7
F	0.8

C	E=T
T	0.3
F	0.9

(c)

Fig. 3: Example of (a) Bayesian network structure, (b) corresponding factor graph, and (c) conditional probability tables defining potential functions (all random variables are Boolean). Circular nodes correspond to random variables, whereas square nodes correspond to factors.

A *Bayesian network* (Pearl, 1988) is represented as a directed acyclic graph, whose vertices again are the RVs in $\boldsymbol{X}$. Figure 3a shows an example. The probability distribution over $\boldsymbol{X}$ is specified by providing the conditional probability distribution for each node given the values of its parents. The simplest way of expressing these conditional probabilities is via conditional probability tables (CPTs), which list the probability associated with each configuration of values to the nodes, cf. Figure 3c. A Bayesian network can directly be converted to a factor graph as follows. For each node X, we introduce a factor ϕ_X to represent the conditional probability distribution of X given its parents. Thus, ϕ_X is computed as a function of only X and its parents. In this case, the product is automatically normalized, i.e., the normalization constant Z sums to 1. Figure 3b shows the factor graph corresponding to the Bayesian network of Figure 3a.

While factor graphs provide a general framework, it is often useful to restrict the discussion to either directed or undirected models. More specifically, directed models are appropriate when one needs to express a causal dependence, while undirected models are better suited to domains containing cyclic dependencies. On the other hand, by describing algorithms for factor graphs, they become immediately available to representations that can be viewed as specializations of factor graphs, such as Markov and Bayesian networks. In this survey, we therefore keep the discussion on the general level of factor graphs (or their lifted counterpart as introduced in Section 3.1) whenever possible, and only consider special cases where necessary.

2.2.2 Inference in Graphical Models

The two most common inference tasks in graphical models are *computing marginals* and *most probable explanation (MPE) inference*. The former computes the marginal probability distributions for a subset $\boldsymbol{X'} \subseteq \boldsymbol{X}$ of the random variables from the full joint distribution defined by the graphical model, cf. Equation (1). Setting $\boldsymbol{Y} = \boldsymbol{X} \setminus \boldsymbol{X'}$, this probability has the following form:

$$P(\boldsymbol{X'} = \boldsymbol{x'}) = \sum_{\boldsymbol{Y}=\boldsymbol{y}} \frac{1}{Z} \prod_{\phi \in \boldsymbol{F}} \phi(\boldsymbol{x'}_\phi, \boldsymbol{y}_\phi). \tag{3}$$

Computing the marginal probability thus corresponds to *summing out* the random variables $\boldsymbol{Y}$. It also is an important step in computing *conditional probabilities* $P(\boldsymbol{X'} = \boldsymbol{x'} | \boldsymbol{Y'} = \boldsymbol{y'}) = P(\boldsymbol{X'} = \boldsymbol{x'}, \boldsymbol{Y'} = \boldsymbol{y'}) / P(\boldsymbol{Y'} = \boldsymbol{y'})$, where values $\boldsymbol{y'}$ for some random variables $\boldsymbol{Y'} \subseteq \boldsymbol{Y}$ are given as *evidence*.

MPE inference computes the most likely joint assignment to a subset $\boldsymbol{X'} \subseteq \boldsymbol{X}$ of the random variables (sometimes also called MAP (maximum a posteriori) state), given values $\boldsymbol{y}$ of all remaining RVs $\boldsymbol{Y} = \boldsymbol{X} \setminus \boldsymbol{X'}$, that is

$$MPE(\boldsymbol{X'} = \boldsymbol{x'} | \boldsymbol{Y} = \boldsymbol{y}) = \text{argmax}_{\boldsymbol{X'}=\boldsymbol{x'}} P(\boldsymbol{X'} = \boldsymbol{x'}, \boldsymbol{Y} = \boldsymbol{y}) \tag{4}$$

Even though the complexity of solving these tasks is exponential in the worst case and thus intractable in general, cf. Koller and Friedman (2009), many instances occurring in practice can be solved efficiently. We next summarize two common inference techniques for graphical models, whose extensions to the lifted case will be discussed in Section 4.

2.2.3 Variable Elimination

One of the earliest and simplest algorithms for exact inference in factor graphs is variable elimination (VE) (Zhang and Poole, 1994; Poole and Zhang, 2003). Suppose we would like to compute the marginal probability distribution of a particular random variable X, as given in Equation (3). VE proceeds in iterations summing out all other random variables $\boldsymbol{Y}$ one by one, exploiting the fact that multiplication distributes over summation and ignoring the constant normalization factor $1/Z$ during computations. An ordering over the variables in $\boldsymbol{Y}$ is established, and in each iteration the next $Y \in \boldsymbol{Y}$ is selected, and the set of factors is split into two groups—the ones that contain Y and the ones that do not. The latter can be pulled out of the sum over the current variable Y. All factors containing Y are multiplied together and the results are summed, thus effectively eliminating (or *summing out*) Y from $\boldsymbol{Y}$. The efficiency of the algorithm is affected by the ordering over $\boldsymbol{Y}$ that was used; heuristics for selecting good orderings are available. In the end, the normalization constant Z is obtained by simply summing the results for all values x of X and results are normalized.

This algorithm can be adapted to find the MAP state, cf. Equation (4). This requires an argmax computation over the variables of interest $\boldsymbol{X'}$ rather

than a summation over all other variables $\boldsymbol{Y}$, but the structure of the problem is similar otherwise. As before, the algorithm imposes an ordering over the variables $\boldsymbol{X'}$ to be processed, and proceeds in iterations, this time, however, eliminating each variable $X \in \boldsymbol{X'}$ by *maximizing* the product of all factors that contain X and remembering which value of X gave the maximum value.

2.2.4 Belief Propagation

Pearl's belief propagation (BP) (Pearl, 1988) is another algorithm for computing marginals in Bayesian networks. As the closely related forward-backward algorithm for hidden Markov models (Rabiner, 1989), BP is an instance of the sum-product algorithm for factor graphs (Kschischang et al, 2001), which owes its name to the fact that it consists of a series of summations and products. We adopt the latter view of operating on factor graphs for our discussion of BP, and refer to Kschischang et al (2001) for full details. The key idea behind BP is that each node in the factor graph sends "messages" to all of its neighbors, based on the ones received from the other neighbors. As the messages ultimately serve to compute marginals at the variable nodes, each message is a function of the respective variable node X involved in the exchange. Given the bipartite nature of the factor graph, there are two types of messages. The first type is sent from a variable node X to a neighboring factor node ϕ. Such a message provides a multiplicative summary of the messages from all other factors the variable participates in:

$$\mu_{X\to\phi}(X) = \prod_{\phi'\in n(X)\setminus\{\phi\}} \mu_{\phi'\to X}(X). \tag{5}$$

Here, $n(X)$ is the set of neighboring nodes of X in the factor graph. These messages are initially set to 1. The second type of message is sent from a factor node ϕ to a neighboring variable node X:

$$\mu_{\phi\to X}(X) = \sum_{\boldsymbol{X}\setminus\{X\}} \left(\phi(\boldsymbol{X}_\phi) \prod_{Y\in\boldsymbol{X}_\phi\setminus\{X\}} \mu_{Y\to\phi}(Y) \right) \tag{6}$$

Here, $\boldsymbol{X}$ denotes all random variables in the factor graph, $\boldsymbol{X}_\phi$ those that are arguments of factor ϕ (and thus ϕ's neighbors in the factor graph). The message thus (a) multiplies for each variable assignment the value of the factor ϕ and the corresponding messages received from all its participating variables except X, and (b) sums these products for all assignments to all RVs except X. These messages are initially set to $\sum_{\boldsymbol{X}\setminus\{X\}} \phi(\boldsymbol{X}_\phi)$.

During BP, a node sends a message to a specific neighbor once it has gotten messages from all its other neighbors. If the factor graph is a tree, this process terminates once a message has been sent for both directions of each edge, at which point the marginal of variable X is exactly the product of all messages $\mu_{\phi\to X}(X)$ directed towards it. If the factor graph contains cycles, marginals can be approximated by running BP for a sequence of iterations or

with damped updates, which is known as loopy BP. Although loopy BP is not guaranteed to output correct results, in practice it frequently converges and, when this happens, the values obtained are typically correct (Murphy et al, 1999; Yedidia et al, 2001).

As variable elimination, BP can be easily adapted to compute the MAP state by replacing the summation operator in Equation (6) with a maximization operator. This is called the max-product algorithm, or, if the underlying graph is a chain, the Viterbi algorithm.

2.2.5 Terminology of Relational Languages

We now briefly introduce the relational languages most popular to define lifted graphical models. We focus on the key language elements required in this context, which are (1) how to define random variables, that is, language elements whose values are governed by random processes; (2) how to define parameterized random variables (par-RVs) (Poole, 2003), whose instances are RVs; and (3) how to define arguments of potential functions, that is, vectors of par-RVs whose instances share the same factor structure and potential function. We refer to Section 3.1 for a formal treatment of par-RVs and par-factors, and to Section 3 for a detailed discussion of lifted graphical model formalisms including additional examples.

Structured Query Language (SQL). When using SQL, one of the most popular query languages for relational databases, to define graphical models, random variables typically correspond to attributes of tuples in the database that take values from the corresponding range of values. Defining vectors of random variables can therefore be done by means of `select` statements of the following type, which return a set of tuples that all have the same attribute structure and will share the same potential function:

```
SELECT <column names>
FROM <table names>
WHERE <selection constraints>
```

For instance, in the example of Figure 1, we can obtain the affiliations of all pairs of co-authors via

```
SELECT r1.affiliation, r2.affiliation
FROM Researcher r1, Researcher r2, Author a1, Author a2
WHERE r1.person = a1.person and a1.paper = a2.paper
      and a2.person = r2.person
```

which could be used for a potential function expressing that co-authors are more likely to have the same affiliation. SQL is used for instance in relational Markov networks, cf. Section 3.2.1.

First-order Logic. Another flexible and expressive representation of relational data frequently used in SRL is first-order logic (FOL). FOL distinguishes among four kinds of symbols: constants, variables, predicates, and functions. *Constants*, which we denote by typewriter lower-case letters such as `x` and `y` in abstract discussions, and by typewriter strings starting with lower-case letters in examples, e.g., `p1` and `r3` in the example of Figure 1, describe the objects in the domain of interest, which we will alternatively call *entities.* In this survey, we assume that entities are typed. *Logical variables*, denoted by typewriter upper-case letters such as `X` and `Y`, refer to arbitrary rather than concrete entities in the domain. *Predicates*, denoted by strings starting with an upper-case letter such as `Publication` and `Author`, represent attributes or relationships between entities. We assume predicates to be typed, e.g., the predicate `Author` only applies to pairs of entities of type paper and person, respectively. *Functions*, denoted by strings starting with an upper-case letter such as `AuthorOf`, evaluate to an entity in the domain when applied to one or more entities, e.g., $\texttt{AuthorOf}(\texttt{x}) = \texttt{y}$. The number of arguments of a predicate or a function is called its *arity.* A *term* is a constant, a variable, or a function on terms. A predicate applied to terms is called an *atom,* e.g., $\texttt{Author}(\texttt{X}, \texttt{Y})$. Terms and atoms are *ground* if they do not contain variables, e.g., $\texttt{Author}(\texttt{p1}, \texttt{r1})$. Ground atoms evaluate to `true` or `false`. Atoms are also called *positive literals*, and atoms preceded by the negation operator $\neg$ are called *negative literals.* A *formula* consists of a set of (positive or negative) literals connected by conjunction ($\wedge$) or disjunction ($\vee$) operators, e.g., $\neg\texttt{Publication}(\texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}) \vee \texttt{Author}(\texttt{W}, \texttt{AuthorOf}(\texttt{W}))$. The variables in formulas are quantified, either by an existential quantifier ($\exists$) or by a universal quantifier ($\forall$). Here we follow the common assumption that when no quantifier is specified for a variable, $\forall$ is understood by default. A formula expressed as a disjunction with at most one positive literal is called a *Horn clause*; if a Horn clause contains exactly one positive literal, then it is a *definite clause.* Using the laws of first-order logic, a definite clause $\neg b_1 \vee \ldots \vee \neg b_n \vee h$ can also be written as an implication $b_1 \wedge \ldots \wedge b_n \Rightarrow h$, e.g., $\texttt{Publication}(\texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}) \Rightarrow \texttt{Author}(\texttt{W}, \texttt{AuthorOf}(\texttt{W}))$ for the formula above. The conjunction $b_1 \wedge \ldots \wedge b_n$ is called the *body*, the single atom h the *head* of the clause. *Grounding* or *instantiating* a formula is done by replacing all variables with ground terms. Formulas containing functions of arity at least one have infinitely many groundings, which is often undesirable when using FOL to specify SRL models. One way to avoid this is to only consider groundings where variables are replaced with constants in all possible type-consistent ways, e.g., $\texttt{Author}(\texttt{p1}, \texttt{X})$ can be grounded to $\texttt{Author}(\texttt{p1}, \texttt{r1})$, $\texttt{Author}(\texttt{p1}, \texttt{r2})$, $\texttt{Author}(\texttt{p1}, \texttt{r3})$ and $\texttt{Author}(\texttt{p1}, \texttt{r4})$ in our example.

When using FOL to define graphical models, random variables typically correspond to ground atoms with values in the set $\{\texttt{true}, \texttt{false}\}$. For example, for publication `p3` and person `r1`, the ground atom $\texttt{Author}(\texttt{p3}, \texttt{r1})$ represents the assertion that `r1` is an author of `p3`. Non-ground atoms correspond to par-RVs, which become instantiated to RVs by grounding. For example, if `X` and `Y` are logical variables, $\texttt{Author}(\texttt{X}, \texttt{Y})$ is a par-RV because once we ground

it by replacing the parameters `X` and `Y` with actual entities, we obtain RVs. We note that the use of FOL in this context does not necessarily imply a FOL characterization of graphical models, and also that such models often do not employ the full expressive power of FOL.

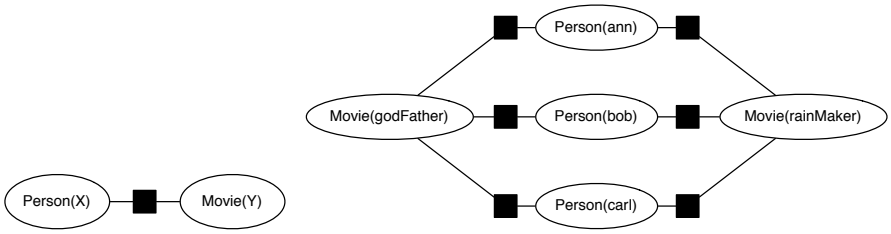
Lifted graphical models using elements of FOL include Markov logic networks (cf. Section 3.2.2), probabilistic soft logic (Section 3.2.3), Bayesian logic programs (Section 3.3.1), relational Bayesian networks (Section 3.3.2) and BLOG (Section 3.3.4).

Object-Oriented Representations. As an alternative to FOL, the attributes and relations of entities can be described using an object-oriented representation. Here again, `x` and `y` represent specific entities in the domain, whereas `X` and `Y` are variables, or entity placeholders. We again assume that entities are typed, which allows us to use chains of attributes and relations, expressed in a notation analogous to that commonly used in object-oriented languages, to identify sets of entities of a certain type starting from a given entity. For example, using the notation of Getoor et al (2007), $\texttt{x.Venue}$ refers to the (typically singleton) set of venues of paper `x`, whereas $\texttt{x.Author}$ refers to the set of its authors. Inverse relations are also allowed, e.g., $\texttt{y.Author}^{-1}$ refers to the set of papers of which `y` is an author. Longer chains are followed for all elements of intermediate sets, e.g., $\texttt{x.Author.Author}^{-1}\texttt{.Venue}$ gives the set of venues of all papers written by any author of `x`. Such chains can be used to specify par-RVs, which are instantiated by replacing variables with entities from the domain. Random variables thus correspond to attributes of relations, and aggregation functions, such as mean, mode, max, or sum, are used to deal with sets of such variables. For example, we can write $\texttt{mode}(\texttt{x.Author.Author}^{-1}\texttt{.Venue})$.

Lifted graphical models using object-oriented aspects include relational Markov networks (Section 3.2.1), probabilistic soft logic (Section 3.2.3), FACTORIE (Section 3.2.4), and probabilistic relational models (Section 3.3.3).

3 Overview of SRL models

Existing SRL representations can be split into two major groups. The first group consists of lifted graphical models, that is, representations that use a structured language to define a probabilistic graphical model. Representations in the second group impose a probabilistic interpretation on logical inference. As discussed in the introduction, to allow for greater depth, here we limit ourselves to the first group of languages. To provide a convenient representation that describes the common core of lifted graphical models, we start with par-factor graphs, short for parameterized factor graphs, defining them in the terminology of Poole (2003). A par-factor graph is analogous to a factor graph (Kschischang et al, 2001), cf. Section 2.2.1, in that it is a general representation for a large class of lifted graphical models, including both directed and undirected representations. Using the language of par-factor graphs, we can



(a) (b)

Fig. 4: Example of a par-factor graph and the corresponding factor graph obtained by instantiating par-RVs $\mathsf{Person}(\mathsf{X})$ and $\mathsf{Movie}(\mathsf{Y})$ for $\mathtt{X} \in \{\mathtt{ann}, \mathtt{bob}, \mathtt{carl}\}$ and $\mathtt{Y} \in \{\mathtt{godFather}, \mathtt{rainMaker}\}$.

discuss how different models specialize them, while keeping the discussion of inference and learning techniques on the general level as much as possible.

3.1 Par-Factor Graphs

Parametrized factors, or par-factors for short, provide a relational language to compactly specify sets of factors in a graphical model that only differ in their vector of RVs, but share the same structure and potential function. Such a set of par-factors defines a family of probability distributions based on a set of relations, which can be combined with different instances of the corresponding database to obtain specific distributions from that family. To emphasize the connection to factor graphs, we refer to a set of par-factors $\boldsymbol{\mathcal{F}} = \{(\mathbf{A}_i, \phi_i, \boldsymbol{\mathcal{C}}_i)\}$ as a *par-factor graph*. Par-factor graphs lift factor graphs analogously to how first-order logic lifts propositional logic. For instance, Figure 4a shows a par-factor graph with a single par-factor over par-RVs $\mathsf{Person}(\mathsf{X})$ and $\mathsf{Movie}(\mathsf{Y})$, which could express a prior probability that any person likes any movie, and Figure 4b shows the corresponding factor graph instantiating the par-RVs for $\mathtt{X} \in \{\mathtt{ann}, \mathtt{bob}, \mathtt{carl}\}$ and $\mathtt{Y} \in \{\mathtt{godFather}, \mathtt{rainMaker}\}$. Formally, a par-factor is a triple $\Phi = (\mathbf{A}, \phi, \boldsymbol{\mathcal{C}})$, where $\mathbf{A}$ is a vector of parameterized random variables (par-RVs), ϕ is a function from the values of RVs instantiating these par-RVs to the non-negative real numbers, and $\boldsymbol{\mathcal{C}}$ is a set of constraints on how the par-RVs may be instantiated. For typed relational languages, type constraints are included in $\boldsymbol{\mathcal{C}}$ by default. Let $\boldsymbol{\mathcal{I}}(\Phi_i)$ denote the set of random variable vectors $\boldsymbol{A}$ that are instantiations of $\mathbf{A}_i$ under constraints $\boldsymbol{\mathcal{C}}_i$. For any $\boldsymbol{A} \in \boldsymbol{\mathcal{I}}(\Phi_i)$, we denote the value assignment $\boldsymbol{x}$ restricted to the RVs $\boldsymbol{A}$ by $\boldsymbol{x}_{\boldsymbol{A}}$. The par-factor graph $\boldsymbol{\mathcal{F}}$ defines a probability distribution as follows, where $\boldsymbol{X}$ is the vector

of all RVs that instantiate par-RVs in $\boldsymbol{\mathcal{F}}$:

$$P(\boldsymbol{X} = \boldsymbol{x}) = \boldsymbol{\mathcal{F}}(\boldsymbol{x}) = \frac{1}{Z} \prod_{\Phi_i \in \boldsymbol{\mathcal{F}}} \prod_{\boldsymbol{A} \in \boldsymbol{\mathcal{I}}(\Phi_i)} \phi_i(\boldsymbol{x}_{\boldsymbol{A}}) \tag{7}$$

That is, the probability distribution is the normalized product of the factors corresponding to all instances of par-factors in the par-factor graph, and as such directly corresponds to the one defined by the underlying factor graph, cf. Equation (1). However, here, all the factors that are instantiations of the same par-factor share common structure and parameters. Especially in the context of parameter learning (cf. Section 5.1), those shared parameters are also called *tied* parameters. Parameter tying allows for better generalization, as it combines a flexible number of random variables with a fixed number of parameters. Par-factor graphs thus exploit both probabilistic and relational structure to compactly represent probability distributions.

As in the propositional case, cf. Section 2.2.1, even though par-factor graphs provide a very general language to specify probabilistic models, it is often useful to restrict the discussion to a specific subclass of such models, and indeed, most research has focused on either directed or undirected models. In the remainder of this section, we discuss how several popular SRL representations can be viewed as special cases of par-factor graphs, that is, how they express the specific $\mathbf{A}_i$-s, the ϕ_i-s, and the $\boldsymbol{\mathcal{C}}_i$-s they consider. This is not meant to be an exhaustive list; rather, our goal is to highlight some of the different flavors of representations.

3.2 Undirected SRL Representations

This subsection discusses undirected lifted graphical models, which all define Markov networks when instantiated. The key differences of these representations lie in the way par-factors are specified, namely using SQL (relational Markov networks), different subsets of first-order logic (Markov logic networks and probabilistic soft logic), or imperative programming (FACTORIE).

3.2.1 Relational Markov Networks

As their name suggests, relational Markov networks (RMNs) (Taskar et al, 2002) define Markov networks through a relational representation, more specifically, an object-oriented language and SQL. We illustrate the key principles using an example from collective classification of hyperlinked documents, as presented by Taskar et al (2002). In an RMN, each par-factor $\Phi = (\mathbf{A}, \phi, \boldsymbol{\mathcal{C}})$ is given by an SQL `select` statement defining $\mathbf{A}$ and $\boldsymbol{\mathcal{C}}$ and a potential function ϕ over instantiations of $\mathbf{A}$ in log-linear form. More specifically, the vector $\mathbf{A}$ of par-RVs corresponding to attributes is established by the `select...from` part, and the constraints $\boldsymbol{\mathcal{C}}$ over instantiations by the `where` part. Par-RVs are

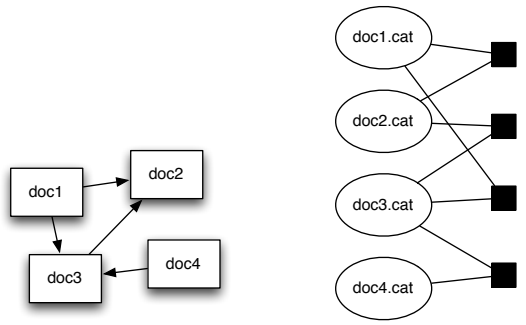


Fig. 5: RMN example: (a) hyperlink structure, (b) Markov network.

instantiated to multinomial RVs, that is, RVs corresponding to attributes of specific tuples that take one of multiple discrete values, and the Markov network contains a clique for each such RV vector. For instance, the following par-factor sets up a clique between the category assignments of any two hyperlinked documents in order to capture the intuition that documents on the web that link to one another typically have correlated categories:

```
SELECT D1.Category, D2.Category
FROM Document D1, Document D2, Link L
WHERE L.From = D1.Key and L.To = D2.Key
```

Figure 5a shows a small example network, Figure 5b the corresponding Markov network set up by the par-factor above. The log-linear potential function ϕ is defined separately via a parameter λ and a feature function f, that is, for any instantiation $\boldsymbol{A}$ of the par-RVs in $\mathbf{A}$ and specific values $\boldsymbol{a}$, we have $\phi(\boldsymbol{A} = \boldsymbol{a}) = \exp(\lambda \cdot f(\boldsymbol{a}))$. The definition of ϕ can be used to incorporate further domain knowledge. For example, if we know that most pages tend to link to pages of the same category, we can define $\phi(\texttt{D1.Category}, \texttt{D2.Category}) = \exp(\lambda \cdot \mathbf{1}[\texttt{D1.Category} = \texttt{D2.Category}])$, where the feature function takes the form of the indicator function $\mathbf{1}[x]$ that returns 1 if the proposition x is true and 0 otherwise. A positive λ encourages hyperlinked pages to be assigned the same category, while a negative λ discourages this.

3.2.2 Markov Logic Networks

Markov logic networks (MLNs) (Richardson and Domingos, 2006; Domingos and Lowd, 2009) also define a Markov network when instantiated. As an illustration, we present an example from (Richardson and Domingos, 2006),

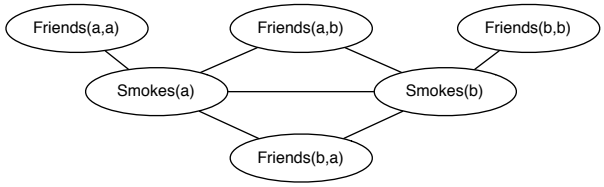


Fig. 6: Markov network of MLN example.

in which the patterns of human interactions and smoking habits are considered. Par-factors in MLNs are specified using first-order logic. Each par-factor $\Phi = (\mathbf{A}, \phi, \boldsymbol{\mathcal{C}})$ is represented by a first-order logic formula F_Φ with an attached weight w_Φ. Each atom in the formula specifies one of the par-RVs in $\mathbf{A}$. In the instantiated Markov network, each instantiation, or grounding, of F_Φ establishes a clique among Boolean-valued RVs corresponding to the ground atoms that appear in that instantiation. For instance, the following formula with weight w encodes that friends have similar smoking habits, i.e., that if two people are friends, then they tend to either both be smokers or both be non-smokers.

$$w : \texttt{Friends}(\texttt{X}, \texttt{Y}) \Rightarrow (\texttt{Smokes}(\texttt{X}) \Leftrightarrow \texttt{Smokes}(\texttt{Y}))$$

The par-RVs in the par-factor defined by this rule are

$$\mathbf{A} = \langle \texttt{Friends}(\texttt{X}, \texttt{Y}), \texttt{Smokes}(\texttt{X}), \texttt{Smokes}(\texttt{Y}) \rangle,$$

and every possible instantiation of these par-RVs establishes a clique in the instantiated Markov network, e.g., if there are only two entities, `a` and `b`, then the instantiated Markov network is the one shown in Figure 6.[1]

The potential function ϕ is implicit in the formula, as we describe next. Let $\boldsymbol{A}$ be the set of RVs in a particular instantiation or grounding f_Φ of the formula F_Φ, and $\boldsymbol{a}$ be a particular assignment of truth values to $\boldsymbol{A}$; then, $\phi(\boldsymbol{A} = \boldsymbol{a}) = \exp(\lambda_\Phi \cdot F_\Phi(\boldsymbol{a}))$, where $\lambda_\Phi = w_\Phi$, and $F_\Phi(\boldsymbol{a}) = 1$ if f_Φ is true for the given truth assignment $\boldsymbol{a}$ and $F_\Phi(\boldsymbol{a}) = 0$ otherwise. In other words, clique potentials in MLNs are represented using log-linear functions in which the first-order logic formula itself acts as a feature function, whereas the weight associated with the formula provides the parameter.

So far, we have not discussed how MLNs specify the constraints $\boldsymbol{\mathcal{C}}$ of a par-factor. MLNs do not have a special mechanism for describing constraints, but constraints can be implicit in the formula structure. Two ways of doing this are as follows. First, we can constrain groundings by providing constants as arguments of par-RVs. For example, writing $\texttt{Friends}(\texttt{a}, \texttt{Y}) \Rightarrow (\texttt{Smokes}(\texttt{a}) \Leftrightarrow \texttt{Smokes}(\texttt{Y}))$ results in the subset of groundings of the formula above where

[1] When grounding the par-RV vector results in a vector with repeated RVs, e.g., $\langle \texttt{Friends}(\texttt{a}, \texttt{a}), \texttt{Smokes}(\texttt{a}), \texttt{Smokes}(\texttt{a}) \rangle$ for $\texttt{X} = \texttt{Y} = \texttt{a}$, those repetitions are not depicted in the graphical representation, but the potential function is still evaluated on the full vector.

$\texttt{X} = \texttt{a}$. Second, when computing conditional probabilities, we can treat some predicates as background knowledge that is given at inference time rather than as definitions of random variables, similarly to the use of the `Link` relation in the RMN example above. For example, suppose we know that at inference time we will observe as evidence the truth values of all groundings of `Friends` atoms, and the goal will be to infer people's smoking habits. Then, the formula $\texttt{Friends}(\texttt{X},\texttt{Y}) \Rightarrow (\texttt{Smokes}(\texttt{X}) \Leftrightarrow \texttt{Smokes}(\texttt{Y}))$ can be seen as setting up a clique between the `Smokes` values only of entities that are friends. If $\texttt{Friends}(\texttt{x},\texttt{y})$ is false for a particular pair of entities `x` and `y`, then the corresponding instantiation of the formula is trivially satisfied, regardless of assignments to groundings of `Smokes`. Such an instantiation thus contributes the same constant factor $\exp(\lambda \cdot 1)$ to the probability of each truth value assignment consistent with the evidence, and can therefore be ignored when instantiating the MLN.

A variant of MLNs are Hybrid MLNs (Wang and Domingos, 2008), which extend MLNs to allow for real-valued random variables. In Hybrid MLNs, the same formula can contain both binary-valued and real-valued atoms. Such formulas are evaluated by interpreting conjunction as a multiplication of values. Another related formalism are the relational continuous models of Choi et al (2010), which allow for par-factors with continuous valued variables, but restricting ϕ to Gaussian potentials.

3.2.3 Probabilistic Soft Logic

Probabilistic soft logic (PSL) (Broecheler et al, 2010) is another lifted Markov network model. As in MLNs, par-RVs in PSL correspond to logical atoms and RVs to ground atoms. In contrast to MLNs, where RVs take Boolean variables, and to hybrid MLNs, where some RVs take Boolean values while others take real values, all RVs in PSL take *soft truth values* from the interval $[0, 1]$. This allows for easy integration of similarity functions. To define par-factors, PSL uses a mixture of first-order logic and object-oriented languages, where the latter provides convenient syntax for specifying sets. Each par-factor $\Phi = (\mathbf{A}, \phi, \boldsymbol{\mathcal{C}})$ over a set of atoms $\mathbf{A}$ is specified via a rule $R_\Phi = l_1 \wedge \ldots \wedge l_m \Rightarrow l_{m+1} \vee \ldots \vee l_n$ with weight w_Φ, where each l_i is either an atom in $\mathbf{A}$ or the negation of such an atom. The potential function ϕ is defined based on R_Φ as discussed below. Constraints in $\boldsymbol{\mathcal{C}}$ can be specified in a similar manner as in MLNs. To illustrate, consider an example by Broecheler et al (2010) in which the task is to infer document similarities in Wikipedia based on document attributes and user interactions with the document. One potentially useful rule states that two documents are similar if the sets of their editors are similar and their text is similar:

$$(\{\texttt{A.editor}\} \approx_{s_1} \{\texttt{B.editor}\}) \wedge (\texttt{A.text} \approx_{s_2} \texttt{B.text}) \Rightarrow (\texttt{A} \approx_{s_3} \texttt{B})$$

Above, $\approx_{s_i}$ represent similarity functions, and a term enclosed in curly braces, as in $\{\texttt{A.editor}\}$, refers to the set of all entities related to the variable through the relation. This rule uses the par-RVs $\mathbf{A} = \{(\{\texttt{A.editor}\} \approx_{s_1} \{\texttt{B.editor}\}), (\texttt{A.text} \approx_{s_2} \texttt{B.text}), (\texttt{A} \approx_{s_3} \texttt{B})\}$. Each grounding of such a rule introduces a

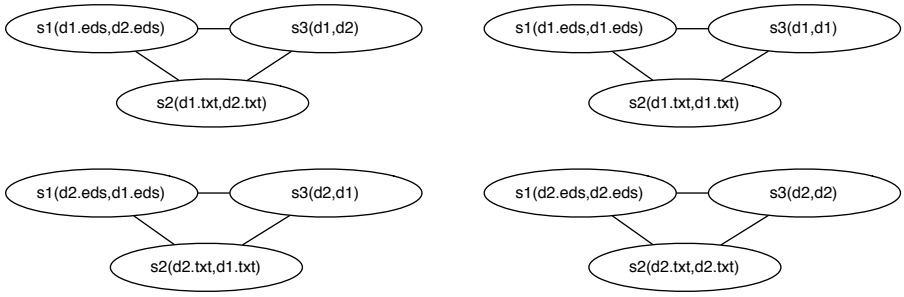


Fig. 7: Markov network of PSL example for two documents.

clique between its random variables in the Markov network, as illustrated in Figure 7. The clique potential ϕ is again implicitly given by the rule. The evaluation $R_{\Phi}(\boldsymbol{a})$ of a rule R_{Φ} on an assignment $\boldsymbol{a}$ to an instantiation $\boldsymbol{A}$ of the par-RVs is obtained by interpreting conjunction and disjunction using the Lukasiewicz t-(co)norms as follows:

$$\begin{aligned} x \wedge y &= \max\{0, x + y - 1\} \\ x \vee y &= \min\{x + y, 1\} \\ \neg x &= 1 - x \end{aligned}$$

For instance, when assigning 1.0 to RV ($\{\texttt{a.editor}\} \approx_{s_1} \{\texttt{b.editor}\}$), 0.9 to ($\texttt{a.text} \approx_{s_2} \texttt{b.text}$), and 0.3 to ($\texttt{a} \approx_{s_3} \texttt{b}$), the value of the above rule is $\min\{\min\{(1-1.0)+(1-0.9), 1\}+0.3, 1\} = 0.4$. The distance to satisfaction of a rule instantiation is then defined as $d(R_{\Phi}(\boldsymbol{a})) = 1 - R_{\Phi}(\boldsymbol{a})$, and the potential of the corresponding clique as $\phi(\boldsymbol{A} = \boldsymbol{a}) = \exp(-w_{\Phi} \cdot (d(R_{\Phi}(\boldsymbol{a})))^p)$, where $p \in \{1, 2\}$ provides a choice of the type of penalty imposed on violated rules.

3.2.4 Imperatively Defined Factor Graphs

Par-factor graphs can also be specified using programming languages, as illustrated by FACTORIE, an implementation of imperatively defined factor graphs (McCallum et al, 2009). FACTORIE uses *Scala* (Odersky et al, 2004), a strongly-typed programming language that combines object-oriented and functional elements. Both par-RVs and par-factors correspond to classes programmed by the user, and RVs to objects instantiating their par-RV's class. Each par-factor $\Phi = (\mathbf{A}, \phi, \boldsymbol{\mathcal{C}})$ is defined as a factor template class that takes the par-RVs $\mathbf{A}$ as arguments. The instantiation constraints $\boldsymbol{\mathcal{C}}$ are provided as a set of `unroll` methods in the class, one for each par-RV, which construct the RVs instantiating the par-RV vector and their connections, and thus build (or "unroll") the instantiated factor graph. Random variables can have arbitrary domains. The potential function ϕ is defined as $\phi(\boldsymbol{A} = \boldsymbol{a}) = \exp(\sum_{i \in I} \lambda_i f_i(\boldsymbol{a}))$, where λ_i are parameters and f_i are sufficient statistics which are implemented

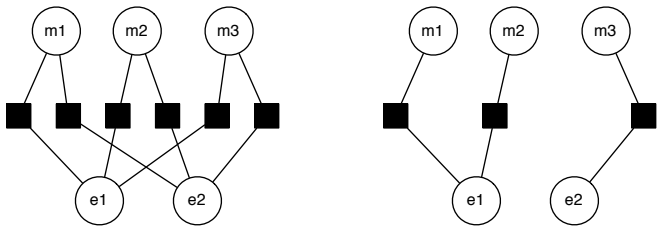


Fig. 8: FACTORIE example with three mentions and two entities: (a) full factor graph, and (b) factor graph as constructed by the `unroll` methods for assignment $\mathtt{m}_1\mathtt{.entity} = \mathtt{e}_1$, $\mathtt{m}_2\mathtt{.entity} = \mathtt{e}_1$, $\mathtt{m}_3\mathtt{.entity} = \mathtt{e}_2$, $\mathtt{e}_1\mathtt{.mentions} = \{\mathtt{m}_1, \mathtt{m}_2\}$, $\mathtt{e}_2\mathtt{.mentions} = \{\mathtt{m}_3\}$.

via a `statistics` method in the factor template class and thus can have arbitrary form. Consider a simplified version of an example from (McCallum et al, 2009). We are given a set of *mentions* of objects in the form of strings, and a set of *entities*, i.e., actual objects. The task is to determine for each mention which entity it refers to (and conversely, for each entity, the set of its mentions). The idea is to set up a factor graph with one factor for each pair of a mention m and an entity e (as shown in Figure 8a), where the statistics $f_i(e, m)$ are 1 if m is assigned to e and m and e are similar (see below), and 0 otherwise. The definition of this factor graph uses two par-RV classes `Entity` and `Mention`. An instance `m` of a `Mention` par-RV is identified by a string `m.string`, e.g., "Elizabeth Smith" or "Liz S.", and takes an `Entity` as its value `m.entity`. An instance `e` of an `Entity` par-RV corresponds to an actual entity in the domain of interest, e.g., a person. Its value `e.mentions` is a set of `Mention`s, and it additionally contains a canonical representation `e.canonical`, which is a unique string computed from the set `e.mentions`. Given an assignment to all instances of these par-RVs, the following factor template generates the factor graph.

```
val corefTemplate = new Template[Mention, Entity]{
   def unroll1(m:Mention) = Factor(m, m.entity)
   def unroll2(e:Entity) = for (mention <- e.mentions)
                                    yield Factor(mention, e)
   def statistics(m:Mention,e:Entity) =
                   Bool(distance(m.string,e.canonical)<0.5)
}
```

More specifically, it sets up a pairwise factor for each mention and its assigned entity (via the `unroll1` method) as well as for each entity and every mention in its set (via the `unroll2` method), as in the example in Figure 8b. This template is programmed to take advantage of the fact that the factor graph is always evaluated for a given assignment during inference, and that all factors

not related to this assignment evaluate to one and thus can be omitted. For each factor in the unrolled graph, the `statistics` method produces sufficient statistics by comparing the distance between the mention and the canonical representation of the entity to a threshold.

3.2.5 Discussion

The SRL representations discussed so far all define undirected graphical models with log-linear potential functions, but differ in the type of modeling freedom they provide. In MLNs and PSL, the feature function takes a fixed form based on the logical structure of a par-factor, whereas RMNs and FACTORIE let the user define the feature function. The probabilistic interpretation is centered on different aspects of the domain in different models, with RMNs focusing on values of attributes, MLNs and PSL on relations between objects, and FACTORIE on objects themselves. MLNs and PSL further differ in the logical structure of par-factors they allow, with PSL's more restricted language allowing for more efficient inference, as we will discuss in Section 4.

3.3 Directed SRL Representations

This subsection describes directed lifted graphical models, which all define Bayesian networks when instantiated. We again cover representations using different relational languages: definite clauses and logic programming (Bayesian logic programs), formulas expressing functions (relational Bayesian networks), a relational database representation with object-oriented elements (PRMs), and a generative model in a language close to first-order logic (BLOG).

As all these models define Bayesian networks, a par-factor $\Phi = (\mathbf{A}, \phi, \boldsymbol{\mathcal{C}})$ always has the following form. The par-RVs $\mathbf{A}$ can be split into a child par-RV C and a vector of parent par-RVs $\mathbf{Pa}(\mathsf{C})$, and the potential function ϕ represents a conditional probability distribution (CPD) for any instance C of C given instances $\boldsymbol{Pa}(C)$ of $\mathbf{Pa}(\mathsf{C})$, that is,

$$\phi(C = c, \boldsymbol{Pa}(C) = \boldsymbol{pa}) = P(C = c \mid \boldsymbol{Pa}(C) = \boldsymbol{pa}).$$

As a consequence, the expression in Equation (7) is automatically normalized, i.e., $Z = 1$. When specifying directed SRL models, care must be taken to ensure that their instantiations result in cycle-free directed graphs. However, as discussed by Jaeger (2002, Section 3.2.1), this problem is undecidable in general, and guarantees exist only for restricted cases.

Furthermore, when specifying directed models at the par-factor level, the number of parents of a node in an instantiated factor graph might depend on the particular grounding. Consider for instance a conditional probability $P(X|Y_1, Z_1, \ldots, Y_n, Z_n)$, where X depends on all instances of Y and Z related to X in a specific way, and n will thus depend on the grounding. We therefore need a way to specify this distribution for arbitrary n. Two common ways to do this are *aggregates* (Perlich and Provost, 2003) and *combining*

rules (e.g., Jaeger, 1997). Aggregates first aggregate the values of all parent variables of the same type into a single value, and provide the conditional probability of the child variable given these aggregated values. That is, in the example, one would use two aggregate functions agg_Y and agg_Z, together with a conditional probability distribution P', to define $P(X|Y_1, Z_1, \ldots, Y_n, Z_n) = P'(X|\text{agg}_Y(Y_1, \ldots, Y_n), \text{agg}_Z(Z_1, \ldots, Z_n))$. An approach based on combining rules, on the other hand, would specify the conditional probability distribution for the child variable for $n = 1$ as well as a function that computes a single conditional distribution from n conditional distributions. In the example, one would thus use a distribution $P''(X|Y, Z)$ and a combining function f, and define $P(X|Y_1, Z_1, \ldots, Y_n, Z_n) = f(P''(X|Y_1, Z_1), \ldots, P''(X|Y_n, Z_n))$. For example, one commonly used combining function is the noisy-or:

$$P(X = x|Y_1 = y_1, \ldots, Y_n = y_n) = 1 - \prod_{1 \leq i \leq n} (1 - P(X = x|Y_i = y_i))$$

The idea behind noisy-or is that each of the $Y_i = y_i$ can independently cause $X = x$ with a certain probability, and X thus takes value x if at least one such causation happens.

3.3.1 Bayesian Logic Programs

In Bayesian logic programs (BLPs) (Kersting and De Raedt, 2001), par-RVs are expressed as logical atoms. The dependency structure of a par-RV C on its parents $\mathbf{Pa}(\mathsf{C})$ is represented as a definite clause, called a Bayesian clause, in which the head consists of C, the body consists of the conjunction of the atoms in $\mathbf{Pa}(\mathsf{C})$, and the implication is replaced by a | to indicate probabilistic dependency. Kersting and De Raedt (2001) give an example from genetics (originally by Friedman et al (1999a)), in which the blood type `bt(X)` of person `X` depends on inheritance of a single gene, one copy of which, `mc(X)` is inherited from `X`'s mother, while the other copy `pc(X)` is inherited from her father. In BLPs, this dependency is expressed as

$$\texttt{bt(X)}|\texttt{mc(X)}, \texttt{pc(X)}$$

Random variables correspond to ground atoms and are not restricted to evaluating to just `true` or `false`, but can have arbitrary finite domains. In the example, the RVs obtained by grounding `mc(X)` and `pc(X)` can take on values from $\{a, b, 0\}$, whereas those for `bt(X)` can take on values from $\{a, b, ab, 0\}$. Par-factors are formed by coupling a Bayesian clause with a potential function ϕ in the form of a CPD over values for an instance of C given values for instances of $\mathbf{Pa}(\mathsf{C})$, e.g., as a conditional probability table. The constraints $\boldsymbol{\mathcal{C}}$ on instantiations can be modelled via *logical predicates*. For instance,

$$\texttt{mc(X)}|\texttt{mother(Y,X)}, \texttt{mc(Y)}, \texttt{pc(Y)}$$

models the dependency of the gene inherited from the mother on the mother's own genes, where `mother` is a logical predicate. When instantiating this par-factor, only groundings for which `mother(Y,X)` holds are considered. In BLPs,

the full power of the logic programming language Prolog can be used to define logical predicates.

Using BLPs, we next give an example of the use of combining rules. Following the example from (Kersting and De Raedt, 2001), suppose that in the genetics domain we have the following two rules:

$$\texttt{bt(X)}|\texttt{mc(X)}$$
$$\texttt{bt(X)}|\texttt{pc(X)}$$

Each of these rules comes with a CPD, the first one giving $P(\texttt{bt(X)}|\texttt{mc(X)})$, and the second one $P(\texttt{bt(X)}|\texttt{pc(X)})$. However, what we need is a single CPD for predicting $\texttt{bt(X)}$ given both of these quantities. Using noisy-or as the combining rule, we get $P(\texttt{bt(X)}|\texttt{mc(X)},\texttt{pc(X)}) = 1 - (1 - P(\texttt{bt(X)}|\texttt{mc(X)})) \cdot (1 - P(\texttt{bt(X)}|\texttt{pc(X)}))$.

3.3.2 Relational Bayesian Networks

Relational Bayesian networks (RBNs) (Jaeger, 2002) also represent par-RVs as logical atoms, whose groundings take values from finite domains. In the most basic form, an RBN contains one par-factor Φ_R for each predicate R in its vocabulary, where the child par-RV C is an atom of R with variables as arguments. Recursive dependencies between par-RVs with the same predicate are possible if acyclicity of the resulting Bayesian network is ensured. The potential ϕ_R is represented as a probability formula in a syntax that bears a close correspondence to first-order logic and is evaluated as a function of the values of the instances of $\mathbf{Pa}(\mathsf{C})$. The par-RVs $\mathbf{A}$ are implicitly given through these probability formulas, which are recursively defined to consist of (i) constants in $[0, 1]$, which in the extreme cases of 1 and 0 correspond to `true` and `false` respectively; (ii) indicator functions, which take tuples of logical variables as arguments and correspond to relational atoms; (iii) convex combinations of formulas, which correspond to Boolean operations on formulas; and, finally, (iv) combination functions, such as mean, that combine the values of several formulas.

To illustrate, consider a slight adaptation of an example by Jaeger (2002), where the task is, given the pedigree of an individual `x`, to reason about the values of two relations, $\texttt{FA(x)}$ and $\texttt{MA(x)}$, which indicate whether `x` has inherited a dominant allele A from its father and mother respectively. The probability formula for $\texttt{FA(X)}$ may be:

$$\phi_{\texttt{FA}}(\texttt{X}) = \phi_{\texttt{knownFather}}(\texttt{X}) \cdot \phi_{\texttt{A-from-father}}(\texttt{X}) + (1 - \phi_{\texttt{knownFather}}(\texttt{X})) \cdot \theta$$

Here, $\phi_{\texttt{knownFather}}(\texttt{X})$ evaluates to 1 if the father of `X` is included in the pedigree and to 0 otherwise; $\phi_{\texttt{A-from-father}}(\texttt{X})$ is defined as the mean over the `FA` and `MA` values of `X`'s father: $\phi_{\texttt{A-from-father}}(\texttt{X}) = \text{mean}\{\texttt{FA(Y)}, \texttt{MA(Y)}|\texttt{father(Y, X)}\}$; and θ is a learnable parameter that can take values in the range $[0, 1]$. Sub-formulas in the form of indicator functions can be used to specify the instantiation constraints $\boldsymbol{\mathcal{C}}$, as is the case with the $\phi_{\texttt{knownFather}}(\texttt{X})$ sub-formula above, and

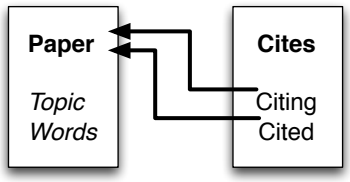


Fig. 9: PRM example

through selection formulas in combination functions, as $\texttt{father}(\texttt{Y},\texttt{X})$ for mean above.

3.3.3 Probabilistic Relational Models

Probabilistic relational models (PRMs) (Koller and Pfeffer, 1998; Getoor et al, 2007) take a relational database perspective and use an object-oriented language, akin to that described in Section 2.2.5, to specify the schema of a relational domain. Both entities and relations are represented as classes, each of which comes with a set of descriptive attributes and a set of reference slots through which classes refer to one another. Internally, each reference slot is defined by an arbitrary SQL query. Using an example by Getoor et al (2007), consider a document citation domain that consists of two classes (illustrated in Figure 9), the `Paper` class with attributes `Paper.Topic` and `Paper.Words`, and the `Cites` class, which establishes a citation relation between two papers via reference slots `Cites.Cited` and `Cites.Citing`. In the most basic form of PRMs, the values of reference slots are assumed given, and the par-RVs correspond to descriptive attributes of objects, either of the object itself, or of objects related to it through chains of reference slots. Constraints $\boldsymbol{\mathcal{C}}$ on par-RV instantiations can be expressed with the SQL queries defining the reference slots. By starting from specific objects, par-RVs are grounded to RVs that take values from the finite domain of the corresponding attribute. Each par-factor is defined by specifying the par-RVs corresponding to the child node C and the parent nodes $\mathbf{Pa}(\mathsf{C})$ respectively, and providing a conditional probability distribution for C given $\boldsymbol{Pa}(C)$. For example, to express that a paper `P`'s topic probabilistically depends on the topics of the papers `P` cites as well as those that cite `P`, one could construct a par-factor where $\mathsf{C} = \texttt{P.Topic}$, and $\mathbf{Pa}(\mathsf{C}) = \{\texttt{P.Citing}^{-1}\texttt{.Cited.Topic}, \texttt{P.Cited}^{-1}\texttt{.Citing.Topic}\}$. Thus, in Figure 9, the first parent par-RV starts from the paper `P`, first follows all `Citing` arrows backwards to find all instances of `Cites` where `P` is the citing paper, and then for all those follows the `Cited` arrow forwards to find all cited papers, whose `Topic` attributes provide the RVs instantiating the par-RV. Clearly, the number of these instantiations can vary across different papers. Like many other directed SRL models, PRMs use aggregation functions to combine the values of such sets of RVs into a single value.

While we have focused here on uncertainty over the attributes of relations, more general forms of PRMs that allow for uncertainty over the values of reference slots have been considered as well, focusing on two situations: when the number of links is known, but not the specific objects that are linked (reference uncertainty), as well as when neither the number of links nor the linked objects are known (existence uncertainty). In the above example, this makes it possible to express uncertainty over the values of reference slots, e.g., the paper appearing as `Cites.Citing` in a given instance of `Cites`, or over the existence of entire tuples of the `Cites` relation. We refer to Getoor et al (2007) for the technical details on these extensions.

3.3.4 BLOG

BLOG, short for Bayesian LOGic, is a typed relational language for specifying generative models (Milch et al, 2005). Par-RVs in BLOG are represented as first-order logic atoms, RVs as ground atoms. Par-factors are given by *dependency statements* of the form

$$\mathsf{C} \text{ if } \boldsymbol{\mathcal{C}} \text{ then} \sim \phi(\mathbf{Pa}(\mathsf{C})),$$

where the if $\boldsymbol{\mathcal{C}}$ then part can be omitted if there are no constraints and ϕ is implemented as a Java class. Such a statement expresses that the value of an instantiation of the child par-RV C, respecting $\boldsymbol{\mathcal{C}}$, is drawn from the probability distribution ϕ given the values of the corresponding instances of the parent par-RVs $\mathbf{Pa}(\mathsf{C})$. For example, Milch et al (2005) model the task of entity resolution in BLOG. They view the set of citations of a given paper as being drawn uniformly at random from the set of known publications. This is captured by the following BLOG statement:

$$\texttt{PubCited}(\texttt{C}) \sim \text{Uniform}(\{\texttt{Publication P}\}).$$

Similarly, the citation string is viewed as being generated by a string corruption model `CitDistrib` as a function of the authors and title of the paper being cited:

$$\texttt{CitString}(\texttt{C}) \sim \text{CitDistrib}(\texttt{TitleString}(\texttt{C}), \texttt{AuthorString}(\texttt{C})).$$

A unique characteristic of BLOG is that it does not assume that the set of entities in a domain is known in advance and instead allows reasoning over variable numbers of entities. This functionality is supported by allowing *number* statements, in which the number of entities of a given type is drawn from a given distribution. For example, in the entity resolution task, the number of researchers #`Researcher` is not known in advance and is instead drawn from a user-defined prior distribution:

$$\#\texttt{Researcher} \sim \text{NumResearchersPrior}().$$

3.3.5 Discussion

The directed SRL representations discussed here all define par-factor graphs whose potential functions correspond to conditional distributions of one child par-RV given a set of parent par-RVs. PRMs take an object-oriented view, where par-RVs correspond to attributes of objects. BLPs, RBNs and BLOG all use logical atoms as par-RVs, but differ in how they specify connections between these: logic programming (BLPs), probability formulas with a syntax close to first-order logic (RBNs), or a relational language for generative models that allows uncertainty over the number of objects (BLOG). To deal with flexible numbers of parents, BLPs and RBNs use combining functions, PRMs use aggregates, and BLOG allows arbitrary code to define the CPD.

3.4 Directed versus Undirected Models

The SRL representations discussed so far define either a directed or an undirected graphical model when instantiated. These representations have relative advantages and disadvantages, analogous to those of directed and undirected graphical models, cf. (Koller and Friedman, 2009). In terms of representation, directed models are appropriate when one needs to express a causal dependence or a generative process as in BLOG. On the other hand, undirected models are better suited to domains containing cyclic dependencies on the ground level, such as a person's smoking habits depending on the smoking habits of his or her friends (and vice versa). In undirected models, the par-factors shared by a single par-RV can naturally be combined by simply multiplying them, though this might not be the best combining rule for the problem at hand, and can actually make the distribution dependent on the domain size (Jain, 2011). Directed models, on the other hand, rely on separately defined combining functions, such as noisy-or, or aggregation functions, such as count, mode, max, and average. The use of combining functions in directed models allows for multiple independent causes of a given par-RV to be learned separately and then combined at prediction time (Heckerman and Breese, 1994), whereas this kind of causal independence cannot be exploited in undirected models. Finally, because factors in directed models represent conditional probability distributions, they are automatically normalized, which simplifies inference. In contrast, in undirected models one needs to find efficient ways of computing, or estimating, the normalization constant Z (in Equation (7)). We will discuss issues pertaining to learning directed and undirected SRL models from data in Section 5.

Hybrid SRL representations combine the positive aspects of directed and undirected models. One such model is relational dependency networks (RDNs) (Neville and Jensen, 2007), which can be viewed as a lifted dependency network model. Dependency networks (Heckerman et al, 2000) are similar to Bayesian networks in that, for each variable X, they contain a factor ϕ_X that represents the conditional probability distribution of X given its parents, or immediate

neighbors, $\boldsymbol{Pa}(X)$. Unlike Bayesian networks, however, dependency networks can contain cycles and do not necessarily represent a coherent joint probability distribution. As in Markov networks, the set of parents $\boldsymbol{Pa}(X)$ of a variable X render it independent of all other variables in the network. Marginals are recovered via sampling, e.g., Gibbs sampling (see Section 4). RDNs lift dependency networks to relational domains. Par-factors in RDNs are similar to those in PRMs and are represented as conditional probability distributions over values for a child par-RV C and the set of its parents $\mathbf{Pa}(\mathsf{C})$. Analogous to dependency networks, however, cycles are allowed and thus, as in dependency networks, RDNs do not always represent a consistent joint probability distribution.

There has also been an effort to unify directed and undirected models by providing an algorithm that converts a given directed model to an equivalent MLN (Natarajan et al, 2010). In this way, one can model multiple causes of the same variable independently while taking advantage of the variety of inference algorithms that have been implemented for MLNs. Bridging directed and undirected models is important also as a step towards representations that combine both directed and undirected sub-components.

4 Inference

As in graphical models, the two key inference tasks in lifted graphical models are marginal inference, cf. Equation (3), and MPE inference, cf. Equation (4). The former computes the probability of an assignment to a subset of the random variables, marginalizing out the remaining ones, and thus summarizes all states corresponding to that assignment; the latter finds the most likely joint assignment to a set of unknowns, given a set of observations, and thus focuses on a single strong explanation for the observations (the MAP state).

We first describe *lifted inference*, that is, inference approaches that operate on the first-order level (Section 4.1), followed by a brief overview of techniques that *ground* the model and perform propositional inference (Section 4.2). While most techniques use one specific probabilistic language, often with Boolean random variables, the par-factor view taken here suggests that much of the existing work could be generalized and applied in other settings as well. In Section 4.3, we conclude with pointers to a variety of recent approaches in the field.

4.1 Lifted Inference

Lifted graphical models compactly specify graphical models by grouping factors with identical structure and parameters into par-factors. Grounding such models to perform inference on a propositional model (cf. Section 4.2) therefore results in potentially large amounts of repeated computations. *Lifted inference* avoids this undesired blow-up by taking advantage of these groups during inference. The literature often distinguishes between *top-down* approaches, which

start from a lifted model and avoid grounding par-factors as much as possible, and *bottom-up* approaches, which start from a propositional model and detect repeated structure. The earliest lifted techniques are based on recognizing identical structure that requires the same computations, and performing the computation only the first time, caching the results and subsequently reusing them (Koller and Pfeffer, 1997; Pfeffer et al, 1999). Lifted inference is receiving much attention recently, and a detailed account of all developments is beyond the scope of this survey. In the following, we therefore illustrate key ideas focusing on two prominent lines of work that lift popular propositional techniques, namely variable elimination (cf. Sec. 2.2.3) and belief propagation (cf. Sec. 2.2.4). We provide further pointers to the literature in Section 4.3.

4.1.1 Lifted Variable Elimination

First-order variable elimination (FOVE) was introduced by Poole (2003) and later significantly extended in a series of works (de Salvo Braz et al, 2005, 2006; Milch et al, 2008; Apsel and Brafman, 2011; Taghipour et al, 2012, 2013b). As in ordinary variable elimination (VE), cf. Section 2.2.3, the goal of FOVE is to obtain the marginal distribution over a set of random variables $\boldsymbol{X}$ by summing out the values of the remaining variables $\boldsymbol{Y}$, that is, to compute

$$P(\boldsymbol{X} = \boldsymbol{x}) = \sum_{\boldsymbol{Y}=\boldsymbol{y}} \prod_{\Phi_i \in \boldsymbol{\mathcal{F}}} \prod_{\boldsymbol{A} \in \boldsymbol{\mathcal{I}}(\Phi_i)} \phi_i(\boldsymbol{y}_{\boldsymbol{A}}, \boldsymbol{x}_{\boldsymbol{A}}).$$

However, where VE sums out or eliminates one random variable at a time, FOVE sums out an entire group of random variables (grounding the same par-RV) simultaneously. We briefly discuss the main elimination operators below. For more details, we refer the reader to the above papers; de Salvo Braz et al (2007) provide a unified treatment, and Kisyński and Poole (2009a) an excellent basic introduction with examples.

FOVE makes two assumptions on the form of the par-factor graph, each of which can be achieved by using a corresponding auxiliary operation first. The *splitting* operation (Poole, 2003) ensures that the par-factors in the model are *shattered* (de Salvo Braz et al, 2005). Two par-factors Φ_1 and Φ_2 are shattered if the corresponding ground factors are either over the same sets of random variables or over completely disjoint ones, that is, $\boldsymbol{\mathcal{I}}(\Phi_1) = \boldsymbol{\mathcal{I}}(\Phi_2)$ or $\boldsymbol{\mathcal{I}}(\Phi_1) \cap \boldsymbol{\mathcal{I}}(\Phi_2) = \emptyset$, where $\boldsymbol{\mathcal{I}}(\Phi) = \{\boldsymbol{A} \mid \boldsymbol{A} \text{ instance of } \mathbf{A} \text{ under } \boldsymbol{\mathcal{C}}\}$. For instance, $(\{\mathtt{p}(\mathtt{a}, \mathtt{X}), \mathtt{q}(\mathtt{X})\}, \phi_1, \emptyset)$ and $(\{\mathtt{p}(\mathtt{b}, \mathtt{X}), \mathtt{q}(\mathtt{X})\}, \phi_2, \emptyset)$ are shattered, as they do not share any grounding of their par-RVs; but $(\{\mathtt{p}(\mathtt{X}, \mathtt{a}), \mathtt{q}(\mathtt{a})\}, \phi_3, \emptyset)$ and $(\{\mathtt{p}(\mathtt{b}, \mathtt{X}), \mathtt{q}(\mathtt{X})\}, \phi_2, \emptyset)$ are not, as $\{\mathtt{p}(\mathtt{b}, \mathtt{a}), \mathtt{q}(\mathtt{a})\}$ instantiates both. Intuitively, this condition ensures that the same reasoning steps will apply to all of the factors resulting from grounding a given par-factor. The *fusion* operation (de Salvo Braz et al, 2005) ensures that the par-RV Y to be eliminated only participates in one par-factor in the model. It essentially multiplies together all par-factors that depend on Y. To facilitate the remainder of this discussion, let Y be the par-RV to be eliminated, and $\Phi = (\mathbf{A}, \phi, \boldsymbol{\mathcal{C}})$ the single par-factor that

depends on Y, that is, $\mathsf{Y} \in \mathbf{A}$. Thus, we have to compute a sum of products, where the sum is over all value assignments y to all random variables Y_j obtained from Y, and the products are over the instantiations of this par-factor:

$$\sum_{Y_1=y_1} \cdots \sum_{Y_m=y_m} \prod_{\boldsymbol{A} \in \boldsymbol{\mathcal{I}}(\Phi)} \phi(y, \boldsymbol{x}_{\boldsymbol{A} \setminus \{Y\}})$$

The first elimination operation, *inversion elimination* (Poole, 2003; de Salvo Braz et al, 2005), simplifies this sum of products to a product of sums. It only applies if there is a one-to-one correspondence between the groundings of Y and those of $\mathbf{A}$, in which case the sum is

$$\sum_{Y_1=y_1} \cdots \sum_{Y_m=y_m} \prod_{i=1}^{m} \phi(y_i, \boldsymbol{x}_{\boldsymbol{A_i} \setminus \{Y\}}).$$

This condition is violated when the logical variables that appear in Y are different from the logical variables in $\mathbf{A}$. For example, inversion elimination would not work for $\mathsf{Y} = \mathsf{q}(\mathtt{X})$ and $\mathbf{A} = \{\mathsf{q}(\mathtt{X}), \mathsf{p}(\mathtt{X}, \mathtt{Y})\}$, because Y does not depend on the logical variable $\mathtt{Y}$, while $\mathbf{A}$ does and thus can have multiple groundings for each grounding of $\mathtt{X}$. If the condition is satisfied, each random variable Y_i only appears once in the product, and the sum is thus equal to the product of sums

$$\prod_{i=1}^{m} \sum_{Y_i=y_i} \phi(y_i, \boldsymbol{x}_{\boldsymbol{A_i} \setminus \{Y\}}).$$

Each sum now only ranges over the possible truth assignments to a single Y_i (e.g., `true` or `false`), rather than over full truth assignments to all instances of Y.

Another elimination operation is *counting elimination* (de Salvo Braz et al, 2005), which is based on the insight that frequently the factors $(\boldsymbol{A}_i, \phi)$ resulting from grounding Φ form a few large groups with identical members. These groups can be easily identified by considering the possible truth assignments $\boldsymbol{a}$ to the random variables $\boldsymbol{A}_i$. For each such truth assignment, counting elimination counts the number of $\boldsymbol{A}_i$s that would have that truth assignment. Then only one factor from each group needs to be evaluated and the result exponentiated to the total number of factors in that group. For instance, with Boolean values a_i, we have $\phi(x, a_1) \cdot \ldots \cdot \phi(x, a_n) = \phi(x, \text{true})^{c_t} \cdot \phi(x, \text{false})^{c_f}$, where c_t and c_f are the numbers of a_is that are true and false, respectively. Thus, instead of computing the product for exponentially many assignments $a_1, \ldots, a_n$, it suffices to consider linearly many cases (c_t, c_f). For counting elimination to be efficient, the choice of grounding substitutions for any of the par-RVs in $\mathbf{A}$ may not constrain the choice of substitutions for the other ones. Although we have described counting elimination in the context of eliminating the groundings of just one par-RV Y, in fact it can be used to eliminate a set of par-RVs.

Elimination *with counting formulas* (Milch et al, 2008) extends this idea to par-RVs *within* a par-factor. Such par-RVs are exchangeable if $\phi(\boldsymbol{A})$ is a

function of the number of random variables $A \in \boldsymbol{A}$ with a particular value rather than the precise identity of these variables. The extended algorithm is called C-FOVE. Apsel and Brafman (2011) consider counting formulas for joins of atoms, whereas Taghipour et al (2012, 2013b) generalize C-FOVE by decoupling its operators from the language used to specify the constraints on par-RV groundings, resulting in an algorithm, GC-FOVE, that provides more flexibility in grouping computations.

As we discussed in Section 3.4, directed models may require aggregation over a set of values. This may happen, for example, when there is a par-factor in which the parent par-RVs contain logical variables that do not appear in the child par-RV. In order to aggregate over such variables in a lifted fashion, Kisyński and Poole (2009b) introduced *aggregation par-factors* and defined a procedure via which an aggregation par-factor is converted to a product of two par-factors, one of which involves a counting formula. In this way, they are able to handle aggregation using C-FOVE.

FOVE and its extensions are examples of top-down approaches that start from a par-factor graph. The ideas underlying lifted variable elimination can also be exploited bottom-up, that is, starting from a factor graph. Such an approach has been proposed by Sen et al (2008a), who first discover shared factors using bisimulation, and then only perform shared computations once. Bisimulation simulates the operation of VE without actually computing factor values. Larger groups and thus additional speedup can be achieved by approximate inference, where factors are grouped based on similar computations or similar values rather than based on equality (Sen et al, 2009). Another example of a bottom-up approach is the BAM algorithm (Mihalkova and Richardson, 2009), which clusters RVs in the factor graph based on the similarity of their neighborhoods and performs computations only for one representative per cluster.

4.1.2 Lifted Belief Propagation

Lifted BP algorithms (Jaimovich et al, 2007; Singla and Domingos, 2008; Kersting et al, 2009; de Salvo Braz et al, 2009) proceed in two stages. In the first stage, the grounded factor graph $\boldsymbol{\mathcal{F}}$ is compressed into a so-called *template graph* $\boldsymbol{\mathcal{T}}$, in which super-nodes represent groups of variable or factor nodes that send and receive the same messages during BP, similarly to what happens in the approach of Sen et al (2008a) discussed above. Two super-nodes are connected by a super-edge if any of their respective members in $\boldsymbol{\mathcal{F}}$ are connected by an edge, and the weight of the super-edge equals the number of ordinary edges it represents. In the second step, a modified version version of BP is performed on the template graph $\boldsymbol{\mathcal{T}}$. For the sake of understanding, we discuss a simplified version of this algorithm here. The message sent from a variable super-node $\boldsymbol{X}$ to a factor super-node ϕ is given by

$$\mu_{\boldsymbol{X}\to\phi}(\boldsymbol{X}) = \mu_{\phi\to\boldsymbol{X}}(\boldsymbol{X})^{w(\boldsymbol{X},\phi)-1} \cdot \prod_{\phi'\in n(\boldsymbol{X})\setminus\{\phi\}} \mu_{\phi'\to\boldsymbol{X}}(\boldsymbol{X})^{w(\boldsymbol{X},\phi')} \tag{8}$$

In the above expression, $w(\boldsymbol{X}, \phi)$ is the weight of the super-edge between $\boldsymbol{X}$ and ϕ, and $n(\boldsymbol{X})$ is the set of neighbors of $\boldsymbol{X}$ in the template graph. This message thus simulates the one sent from a variable to a factor in the ground case, as given in Equation (5), in that it summarizes the messages received from all factors except the receiving one. Messages from factor super-nodes to variable super-nodes are identical to those in the ground case, and the (unnormalized) result for each variable $\boldsymbol{X}$ is obtained by multiplying all incoming messages exponentiated with the weight of the corresponding super-edge.

Next, we describe how the template factor graph is constructed. The first algorithm was given by Jaimovich et al (2007). This algorithm targets the scenario when no evidence is provided and is based on the insight that in this case, factor nodes and variable nodes can be grouped into types such that two factor/variable nodes are of the same type if they are groundings of the same par-factor/parameterized variable. The lack of evidence ensures that the grounded factor graph is completely symmetrical and any two nodes of the same type have *identical* local neighborhoods, i.e., they have the same numbers of neighbors of each type. As a result, using induction on the iterations of loopy BP, it can be seen that all nodes of the same type send and receive identical messages. As pointed out by Jaimovich et al, the main limitation of this algorithm is that it requires that no evidence be provided, and so it is mostly useful during learning when the data likelihood in the absence of evidence is computed.

Singla and Domingos (2008) built upon the algorithm of Jaimovich et al (2007) and introduced lifted BP for the general case when evidence is provided. In the absence of evidence, their algorithm reduces to that of Jaimovich et al. In this case, the construction of the template graph is a bit more complex and proceeds in stages that simulate BP to determine how the propagation of the evidence affects the types of messages that get sent. Initially, there are three variable super-nodes containing the true, false, and unknown variables respectively. In subsequent iterations, super-nodes are continually refined as follows. First, factor super-nodes are further separated into types such that the factor nodes of each type are functions of the same set of variable super-nodes. Then the variable super-nodes are refined such that variable nodes have the same types if they participate in the same numbers of factor super-nodes of each type. This process is guaranteed to converge, at which point the minimal (i.e., least granular) template factor graph is obtained.

Kersting et al (2009) provide a generalized and simplified description of Singla and Domingos (2008)'s algorithm, casting it in terms of general factor graphs, rather than factor graphs defined by probabilistic logical languages, as was done by Singla and Domingos. Finally, de Salvo Braz et al (2009) have extended lifted BP for the any-time case, combining the approach of Singla and Domingos (2008) with that of Mooij and Kappen (2008).

4.2 Inference on the Instantiated Model

Much of today's work on inference in lifted graphical models focuses on lifted inference. However, especially in the presence of evidence which breaks the symmetries in the model, how to efficiently perform propositional inference in the graphical model obtained by instantiating a lifted graphical model is still of interest. In this section, we discuss ways to reduce the size of the ground model in the first step, which can be combined with any existing inference technique for graphical models, as well as a number of techniques that operate on representations different from a factor graph for the instantiated model, and potentially interleave instantiation and inference to further improve efficiency.

4.2.1 Knowledge-Based Model Construction

Knowledge-based model construction (KBMC) is one of the earliest techniques used to efficiently instantiate a given SRL model (Wellman et al, 1992). It dynamically instantiates the model only to the extent necessary to answer a particular query of interest. KBMC has been adapted to both directed (e.g., Koller and Pfeffer, 1997; Pfeffer et al, 1999; Getoor et al, 2002) and undirected models (e.g., Richardson and Domingos, 2006). Application of KBMC in these and other frameworks exploits the conditional independence properties implied by the factor graph structure of the instantiated model; in particular, the fact that in answering a query about a set of random variables $\boldsymbol{X}$, one only needs to reason about variables that are not rendered conditionally independent of $\boldsymbol{X}$ given the values of observed variables. KBMC can also exploit the structure of par-factor definitions and the evidence, as done in the FROG algorithm (Shavlik and Natarajan, 2009) for MLNs. FROG discards groundings of clauses that are always satisfied because one of their atoms is true according to the evidence, which often results in significantly smaller ground models.

4.2.2 MPE Inference

MPE inference essentially is an optimization task, which, if represented suitably, can be solved using existing methods (e.g., Taskar, 2004; Wainwright et al, 2005; Yanover et al, 2006; Koller and Friedman, 2009). For instance, given evidence $\boldsymbol{Y} = \boldsymbol{y}$, the MPE inference task

$$\text{argmax}_{\boldsymbol{X}=\boldsymbol{x}} \prod_{\phi \in \boldsymbol{\mathcal{F}}} \phi(\boldsymbol{x}_\phi, \boldsymbol{y}_\phi)$$

on a graphical model with discrete random variables can be represented as an integer linear program with a variable $v_\phi^{\boldsymbol{x}}$ for each factor ϕ and each assignment $\boldsymbol{x}$ to the non-evidence random variables $\boldsymbol{X}$. These $v_\phi^{\boldsymbol{x}}$ are restricted to take values 0 or 1, and the program contains constraints requiring that (1) for each factor ϕ exactly one of the $v_\phi^{\boldsymbol{x}}$ is set to 1 at any given time and (2) the values of $v_{\phi_1}^{\boldsymbol{x}}$ and $v_{\phi_2}^{\boldsymbol{x}}$ where ϕ_1 and ϕ_2 share variables are consistent. Intuitively, those

variables "choose" a consistent assignment to the random variables across all factors. Let $\boldsymbol{\mathcal{V}}$ be the set of all such variables $v_\phi^{\boldsymbol{x}}$. MPE inference is then equivalent to solving

$$\text{argmax}_{\boldsymbol{\mathcal{V}}} \sum_{\phi \in \boldsymbol{\mathcal{F}}, \boldsymbol{x}} v_\phi^{\boldsymbol{x}} \cdot \log \phi(\boldsymbol{x}_\phi, \boldsymbol{y}_\phi)$$

subject to those constraints. While solving this integer linear program is still NP-hard in general, it is tractable for certain classes of Markov networks. For instance, Taskar et al (2004) have shown that for *associative* Markov networks – that is, Markov networks whose par-factors favor the same values for RVs in the same clique – it is tractable for binary random variables, and can be closely approximated for the non-binary case.

This view of MPE inference as optimization has been applied for both MLNs (Riedel, 2008) and PSL (Broecheler et al, 2010), but using language-specific representations that take advantage of the structure of the potential functions. For both languages, MPE inference maximizes a weighted sum of feature functions that are defined in terms of logical formulas over ground atoms; namely, the truth value in case of MLNs, and the distance to satisfaction for PSL. The optimization problem contains a variable for each ground atom with unknown truth value, and a variable for each feature function, that is, each ground formula. Its objective function replaces the feature functions in the weighted sum by the corresponding variables, and its constraints relate the values of feature function variables to the value of the underlying formula in terms of the atom variables and the truth values of evidence atoms.

In the case of MLNs, all variables take values 0 or 1, and the constraints express that the value of feature variables has to be equal to the value of the underlying logical formula. Noessner et al (2013) introduce an improved formulation that aims at simplifying inference in the integer linear program by decreasing the size of the program and better exposing its symmetries. Mladenov et al (2012) exploit the link between MPE inference and linear programming on the lifted level and apply the resulting lifted linear programming approach to MLNs.

For Boolean random variables, an alternative way to view the optimization is that of finding a joint assignment to the par-RV instantiations that maximizes the weight of a set of logical formulas, such as the ground instantiations of the clauses in an MLN. In other words, performing MPE inference in such models is equivalent to solving a weighted satisfiability problem using, e.g, the MaxWalkSat algorithm (Kautz et al, 1997), as discussed by Richardson and Domingos (2006). The memory efficiency of this approach can be improved using the general technique of *lazy inference*, that is, by only maintaining *active* RVs and *active* formula instantiations in memory, as done in the LazySAT algorithm (Singla and Domingos, 2006). Initially, all RVs are set to `false`, and the set of active RVs consists of all RVs participating in formula instantiations that are not satisfied by the initial assignment of `false` values. A formula instantiation is activated if it can be made unsatisfied by flipping the value of zero or more active RVs. Thus the initial set of active formula instantiations

consists of those activated by the initially active RVs. The algorithm then carries on with the iterations of MaxWalkSat, activating RVs when their value gets flipped and then activating the relevant rule instantiations.

In the case of PSL, variables take values from $[0, 1]$ and the weights are constrained to be nonnegative. The inference objective is to minimize the negative of the weighted sum of feature functions, which measure the distance to satisfaction of the underlying logical rules. Since the features are convex and the weights are nonnegative, we thus obtain a convex, rather than discrete, optimization task for inference, which is more efficient to solve. Bach et al (2012, 2013) introduce efficient consensus-optimization algorithms to perform inference in this setting.

In practice, in addition to using lazy inference (as discussed above), these approaches do not construct the program for the entire instantiated model up front, but instead interleave program construction and solving, an approach also known as *cutting plane inference* due to its relation to cutting plane algorithms developed in the operations research community. The key observation here is that many formula instantiations are satisfied by setting par-RV instantiations to a default value of `false` (for MLNs) or 0 (for PSL), and that constraints corresponding to such satisfied formulas do not influence the solution of the optimization task. Inference therefore starts from an assignment of default values to all variables, and then iterates between adding constraints for all formulas not satisfied by the current assignment, and solving the resulting extended task to obtain the next assignment. This process continues until a solution that satisfies all constraints is found. In the worst case, it may be necessary to consider the full set of constraints; however, in practice, it is often possible to find a solution based on a small subset only.

4.2.3 Approximate Inference by Sampling

As mentioned in Section 2.2.2, exact inference in graphical models is intractable in general. An alternative approach is to perform approximate inference, based on sampling. Sampling uses the probabilistic model to independently draw a large number of value assignments (samples) to all random variables. It estimates marginal distributions as the relative frequencies of the values occurring among those samples. Sampling from a Bayesian network respecting the order of random variables from parents to children is straightforward; sampling from a Markov network is much more difficult. Furthermore, the presence of evidence imposes additional constraints on the form of useful samples. Markov chain Monte Carlo (MCMC) sampling algorithms form a popular class of approaches addressing these issues. Rather than generating each sample from scratch directly using the graphical model, MCMC algorithms draw a sequence of samples by making random modifications to the current sample based on a so-called *proposal distribution*, which is typically easier to evaluate than the actual distribution of interest. We refer to Bishop (2006, Ch. 11) for a general introduction to sampling and MCMC, and to Koller and Friedman (2009, Ch. 12) for one focused on graphical models.

Gibbs sampling is an example of an MCMC algorithm that has been used with both directed and undirected lifted graphical models, e.g., FACTORIE (McCallum et al, 2009), BLOG (Arora et al, 2010) and MLNs (Richardson and Domingos, 2006). Gibbs sampling repeatedly iterates over all random variables whose values are not fixed by the given evidence, in each step sampling a new value for the current variable V conditioned on the values of all other random variables in the current sample. Due to the independencies encoded by the factor graph, this is equivalent to sampling the value of V conditioned on its Markov blanket, that is, all random variables co-occurring with V in a factor. For many types of graphical models, this distribution can effectively be computed from the neighborhood of V in the factor graph. While Gibbs sampling converges to the target distribution under fairly general conditions (Tierney, 1994), those do not always hold in lifted graphical models. One case where Gibbs sampling can converge to incorrect results is in the presence of deterministic or near-deterministic dependencies, as these can prevent sampling from leaving a certain region in the space of all variable assignments. This problem can be avoided for instance by jointly sampling new values for blocks, or groups, of variables with closely coordinated assignments. An alternative solution is slice sampling (Damien et al, 1999). Slice sampling introduces auxiliary variables to identify "slices" that "cut" across the modes of the distribution. It then alternates between sampling the auxiliary variables given the current values of the original ones, thus identifying a slice, and sampling the original random variables uniformly from the current slice. The MC-SAT algorithm for MLNs is based on slice sampling (Poon and Domingos, 2006). It introduces an auxiliary random variable for each ground clause, and thus each factor, in the MLN. A slice corresponds to a set of ground clauses that have to be satisfied by the next sampled truth value assignment, where clauses with larger weights are more likely to be included in this set. MC-SAT samples (nearly) uniformly from this slice using the SampleSAT algorithm (Wei et al, 2004). Again, lazy inference can be used to restrict the set of random variables that need to be considered explicitly (Poon et al, 2008).

An orthogonal concern is the efficiency of sampling. One approach to speeding up sampling is to use memoization (Pfeffer, 2007), in which values of past samples are stored and reused, instead of generating a new sample. If care is taken to keep reuses independent of one another, the accuracy of sampling can be improved by allowing the sampler to draw a larger number of samples in the allotted time.

A variety of other approaches to and aspects of sampling for lifted graphical models have been discussed in the literature, e.g., a Metropolis-Hastings algorithm for BLOG (Milch and Russell, 2006), an MCMC scheme to compute marginals in PSL (Broecheler and Getoor, 2010), or FACTORIE's support for user-defined MCMC proposal distributions (McCallum et al, 2009). A number of recent works combine lifted inference and sampling (Niepert, 2012; Venugopal and Gogate, 2012; Gogate et al, 2012; Niepert, 2013).

4.3 Discussion

This section has surveyed inference techniques on the instantiated model as well as some of the basic approaches to lifted inference. An overview of lifted inference from the perspective of top-down vs bottom-up inference is given by Kersting (2012), and an in-depth tutorial by Kersting et al (2011). Lifted inference is a very active area of research, and there are many recent publications that have not been discussed here, including work on knowledge compilation (Van den Broeck et al, 2011; Van den Broeck and Davis, 2012; Van den Broeck et al, 2014), message passing (Ahmadi et al, 2011; Hadiji et al, 2011; Kersting et al, 2010a), online inference (Nath and Domingos, 2010), lifted inference for models with continuous variables and Kalman filtering (Choi et al, 2010, 2011a), variational inference (Choi and Amir, 2012; Bui et al, 2013), lifted inference with evidence (Bui et al, 2012; Van den Broeck and Davis, 2012; Van den Broeck and Darwiche, 2013), work that examines the completeness of lifted inference formalisms (Van den Broeck, 2011; Taghipour et al, 2013c; Jaeger and Van den Broeck, 2012; Jaeger, 2014), and many other advanced topics, e.g., (Kiddon and Domingos, 2011; Gogate and Domingos, 2011; Choi et al, 2011b; Gomes and Santos Costa, 2012; Jha et al, 2010; Van den Broeck et al, 2012; Hadiji and Kersting, 2013; Taghipour et al, 2013a; Sarkhel et al, 2014).

5 Learning

The task of learning a lifted graphical model in the form of a par-factor graph can be formalized as follows: given a set of training examples $\boldsymbol{\mathcal{D}}$, that is, assignments $\boldsymbol{x}$ to random variables $\boldsymbol{X}$, a hypothesis space $\boldsymbol{\mathcal{H}}$ in the form of a set of par-factor graphs over $\boldsymbol{X}$, and a scoring function $\text{score}(h, \boldsymbol{\mathcal{D}})$ for $h \in \boldsymbol{\mathcal{H}}$ (typically based on the probability of the training examples), find a hypothesis $h^* \in \boldsymbol{\mathcal{H}}$ that maximizes the score, i.e., $h^* = \arg\max_{h \in \boldsymbol{\mathcal{H}}} \text{score}(h, \boldsymbol{\mathcal{D}})$. Analogous to learning of graphical models, learning of par-factor graphs can be decomposed into *parameter learning* and *structure learning*. In parameter learning, the hypothesis space consists of different parameters for a par-factor graph with given dependency structure, i.e., where all sets of par-RVs $\mathbf{A}_i$ participating together in par-factors, their instantiation constraints $\boldsymbol{\mathcal{C}}_i$, and the general form of potential functions ϕ_i are known, but values for the parameters of these ϕ_i have to be learned. The goal of structure learning, on the other hand, is to discover both the dependency structure of the model and the parameters of the potential functions, that is, the hypothesis space $\boldsymbol{\mathcal{H}}$ no longer uniquely determines the $\mathbf{A}_i$ and $\boldsymbol{\mathcal{C}}_i$. As we will discuss in more detail below, structure learning is often cast as a heuristic search through the space of possible structures, cf. (De Raedt and Kersting, 2010).

Directed and undirected models pose different challenges to learning algorithms. In the case of fully observed data, parameter learning has a closed form solution for directed models, but requires optimization in the undirected

case. When learning the structure of directed models, care has to be taken to ensure acyclicity. Furthermore, structure learning approaches typically learn parameters for many structures with only small local differences, in which case high efficiency gains can be achieved by adapting the parameters of previous structures locally instead of re-learning all parameters from scratch. However, this is only possible if the scoring function is decomposable. This is often the case for directed models, where only the CPDs of nodes whose sets of parents have changed need to be updated. In undirected graphical models, on the other hand, all parameters are connected via the normalization constant Z, and even local changes therefore require adjusting the parameters of the entire model.

5.1 Parameter Learning

Algorithms for parameter learning of graphical models can directly be extended for parameter learning of lifted graphical models. This extension is based on the fact that, as discussed in Section 3.1, an instantiated par-factor graph is simply a factor graph in which subsets of the factors, namely the ones that are instantiations of the same par-factor, have tied parameters. Thus, in its most basic form, parameter learning in par-factor graphs can be reduced to parameter learning in factor graphs by forcing factors that are instantiations of the same par-factor to have their parameters tied.

We now provide a brief overview of basic approaches to parameter learning in graphical models (see Koller and Friedman (2009) for more details) and discuss how they can be easily extended to allow for learning with tied parameters. We follow the common distinction between *generative* approaches such as maximum likelihood or Bayesian parameter estimation, whose aim is to approximate the joint distribution well, and *discriminative* approaches such as max-margin methods, whose aim is to optimize the conditional probability $P(\boldsymbol{X}|\boldsymbol{Y})$ used to predict values of $\boldsymbol{X}$ given evidence $\boldsymbol{Y}$.

For generative models, the simplest case is that of fully observed training data $\boldsymbol{\mathcal{D}}$. In this case, each training example in $\boldsymbol{\mathcal{D}}$ is a complete assignment $\boldsymbol{x}$ to all random variables $\boldsymbol{X}$ in the factor graph $\langle \boldsymbol{X}, \boldsymbol{F} \rangle$, where examples are assumed to be independent and identically distributed (i.i.d.). We denote the vector of learnable parameters in the factors of $\boldsymbol{F}$ by $\boldsymbol{\lambda}$. Maximum likelihood parameter estimation (MLE) uses the likelihood of observing the training data $\boldsymbol{\mathcal{D}}$ as the scoring function $\text{score}(h, \boldsymbol{\mathcal{D}})$, i.e., we are interested in finding parameter values $\boldsymbol{\lambda}^*$ such that

$$\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}} \prod_{\boldsymbol{x} \in \boldsymbol{\mathcal{D}}} P_{\boldsymbol{\lambda}}(\boldsymbol{X} = \boldsymbol{x}). \tag{9}$$

We use subscript $\boldsymbol{\lambda}$ here to emphasize the dependency of P on the parameter values. For directed models, e.g., Bayesian networks, parameter learning means learning a conditional probability distribution (CPD) for each node given its parents. Thus, in the simplest scenario, $\boldsymbol{\lambda}$ consists of the parameters of a set

of conditional probability tables, one for each node. The maximum likelihood estimate for the entry of a node C taking on a value c, given that its parents $\boldsymbol{Pa}(C)$ have values $\boldsymbol{pa}$, is found simply by calculating the proportion of time that configuration of values is observed in $\boldsymbol{\mathcal{D}}$:

$$P^{\mathrm{MLE}}_{\boldsymbol{\mathcal{D}}}(C = c|\boldsymbol{Pa}(C) = \boldsymbol{pa}) = \frac{\mathrm{count}_{\boldsymbol{\mathcal{D}}}(C = c, \boldsymbol{Pa}(C) = \boldsymbol{pa})}{\sum_{c'} \mathrm{count}_{\boldsymbol{\mathcal{D}}}(C = c', \boldsymbol{Pa}(C) = \boldsymbol{pa})} \tag{10}$$

In undirected models, the MLE parameters cannot be calculated in closed form, and one needs to use gradient ascent or some other optimization procedure. Supposing that, as introduced in Section 2.2.1, our representation is a log-linear model with one parameter per factor, then the gradient of the data log-likelihood with respect to the parameter λ_i of a potential function $\phi_i(\boldsymbol{X}) = \exp(\lambda_i \cdot f_i(\boldsymbol{X}))$ is given by:

$$\frac{\partial \log \prod_{\boldsymbol{x} \in \boldsymbol{\mathcal{D}}} P_{\boldsymbol{\lambda}}(\boldsymbol{X} = \boldsymbol{x})}{\partial \lambda_i} = \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{D}}} (f_i(\boldsymbol{x}_i) - \mathbb{E}_{\boldsymbol{\lambda}}[f_i(\boldsymbol{y}_i)]) \tag{11}$$

Here, $\boldsymbol{x}_i$ are the values in $\boldsymbol{x}$ for the variables participating in ϕ_i, and $\mathbb{E}_{\boldsymbol{\lambda}}[f_i(\boldsymbol{y}_i)]$ is the expected value of f_i according to the current estimate for all parameters $\boldsymbol{\lambda}$.

In the case where the data is not fully observed, that is, each example in $\boldsymbol{\mathcal{D}}$ assigns values to a subset of the random variables $\boldsymbol{X}$ only, the standard approach is to resort to an expectation-maximization (EM) algorithm, which requires to perform inference during parameter learning to estimate unobserved values.

We next describe how Equations (10) and (11) are extended to work with tied parameters coming from par-factors. This is done by computing counts and function values on the level of par-factors rather than factors, that is, by aggregating them over all factors that instantiate the same par-factor. In the relational setting, the training data $\boldsymbol{\mathcal{D}}$ often consists of a single "mega-example" that assigns values $\boldsymbol{x}$ to the random variables $\boldsymbol{X}$ in a factor graph $\langle \boldsymbol{X}, \boldsymbol{F} \rangle$ grounding the par-factor graph of interest for a specific domain. Because of parameter tying, such an example typically contains many, often interdependent, instances of each par-factor, which parameter learning approaches treat as i.i.d. data.

In directed models, factors with tied parameters share their CPDs. Thus, in this case, in Equation (10) counts are computed not just for a single node, or instantiation of a par-factor, but for all nodes that are instantiations of that par-factor and thus share their CPD. Let $\boldsymbol{C}$ be that set of nodes, and let $\boldsymbol{Pa}(C)$ be the set of parents of node C. Then for all $C \in \boldsymbol{C}$, Equation (10) becomes:

$$P^{\mathrm{MLE}}_{\boldsymbol{\mathcal{D}}}(C = c|\boldsymbol{Pa}(C) = \boldsymbol{pa}) = \frac{\sum_{C \in \boldsymbol{C}} \mathrm{count}_{\boldsymbol{\mathcal{D}}}(C = c, \boldsymbol{Pa}(C) = \boldsymbol{pa})}{\sum_{C \in \boldsymbol{C}} \sum_{c'} \mathrm{count}_{\boldsymbol{\mathcal{D}}}(C = c', \boldsymbol{Pa}(C) = \boldsymbol{pa})} \tag{12}$$

In the undirected case, instead of a separate instance of Equation (11) for each factor, we now get one gradient for each par-factor Φ_i's parameter λ_i, summarizing the ones for all its instantiations:

$$\frac{\partial \log \prod_{\boldsymbol{x} \in \boldsymbol{\mathcal{D}}} P_{\boldsymbol{\lambda}}(\boldsymbol{X} = \boldsymbol{x})}{\partial \lambda_i} = \sum_{\boldsymbol{A} \in \boldsymbol{\mathcal{I}}(\Phi_i)} \sum_{\boldsymbol{x} \in \boldsymbol{\mathcal{D}}} (f_i(\boldsymbol{x}_{\boldsymbol{A}}) - \mathbb{E}_{\boldsymbol{\lambda}}[f_i(\boldsymbol{y}_{\boldsymbol{A}})]) \tag{13}$$

As before, $\boldsymbol{\mathcal{I}}(\Phi_i)$ is the set of factors that are instantiations of Φ_i, and $\boldsymbol{x}_{\boldsymbol{A}}$ are the values for the random variables $\boldsymbol{A}$ in an instantiation of par-factor Φ_i.

While the above discussion focused on one particular scoring function, that of maximum likelihood estimation, in practice other scoring functions exist. For example, rather than optimizing the data likelihood, one can significantly improve efficiency by instead optimizing the pseudo-likelihood (Besag, 1975). To do so, the joint probability $P_{\boldsymbol{\lambda}}(\boldsymbol{X} = \boldsymbol{x})$ in Equation (9) is replaced by $\prod_{X \in \boldsymbol{X}} P_{\boldsymbol{\lambda}}(X = x | \boldsymbol{X}_{MB} = \boldsymbol{x}_{MB})$, the product of the conditional probability of each RV X given the variables $\boldsymbol{X}_{MB}$ in its *Markov blanket*, that is, all RVs appearing together with X in some factor. While using the pseudo-likelihood avoids the computational complexity of dealing with the partition function, the price to be paid for the increased efficiency is that it may no longer be possible to learn a model that covers all dependencies. Again, in the case of lifted models, all instantiations of the ith par-factor contribute to the sufficient statistics used to estimate λ_i.

An alternative to maximum (pseudo-)likelihood that is used, for instance, to reduce overfitting, is Bayesian learning, where one imposes a prior probability distribution over the parameters that are learned, thus defining a joint distribution over parameters and data (e.g., Heckerman, 1999; Koller and Friedman, 2009).

Generative approaches to parameter learning in lifted graphical models have been developed for instance for PRMs, both with respect to a maximum likelihood criterion and a Bayesian criterion (Getoor, 2002), for PSL (Broecheler et al, 2010; Bach et al, 2013), and for MLNs, where several approaches to improve efficiency of gradient descent parameter learning methods have been considered (Lowd and Domingos, 2007).

Discriminative approaches to parameter learning are motivated by the fact that probabilistic models are often used to predict the values of one set of random variables $\boldsymbol{X}$ given the values of the remaining variables $\boldsymbol{Y}$, in which case it is sufficient to optimize the conditional probability $P(\boldsymbol{X}|\boldsymbol{Y})$ rather than the joint probability $P(\boldsymbol{X}, \boldsymbol{Y})$. Specifically, max-margin approaches as introduced by Taskar et al (2003) learn parameters that maximize the margin between the probability of the correct assignment $\boldsymbol{x}$ given $\boldsymbol{y}$ and that of other assignments $\boldsymbol{x'}$. For lifted graphical models, discriminative parameter learning has been considered e.g., for MLNs (Singla and Domingos, 2005; Huynh and Mooney, 2009, 2011) and PSL (Bach et al, 2013).

One issue that arises when learning the parameters of an SRL model as described above is computing the sufficient statistics, e.g., the counts in Equation (12) and the sums in Equation (13). Models that are based on a database

Algorithm 1 Structure Learning Algorithm (instantiation of procedures in lines 2, 3, 5 and 8 determines exact behavior)

Input: Hypothesis space $\boldsymbol{\mathcal{H}}$ (describing par-factor graphs), training data $\boldsymbol{\mathcal{D}}$ (assignments to random variables), scoring function score$(\cdot, \boldsymbol{\mathcal{D}})$
Output: A par-factor graph $h \in \boldsymbol{\mathcal{H}}$
Procedure:
1: $\boldsymbol{\mathcal{G}} \leftarrow \emptyset$; $h \leftarrow \emptyset$;
2: **while** CONTINUE$(\boldsymbol{\mathcal{G}}, h, \boldsymbol{\mathcal{H}}, \text{score}(\cdot, \boldsymbol{\mathcal{D}}))$ **do**
3: $\quad \boldsymbol{\mathcal{R}} \leftarrow$REFINECANDIDATES$(\boldsymbol{\mathcal{G}}, \boldsymbol{\mathcal{H}})$
4: $\quad$ **for each** $r \in \boldsymbol{\mathcal{R}}$ **do**
5: $\quad\quad r \leftarrow$LEARNPARAMETERS$(r, \text{score}(\cdot, \boldsymbol{\mathcal{D}}))$
6: $\quad$ **end for**
7: $\quad h \leftarrow \arg\max_{h' \in \boldsymbol{\mathcal{R}} \cup \{h\}} \text{score}(h', \boldsymbol{\mathcal{D}})$
8: $\quad \boldsymbol{\mathcal{G}} \leftarrow$SELECT$(\boldsymbol{\mathcal{R}}, \text{score}(\cdot, \boldsymbol{\mathcal{D}}))$
9: **end while**
10: **return** h

representation can take advantage of database operations to compute sufficient statistics efficiently. For example, in PRMs, the computation of sufficient statistics is cast as the construction of an appropriate view of the data, on which simple database queries are run to obtain the statistics (Getoor, 2002). Caching is used to achieve further speed-ups.

Another issue for parameter learning in undirected SRL models is computing the expectations in Equation (13), which is intractable in general. This issue has been addressed for instance by using sampling to approximate the expectations (Richardson and Domingos, 2006), by using the values in the MAP state as expectations (Singla and Domingos, 2005; Broecheler et al, 2010), or by using the pseudo-likelihood as discussed above.

Using lifted inference for parameter learning is challenging, as evidence often breaks the symmetries in the model and makes lifted techniques fall back on propositional techniques. Ahmadi et al (2012) address this problem by decomposing the factor graph into possibly overlapping pieces, exploiting symmetries for lifted inference locally on the level of pieces rather than globally. Their online learning method then iterates over these pieces to update parameters. Ahmadi et al (2013) further scale up this approach by extending it to a MapReduce setting.

5.2 Structure Learning

The goal of structure learning is to find the skeleton of dependencies and regularities that make up the set of par-factors. Structure learning in SRL builds heavily on corresponding work in graphical models and inductive logic programming (ILP). Algorithm 1 shows a schematic structure learning procedure that realizes a search for the best par-factor graph h in the space $\boldsymbol{\mathcal{H}}$ of possible par-factor graphs according to the scoring function score$(\cdot, \boldsymbol{\mathcal{D}})$ on the training data set $\boldsymbol{\mathcal{D}}$. As for parameter learning, the data $\boldsymbol{\mathcal{D}}$ often consists of a single, interconnected "mega-example" containing many ground instances of

the par-RVs of interest and their relations. The schematic algorithm relies on a number of procedures (names in CAPS) that need to be instantiated to obtain a concrete algorithm. The algorithm proceeds in iterations until a stopping criterion is met (line 2, procedure CONTINUE). In each iteration, a set $\boldsymbol{\mathcal{R}}$ of new candidate par-factor graph structures is derived from the current set of par-factor graphs $\boldsymbol{\mathcal{G}}$ (line 3, procedure REFINECANDIDATES), and for each of those candidate structures, parameters are learned (line 5, procedure LEARNPARAMETERS). Then, the current best hypothesis h is determined (line 7), and a subset of $\boldsymbol{\mathcal{R}}$ is selected to be passed on to the next round (line 8, procedure SELECT). Finally, the best scoring model is returned. In principle, this algorithm could be instantiated to perform a complete search of the hypothesis space $\boldsymbol{\mathcal{H}}$, but typically, some form of greedy heuristic search will be realized. REFINECANDIDATES specifies how new par-factor graph structures are derived from a given one. Initially, this will typically produce trivial par-factors, e.g., ones consisting of single par-RVs, while later on, it will perform several kinds of simple incremental changes, such as the addition or removal of a par-RV in a par-factor. Algorithm 1 is directly analogous to approaches for learning in graphical models, such as those by Della Pietra et al (1997) and Heckerman (1999), as well as to approaches developed in ILP, such as the FOIL algorithm (Quinlan, 1990). Variants of this algorithm, adapted to the particular SRL representation, have been used by several authors. We will illustrate such techniques for both directed and undirected models below, focusing on PRMs and MLNs as representatives of the two classes, respectively. One of the difficulties of learning the structure of par-factor graphs via search, as performed in Algorithm 1, is that the space over possible structures is very large and contains many local maxima and plateaus. Two ways to address these challenges are to modify the type of search performed (roughly, the SELECT procedure), or to restrict the hypothesis space $\boldsymbol{\mathcal{H}}$ to be searched using some form of pre-processing.

Directed models An instantiation of the general algorithm that learns PRMs is described by Getoor (2002). In this case, the REFINECANDIDATES method checks for acyclicity in the resulting structure and employs classic revision operators for directed graphical models, such as adding, deleting, or reversing an edge. In addition to a greedy hill-climbing algorithm that always prefers high-scoring structures over lower-scoring ones, Getoor (2002) presents a randomized technique with a simulated annealing flavor where at the beginning of learning the structure search procedure takes random steps with some probability p and greedy steps with probability $1 - p$. As learning progresses, p is decreased, gradually steering learning away from random choices.

One approach to reduce the hypothesis space, used for PRM learning, is to constrain the set of potential parents of each par-RV X (Friedman et al, 1999a). This algorithm proceeds in stages, in each stage k forming the set of potential parents of X as those par-RVs that can be reached from X through a chain of relations of length at most k. Structure learning at stage k is then constrained to search only over those potential parent sets. The algorithm

further constrains potential parent candidates by requiring that they "add value" beyond what is already captured in the currently learned set of parents. More specifically, the set of potential parents of par-RV X at stage k consists of the parents in the learned structure from stage $k-1$, and any par-RVs reachable through relation chains of length at most k that lead to a higher value in a specially designed score measure. This algorithm directly ports scoring functions that were developed for an analogous learning technique for Bayesian networks (Friedman et al, 1999b).

Undirected models For the case of undirected models, Kok and Domingos (2005) introduced a version of the search-based structure learning algorithm for MLNs. Their algorithm proceeds in iterations, each time searching for the best clause to add to the model. Searching can be performed using one of two possible strategies. The first one, beam search, keeps the best k clause candidates at each step of the search. On the other hand, with the second one, shortest-first search, the algorithm tries to find the best clauses of length i before it moves on to length $i+1$. Candidate clauses in this algorithm are scored using the weighted pseudo log-likelihood measure, an adaptation of the pseudo log-likelihood that weighs the pseudo likelihood of each grounded atom by 1 over the number of groundings of its predicate to prevent predicates with larger arity from dominating the expression.

Iterative local search techniques (Lourenço et al, 2003) alternate between two types of search steps, either moving towards a locally optimal solution, or perturbing the current solution in order to escape from local optima. This approach has been used to avoid local maxima when learning MLNs in a discriminative setting, where the focus is on predicting a specific target predicate given evidence on all other predicates (Biba et al, 2008).

An alternative approach is to search for structures of increasing complexity, at each stage using the structures found at the previous stage to constrain the search space. Such a strategy was employed by Khosravi et al (2010) for learning MLN structure in domains that contain many descriptive attributes. Their approach, which is similar to the technique employed to constrain the search space in PRMs (Friedman et al, 1999a) described above, distinguishes between two types of tables – attribute tables that describe a single entity type, and relationship tables that describe relationships between entities. The algorithm, called MBN, then proceeds in three stages. In the first stage dependencies local to attribute tables are learned. In the second stage, dependencies over a join of an attribute table and a relationship table are learned, but the search space is constrained by requiring that all dependencies local to the attribute table found in the first stage remain the same. Finally, in the third stage dependencies over a join of two relationship tables, joined with relevant attribute tables, are learned, and the search space is similarly constrained. An orthogonal characteristic of MBN is that, although the goal is to learn an undirected SRL model, dependencies are learned using a Bayesian network learner. The directed structures are then converted to undirected ones by "moralizing" the graphs (i.e., by adding edges between all pairs of parents of the same node

and dropping edge directions). The advantage of this approach is that structure learning in directed models is significantly faster than structure learning in undirected models due to the decomposability of the score, which allows it to be updated locally, only in parts of the structure that have been modified, and thus scoring of candidate structures is more efficient. Schulte (2011) introduces a pseudo-likelihood measure for directed par-factor graphs and shows that the algorithm of Khosravi et al (2010) can be seen as optimizing this measure. This algorithm has also been combined with a decision tree learner to obtain more compact models (Khosravi et al, 2012), and generalized into a learning framework that organizes the search space as a lattice (Schulte and Khosravi, 2012). The latter also incorporates learning recursive dependencies in the directed model as introduced by Schulte et al (2012).

A series of algorithms have been developed to restrict the hypothesis space for MLN structure learning. The first in the series was BUSL (Mihalkova and Mooney, 2007), which is based on the observation that, once an MLN is instantiated into a Markov network, the instantiations of each clause of the MLN define a set of identically structured cliques in the Markov network. BUSL inverts this process of instantiation and constrains the search space by first inducing lifted templates for such cliques by learning a so-called Markov network template, an undirected graph of dependencies whose nodes are not ordinary variables but par-RVs. Then clause search is constrained to the cliques of this Markov network template. Markov network templates are learned by constructing, from the perspective of each predicate, a table in which there is a row for each possible instantiation of the predicate and a column for possible par-RVs, with the value of a cell i, j being set to 1 if the data contains a true instantiation of the jth par-RV such that variable substitutions are consistent with the ith predicate instantiation. The Markov network template is learned from this table by any Markov network learner.

A further MLN learner that is based on constraining the search space is the LHL algorithm (Kok and Domingos, 2009). LHL limits the set of clause candidates that are considered by using relational pathfinding (Richards and Mooney, 1992) to focus on more promising ones. Developed in the ILP community, relational pathfinding searches for clauses by tracing paths across the true instantiations of relations in the data. Figure 10 gives an example in which the clause $\texttt{Credits}(\texttt{C},\texttt{A}) \wedge \texttt{Credits}(\texttt{C},\texttt{B}) \Rightarrow \texttt{WorkedFor}(\texttt{A},\texttt{B})$ is learned by tracing the thick-lined path between `brando` and `coppola` and variablizing appropriately. However, because in real-world relational domains the search space over relational paths may be very large, a crucial aspect of LHL is that it does not perform relational pathfinding over the original relational graph of the data but over a so-called *lifted hypergraph*, which is formed by clustering the entities in the domain via an agglomerative clustering procedure, itself implemented as an MLN. Intuitively, entities are clustered together if they tend to participate in the same kinds of relations with entities from other clusters. Structure search is then limited only to clauses that can be derived as relational paths in the lifted hypergraph.

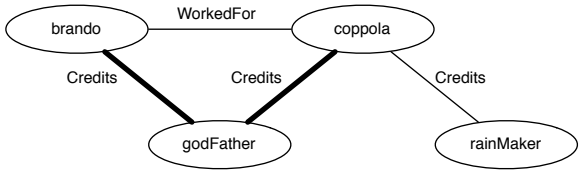


Fig. 10: Example of relational pathfinding.

Kok and Domingos (2010) have proposed constraining the search space by identifying so-called *structural motifs*, which capture commonly occurring patterns among densely connected entities in the domain. The resulting algorithm, called LSM, proceeds by first identifying motifs and then searching for clauses by performing relational pathfinding within them. To discover motifs, LSM starts from an entity i in the relational graph and performs a series of random walks. Entities that are reachable within a thresholded hitting time and the hyperedges among them are included in the motif and the paths via which they are reachable from i are recorded. Next, the entities included in the motif are clustered by their hitting times into groups of potentially symmetrical nodes. The nodes within each group are then further clustered in an agglomerative manner by the similarity of distributions over paths via which they are reachable from i. This process results in a lifted hypergraph, analogous to the one produced by LHL; however, whereas in LHL nodes were clustered based on their close neighborhood in the relational graph, here they are clustered based on their longer-range connections to other nodes. Motifs are extracted from the lifted hypergraphs via depth-first search.

Structure learning techniques that do not follow the search-based pattern of Algorithm 1 have been developed as well. One technique developed in the graphical models community that has been extended to par-factor graphs is that of structure selection through appropriate regularization. In this approach (Lee et al, 2006; Lowd and Davis, 2010), a large number of factors of a Markov network are evaluated at once by training parameters over them and using the L_1 norm as a regularizer (as opposed to the typically used L_2 norm). Since the L_1 norm imposes a strong penalty on smaller parameters, its effect is that it forces more parameters to 0, which are then pruned from the model. Huynh and Mooney (2008) extended this technique for structure learning of MLNs by first using Aleph (Srinivasan, 2001), an off-the-shelf ILP learner, to generate a large set of potential par-factors (in this case, first-order clauses), and then performed L_1-regularized parameter learning over this set.

Khot et al (2011) have extended the functional gradient boosting approach to learning relational dependency networks of Natarajan et al (2012) to MLNs. In contrast to previous approaches, they learn structure and parameters simultaneously, thus avoiding the cost of repeated parameter estimation. Essentially, for each par-RV to be queried, the approach learns a set of non-recursive Horn clauses with that par-RV in the head. This is done through a sequence of

functional gradient steps, each of which adds clauses based on the point-wise gradients of the training examples, that is, the ground instances of the respective par-RV, in the current model.

5.2.1 Structure Revision and Transfer Learning

Our discussion so far has focused on learning structure from scratch. While approaches based on search, such as Algorithm 1, can be easily adapted to perform revision by initializing them with a given structure, some work in the area has also focused on approaches specifically designed for structure revision and transfer learning. For example, Paes et al (2005) introduced an approach for revision of BLPs based on work in theory revision in the ILP community, where the goal is, given an initial theory, to minimally modify it such that it becomes consistent with a set of examples. The BLP revision algorithm follows the methodology of the FORTE theory revision system (Richards and Mooney, 1995), first generating revision points in places where the given set of rules fails and next focusing the search for revisions to ones that could address the discovered revision points. The FORTE methodology was also followed in TAMAR, an MLN transfer learning system (Mihalkova et al, 2007), which generates revision points on MLN clauses by performing inference and observing the ways in which the given clauses fail. TAMAR was designed for transfer learning (e.g., Banerjee et al, 2006), where the goal is to first map, or translate, the given structure from the representation of a source domain to that of a target and then to revise it. Thus, in addition to the revision module, it also contains a mapping module, which discovers the best mapping of the source predicates to the target ones. The problem of mapping a source structure to a target domain was also considered in the constrained setting where data in the target domain is extremely scarce (Mihalkova and Mooney, 2009).

Rather than taking a structure learned specifically for a source domain and trying to adapt it to a target domain of interest, an alternative approach to transfer learning is to extract general knowledge in the source domain that can then be applied to a variety of target domains. This is the approach taken in DTM (Davis and Domingos, 2009), which uses the source data to learn general clique templates expressed as second-order Markov logic clauses, i.e., with quantification both over the predicates and the variables. During this step, care is taken to ensure that the learned clique templates capture general regularities and are not likely to be specific to the source domain. Then, in the target domain DTM allows for several possible mechanisms for using the clique templates to define the hypothesis space.

5.2.2 Learning Causal Models

Learning the causal structure in a domain is an important type of structure learning task that is receiving growing attention (Pearl, 2009), but is notoriously difficult given observational data only. As many have argued, there are

advantages to building models that are causal, which, assuming that one has the right set of variables, tend to be simpler models (e.g., Pearl, 1988; Heckerman, 1999; Koller and Friedman, 2009). Many SRL models are based on rules, which makes it tempting to interpret the direction of these rules as the direction of causal influence. However, as in the propositional case, structure learning approaches are typically based on correlation rather than causation between variables, and therefore do not necessarily justify this interpretation. Specifically, knowing the joint distribution, or correlations, between random variables is often not sufficient to make decisions or take actions that result in changes to other variables of interest in the domain. This additionally requires knowledge about the underlying mechanisms of the domain, that is, about which variable values, if changed, will change the values of which other variables. More generally, if one wishes to make scientific discoveries, this requires discovering and understanding the underlying causal processes in the domain.

Despite its growing importance, learning causal models has so far received little attention in the SRL community. Recent examples are the algorithms of Maier et al (2010, 2013), who build upon principles that infer the directionality of rules used for causal discovery in propositional domains by Spirtes et al (2001), and the work by Rattigan et al (2011), who introduce a strategy to factor out common causes by grouping entities with a common neighbor in a relational structure.

5.3 Discussion

This section has surveyed learning for lifted graphical models. While there are important differences in approaches to learning in directed versus undirected models, there are many important commonalities as well. Parameter learning often requires the ability to perform inference, as such, it relies on methods for inference in lifted graphical models. Structure learning often involves some form of search over potential rules or factors in some systematic yet tractable manner. Beyond the work described here, examples of recent work in structure learning include (Lowd, 2012; Nath and Richardson, 2012; Khot et al, 2013).

6 Conclusion

Multi-relational data, in which entities of different types engage in a rich set of relations, is ubiquitous in many domains of current interest, such as social networks, computational biology, web and social media applications, natural language processing, automatic knowledge acquisition, and many more. Furthermore, for applications to be successful, modeling and reasoning needs to simultaneously address the inherent uncertainty often present in such domains as well as their relational structure. Learning in such settings is much more challenging as well, as the classical assumption of i.i.d. data no longer applies. Instead, we face highly structured but noisy data, often in the form of a single,

large, interconnected example or network. While SRL provides powerful tools to address this challenge, it is still a young field with many open questions, concerning specific inference and learning settings as discussed throughout this paper, but also fundamental questions on the theory of learning in this setting and the guarantees that can or cannot be achieved. In this survey, we have provided a synthesis of the current state of the field by outlining the main ideas underlying representation, inference and learning of lifted graphical models. We have reviewed a general form for a lifted graphical model, a par-factor graph, and shown how a number of existing statistical relational representations map to this formalism. We have discussed inference algorithms, including lifted inference algorithms, that efficiently compute the answers to probabilistic queries. We have also reviewed work in learning lifted graphical models from data. It is our belief that the need for statistical relational models (whether they go by that name or another) will grow in the coming decades, as we are inundated with structured and unstructured data, including noisy relational data automatically extracted from text and noisy information from sensor networks, and with the need to reason effectively with this data. We expect to see further applications of SRL methods in such domains, and we hope that this synthesis of ideas from many different research groups will provide an accessible starting point for new researchers in this expanding field.

Acknowledgements We would like to thank Galileo Namata and Theodoros Rekatsinas for their comments on earlier versions of this paper. A. Kimmig is a postdoctoral fellow of the Research Foundation Flanders (FWO Vlaanderen). L. Mihalkova is supported by a CI fellowship under NSF Grant # 0937060 to the Computing Research Association. L. Getoor is supported by NSF Grants # IIS0746930 and # CCF0937094. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or the CRA.

References

Ahmadi B, Kersting K, Sanner S (2011) Multi-evidence lifted message passing, with application to PageRank and the Kalman filter. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)

Ahmadi B, Kersting K, Natarajan S (2012) Lifted online training of relational models with stochastic gradient methods. In: Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD-12)

Ahmadi B, Kersting K, Mladenov M, Natarajan S (2013) Exploiting symmetries for scaling loopy belief propagation and relational training. Machine Learning 92(1):91–132

Airoldi EM, Blei DM, Fienberg SE, Xing EP (2008) Mixed membership stochastic blockmodels. Journal of Machine Learning Research 9:1981–2014

Apsel U, Brafman RI (2011) Extended lifted inference with joint formulas. In: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI-11)

Arora NS, de Salvo Braz R, Sudderth EB, Russell SJ (2010) Gibbs sampling in open-universe stochastic languages. In: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI-10)

Bach SH, Broecheler M, Getoor L, O'Leary DP (2012) Scaling MPE inference for constrained continuous Markov random fields with consensus optimization. In: Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS-12)

Bach SH, Huang B, London B, Getoor L (2013) Hinge-loss Markov random fields: Convex inference for structured prediction. In: Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI-13)

Bakir GH, Hofmann T, Schölkopf B, Smola AJ, Taskar B, Vishwanathan SVN (eds) (2007) Predicting Structured Data. MIT Press

Banerjee B, Liu Y, Youngblood GM (eds) (2006) Structural Knowledge Transfer for Machine Learning. Workshop at ICML-06

Besag J (1975) Statistical analysis of non-lattice data. The Statistician 24(3):179–195

Biba M, Ferilli S, Esposito F (2008) Discriminative structure learning of Markov logic networks. In: Proceedings of the 18th International Conference on Inductive Logic Programming (ILP-08)

Bishop CM (2006) Pattern Recognition and Machine Learning. Springer

Broecheler M, Getoor L (2010) Computing marginal distributions over continuous Markov networks for statistical relational learning. In: Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS-10)

Broecheler M, Mihalkova L, Getoor L (2010) Probabilistic similarity logic. In: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI-10)

Bui HH, Huynh TN, de Salvo Braz R (2012) Exact lifted inference with distinct soft evidence on every object. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-12)

Bui HH, Huynh TN, Riedel S (2013) Automorphism groups of graphical models and lifted variational inference. In: Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI-13)

Chang J, Blei DM (2009) Relational topic models for document networks. In: Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS-09)

Choi J, Amir E (2012) Lifted relational variational inference. In: Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI-12)

Choi J, Amir E, Hill DJ (2010) Lifted inference for relational continuous models. In: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI-10)

Choi J, Guzmán-Rivera A, Amir E (2011a) Lifted relational Kalman filtering. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)

Choi J, de Salvo Braz R, Bui HH (2011b) Efficient methods for lifted inference with aggregate factors. In: Proceedings of the 25th AAAI Conference on

Artificial Intelligence (AAAI-11)
Chu W, Sindhwani V, Ghahramani Z, Keerthi SS (2006) Relational learning with Gaussian processes. In: Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS-06)
Damien P, Wakefield J, Walker S (1999) Gibbs sampling for Bayesian non-conjugate and hierarchical models by using auxiliary variables. Journal of the Royal Statistical Society 61(2):331–344
Davis J, Domingos P (2009) Deep transfer via second-order Markov logic. In: Proceedings of the 26th International Conference on Machine Learning (ICML-09)
De Raedt L (1996) Advances in Inductive Logic Programming. IOS Press
De Raedt L (2008) Logical and relational learning. Springer
De Raedt L, Kersting K (2003) Probabilistic logic learning. ACM-SIGKDD Explorations: Special Issue on Multi-relational Data Mining 5(5):31–48
De Raedt L, Kersting K (2004) Probabilistic inductive logic programming. In: Proceedings of the 15th International Conference on Algorithmic Learning Theory (ALT-04)
De Raedt L, Kersting K (2010) Statistical relational learning. In: Sammut C, Webb GI (eds) Encyclopedia of Machine Learning, Springer
De Raedt L, Kimmig A, Toivonen H (2007) ProbLog: A probabilistic Prolog and its application in link discovery. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)
De Raedt L, Frasconi P, Kersting K, Muggleton S (eds) (2008) Probabilistic Inductive Logic Programming. Springer
de Salvo Braz R, Amir E, Roth D (2005) Lifted first-order probabilistic inference. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)
de Salvo Braz R, Amir E, Roth D (2006) MPE and partial inversion in lifted probabilistic variable elimination. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)
de Salvo Braz R, Amir E, Roth D (2007) Lifted first-order probabilistic inference. In: Getoor L, Taskar B (eds) Introduction to Statistical Relational Learning, MIT Press
de Salvo Braz R, Amir E, Roth D (2008) A survey of first-order probabilistic models. In: Holmes DE, Jain LC (eds) Innovations in Bayesian Networks, Springer
de Salvo Braz R, Natarajan S, Bui H, Shavlik J, Russell S (2009) Anytime lifted belief propagation. In: Proceedings of the International Workshop on Statistical Relational Learning (SRL-09)
Della Pietra S, Della Pietra VJ, Lafferty JD (1997) Inducing features of random fields. IEEE Transactions on Pattern Analysis and Machine Intelligence 19(4):380–393
Dietterich T, Getoor L, Murphy K (eds) (2004) Statistical Relational Learning and its Connections to Other Fields (SRL-04)
Domingos P, Kersting K (eds) (2009) International Workshop on Statistical Relational Learning (SRL-09)

Domingos P, Lowd D (2009) Markov Logic: An Interface Layer for Artificial Intelligence. Morgan & Claypool

Džeroski S, Lavrač N (eds) (2001) Relational Data Mining. Springer

Fern A, Getoor L, Milch B (eds) (2006) Open Problems in Statistical Relational Learning (SRL-06). Workshop at ICML-06

Friedman N, Getoor L, Koller D, Pfeffer A (1999a) Learning probabilistic relational models. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)

Friedman N, Nachman I, Peér D (1999b) Learning Bayesian network structure from massive datasets: The "sparse candidate" algorithm. In: Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)

Gaudel R, Sebag M, Cornuéjols A (2007) A phase transition-based perspective on multiple instance kernels. In: Proceedings of the 17th International Conference on Inductive Logic Programming (ILP-07)

Getoor L (2002) Learning statistical models from relational data. PhD thesis, Stanford University

Getoor L, Taskar B (eds) (2007) Introduction to Statistical Relational Learning. MIT Press

Getoor L, Friedman N, Koller D, Taskar B (2002) Learning probabilistic models of link structure. Journal of Machine Learning Research 3:679–707

Getoor L, Friedman N, Koller D, Pfeffer A, Taskar B (2007) Probabilistic relational models. In: Getoor L, Taskar B (eds) Statistical Relational Learning, MIT Press

Gogate V, Domingos P (2011) Probabilistic theorem proving. In: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI-11)

Gogate V, Jha AK, Venugopal D (2012) Advances in lifted importance sampling. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-12)

Gogate V, Kersting K, Natarajan S, Poole D (eds) (2013) Statistical Relational AI Workshop at AAAI-13

Gomes T, Santos Costa V (2012) Evaluating inference algorithms for the Prolog factor language. In: Proceedings of the 22nd International Conference on Inductive Logic Programming (ILP-12)

Hadiji F, Kersting K (2013) Reduce and re-lift: Bootstrapped lifted likelihood maximization for MAP. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI-13)

Hadiji F, Ahmadi B, Kersting K (2011) Efficient sequential clamping for lifted message passing. In: Proceedings of the 34th Annual German Conference on AI (KI-11)

Heckerman D (1999) A tutorial on learning with Bayesian networks. In: Jordan M (ed) Learning in Graphical Models, MIT Press

Heckerman D, Breese JS (1994) A new look at causal independence. In: Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence (UAI-94)

Heckerman D, Chickering DM, Meek C, Rounthwaite R, Kadie C (2000) Dependency networks for inference, collaborative filtering and data visualiza-

tion. Journal of Machine Learning Research 1:49–75
Huynh TN, Mooney RJ (2008) Discriminative structure and parameter learning for Markov logic networks. In: Proceedings of the 25th International Conference on Machine Learning (ICML-08)
Huynh TN, Mooney RJ (2009) Max-margin weight learning for Markov logic networks. In: Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD-09)
Huynh TN, Mooney RJ (2011) Online max-margin weight learning for Markov logic networks. In: Proceedings of the Eleventh SIAM International Conference on Data Mining (SDM-11)
Jaeger M (1997) Relational Bayesian networks. In: Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI-97)
Jaeger M (2002) Relational Bayesian networks: A survey. Linköping Electronic Articles in Computer and Information Science 7(015)
Jaeger M (2014) Lower complexity bounds for lifted inference. Theory and Practice of Logic Programming (to appear – available as arXiv:1204.3255v2)
Jaeger M, Van den Broeck G (2012) Liftability of probabilistic inference: Upper and lower bounds. In: Proceedings of the 2nd International Workshop on Statistical Relational AI (StaRAI-12)
Jaimovich A, Meshi O, Friedman N (2007) Template based inference in symmetric relational Markov random fields. In: Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI-07)
Jain D (2011) Knowledge engineering with Markov logic networks: A review. In: Proceedings of the Third Workshop on Dynamics of Knowledge and Belief (DKB-11)
Jensen D, Neville J, Gallagher B (2004) Why collective inference improves relational classification. In: Proceedings of the 10th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD-04)
Jha A, Gogate V, Meliou A, Suciu D (2010) Lifted inference seen from the other side: The tractable features. In: Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS-10)
Kautz H, Selman B, Jiang Y (1997) A general stochastic approach to solving problems with hard and soft constraints. In: Gu D, Du J, Pardalos P (eds) DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 35, American Mathematical Society, pp 573–586
Kautz H, Kersting K, Natarajan S, Poole D (eds) (2012) Statistical Relational AI Workshop at UAI-12
Kersting K (2012) Lifted probabilistic inference. In: Proceedings of the 20th European Conference on Artificial Intelligence (ECAI-12)
Kersting K, De Raedt L (2001) Towards combining inductive logic programming with Bayesian networks. In: Proceedings of the 11th International Conference on Inductive Logic Programming (ILP-01)
Kersting K, Ahmadi B, Natarajan S (2009) Counting belief propagation. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI-09)

Kersting K, Massaoudi YE, Hadiji F, Ahmadi B (2010a) Informed lifting for message-passing. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)

Kersting K, Russell S, Kaelbling LP, Halevy A, Natarajan S, Mihalkova L (eds) (2010b) Statistical Relational AI Workshop at AAAI-10

Kersting K, Poole D, Natarajan S (2011) Lifted inference in probabilistic logical models. Tutorial at IJCAI-11

Khosravi H, Schulte O, Man T, Xu X, Bina B (2010) Structure learning for Markov logic networks with many descriptive attributes. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)

Khosravi H, Schulte O, Hu J, Gao T (2012) Learning compact Markov logic networks with decision trees. Machine Learning 89(3):257–277

Khot T, Natarajan S, Kersting K, Shavlik JW (2011) Learning Markov logic networks via functional gradient boosting. In: Proceedings of the 11th IEEE International Conference on Data Mining (ICDM-11)

Khot T, Natarajan S, Kersting K, Shavlik J (2013) Learning relational probabilistic models from partially observed data - opening the closed-world assumption. In: Proceedings of the 23rd International Conference on Inductive Logic Programming (ILP-13)

Kiddon C, Domingos P (2011) Coarse-to-fine inference and learning for first-order probabilistic models. In: Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)

Kisyński J, Poole D (2009a) Constraint processing in lifted probabilistic inference. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI-09)

Kisyński J, Poole D (2009b) Lifted aggregation in directed first-order probabilistic models. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)

Kok S, Domingos P (2005) Learning the structure of Markov logic networks. In: Proceedings of the 22nd International Conference on Machine Learning (ICML-05)

Kok S, Domingos P (2009) Learning Markov logic network structure via hypergraph lifting. In: Proceedings of the 26th International Conference on Machine Learning (ICML-09)

Kok S, Domingos P (2010) Learning Markov logic networks using structural motifs. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10)

Koller D, Friedman N (2009) Probabilistic Graphical Models: Principles and Techniques. MIT Press

Koller D, Pfeffer A (1997) Object-oriented Bayesian networks. In: Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI-97)

Koller D, Pfeffer A (1998) Probabilistic frame-based systems. In: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)

Kschischang FR, Frey BJ, Loeliger HA (2001) Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory 47(2):498–519

Kuwadekar A, Neville J (2011) Relational active learning for joint collective classification models. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11)

Lafferty J, McCallum A, Pereira F (2001) Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the 18th International Conference on Machine Learning (ICML-01)

Lavrač N, Džeroski S (1993) Inductive Logic Programming: Techniques and Applications. Routledge

Lee S, Ganapathi V, Koller D (2006) Efficient structure learning of Markov networks using L_1-regularization. In: Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS-06)

Li WJ, Yeung DY, Zhang Z (2009) Probabilistic relational PCA. In: Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS-09)

London B, Getoor L (2013) Collective classification of network data. In: Aggarwal CC (ed) Data Classification: Algorithms and Applications, CRC Press

Lourenço HR, Martin OC, Stützle T (2003) Iterated local search. In: Glover FW, Kochenberger GA (eds) Handbook of Metaheuristics, Springer

Lowd D (2012) Closed-form learning of Markov networks from dependency networks. In: Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI-12)

Lowd D, Davis J (2010) Learning Markov network structure with decision trees. In: Proceedings of the 10th IEEE International Conference on Data Mining (ICDM-10)

Lowd D, Domingos P (2007) Efficient weight learning for Markov logic networks. In: Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-07)

Macskassy S, Provost F (2007) Classification in networked data: A toolkit and a univariate case study. Journal of Machine Learning Research 8:935–983

Maier M, Taylor B, Oktay H, Jensen D (2010) Learning causal models of relational domains. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)

Maier M, Marazopoulou K, Arbour D, Jensen D (2013) A sound and complete algorithm for learning causal models from relational data. In: Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI-13)

Mansinghka V, Roy D, Goodman N (eds) (2012) NIPS workshop on Probabilistic Programming: Foundations and Applications

McCallum A, Schultz K, Singh S (2009) FACTORIE: Probabilistic programming via imperatively defined factor graphs. In: Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS-09)

Mihalkova L, Mooney RJ (2007) Bottom-up learning of Markov logic network structure. In: Proceedings of the 24th International Conference on Machine Learning (ICML-07)

Mihalkova L, Mooney RJ (2009) Transfer learning from minimal target data by mapping across relational domains. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)

Mihalkova L, Richardson M (2009) Speeding up inference in statistical relational learning by clustering similar query literals. In: Proceedings of the 19th International Conference on Inductive Logic Programming (ILP-09)

Mihalkova L, Huynh T, Mooney RJ (2007) Mapping and revising Markov logic networks for transfer learning. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)

Milch B, Russell S (2006) General-purpose MCMC inference over relational structures. In: Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI-06)

Milch B, Marthi B, Russell S, Sontag D, Ong DL, Kolobov A (2005) BLOG: probabilistic models with unknown objects. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)

Milch B, Zettlemoyer LS, Kersting K, Haimes M, Kaelbling LP (2008) Lifted probabilistic inference with counting formulas. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)

Mladenov M, Ahmadi B, Kersting K (2012) Lifted linear programming. In: Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS-12)

Mooij J, Kappen B (2008) Bounds on marginal probability distributions. In: Proceedings of the 22nd Annual Conference on Neural Information Processing Systems (NIPS-08)

Muggleton S (1991) Inductive Logic Programming. New Generation Computing 8(4):295–318

Muggleton S (ed) (1992) Inductive Logic Programming. Academic Press

Muggleton S (1996) Stochastic logic programs. In: Proceedings of the 6th International Workshop on Inductive Logic Programming (ILP-96)

Muggleton S, De Raedt L (1994) Inductive logic programming: Theory and methods. Journal of Logic Programming 19/20:629–679

Munoz D, Bagnell J, Vandapel N, Hebert M (2009) Contextual classification with functional max-margin Markov networks. In: Proceedings of the 22nd IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR-09)

Murphy KP, Weiss Y, Jordan MI (1999) Loopy belief propagation for approximate inference: An empirical study. In: Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)

Natarajan S, Khot T, Lowd D, Kersting K, Tadepalli P, Shavlik J (2010) Exploiting causal independence in Markov logic networks: combining undirected and directed models. In: Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD-10)

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

Nath A, Domingos P (2010) Efficient lifting for online probabilistic inference. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)

Nath A, Richardson M (2012) Counting-MLNs: Learning relational structure for decision making. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-12)

Neville J, Jensen D (2007) Relational dependency networks. Journal of Machine Learning Research 8:653–692

Niepert M (2012) Markov chains on orbits of permutation groups. In: Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI-12)

Niepert M (2013) Symmetry-aware marginal density estimation. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI-13)

Noessner J, Niepert M, Stuckenschmidt H (2013) RockIt: Exploiting parallelism and symmetry for MAP inference in statistical relational models. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI-13)

Odersky M, Altherr P, Cremet V, Emir B, Maneth S, Micheloud S, Mihaylov N, Schinz M, Stenman E, Zenger M (2004) An overview of the Scala programming language. Tech. Rep. IC/2004/64, EPFL Lausanne, Switzerland

van Otterlo M (2009) The Logic of Adaptive Behavior - Knowledge Representation and Algorithms for Adaptive Sequential Decision Making under Uncertainty in First-Order and Relational Domains. IOS Press

Paes A, Revoredo K, Zaverucha G, Santos Costa V (2005) Probabilistic first-order theory revision from examples. In: Proceedings of the 15th International Conference on Inductive Logic Programming (ILP-05)

Pearl J (1988) Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann

Pearl J (2009) Causality: Models, Reasoning and Inference, 2nd edn. Cambridge University Press

Perlich C, Provost FJ (2003) Aggregation-based feature invention and relational concept classes. In: Proceedings of the 9th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD-03)

Pfeffer A (2007) Sampling with memoization. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)

Pfeffer A, Koller D, Milch B, Takusagawa KT (1999) SPOOK: A system for probabilistic object-oriented knowledge representation. In: Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)

Poole D (2003) First-order probabilistic inference. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)

Poole D, Zhang NL (2003) Exploiting contextual independence in probabilistic inference. Journal of Artificial Intelligence Research 18:263–313

Poon H, Domingos P (2006) Sound and efficient inference with probabilistic and deterministic dependencies. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)

Poon H, Domingos P, Sumner M (2008) A general method for reducing the complexity of relational inference and its application to MCMC. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)

Quinlan JR (1990) Learning logical definitions from relations. Machine Learning 5(3):239–266

Rabiner LR (1989) A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2)

Rattigan MJ, Maier ME, Jensen D (2011) Relational blocking for causal discovery. In: Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)

Richards BL, Mooney RJ (1992) Learning relations by pathfinding. In: Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)

Richards BL, Mooney RJ (1995) Automated refinement of first-order Horn-clause domain theories. Machine Learning 19(2):95–131

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

Riedel S (2008) Improving the accuracy and efficiency of MAP inference for Markov logic. In: Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI-08)

Roy D, Mansinghka V, Winn J, McAllester D, Tenenbaum J (eds) (2008) NIPS workshop on probabilistic programming: universal languages and inference; systems; and applications

Sarkhel S, Venugopal D, Singla P, Gogate V (2014) Lifted MAP inference for Markov Logic Networks. In: Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS-14)

Schulte O (2011) A tractable pseudo-likelihood function for Bayes nets applied to relational data. In: Proceedings of the Eleventh SIAM International Conference on Data Mining (SDM-11)

Schulte O, Khosravi H (2012) Learning graphical models for relational data via lattice search. Machine Learning 88(3):331–368

Schulte O, Khosravi H, Man T (2012) Learning directed relational models with recursive dependencies. Machine Learning 89(3):299–316

Sen P, Deshpande A, Getoor L (2008a) Exploiting shared correlations in probabilistic databases. In: Proceedings of the 34th International Conference on Very Large Data Bases (VLDB-08)

Sen P, Namata GM, Bilgic M, Getoor L, Gallagher B, Eliassi-Rad T (2008b) Collective classification in network data. AI Magazine 29(3):93–106

Sen P, Deshpande A, Getoor L (2009) Bisimulation-based approximate lifted inference. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI-09)

Shavlik J, Natarajan S (2009) Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)

Singla P, Domingos P (2005) Discriminative training of Markov logic networks. In: Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05)

Singla P, Domingos P (2006) Memory-efficient inference in relational domains. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)

Singla P, Domingos P (2008) Lifted first-order belief propagation. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)

Spirtes P, Glymour C, Scheines R (2001) Causation, Prediction, and Search. MIT Press

Srinivasan A (2001) The Aleph manual. `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`

Suciu D, Olteanu D, Ré C, Koch C (2011) Probabilistic Databases. Synthesis Lectures on Data Management, Morgan & Claypool

Taghipour N, Fierens D, Davis J, Blockeel H (2012) Lifted variable elimination with arbitrary constraints. In: Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS-12)

Taghipour N, Davis J, Blockeel H (2013a) First-order decomposition trees. In: Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS-13)

Taghipour N, Fierens D, Davis J, Blockeel H (2013b) Lifted variable elimination: Decoupling the operators from the constraint language. Journal of Artificial Intelligence Research 47:393–439

Taghipour N, Fierens D, Van den Broeck G, Davis J, Blockeel H (2013c) Completeness results for lifted variable elimination. In: Proceedings of the 16th International Conference on Artificial Intelligence and Statistics (AISTATS-13)

Taskar B (2004) Learning structured prediction models: A large margin approach. PhD thesis, Stanford University

Taskar B, Abbeel P, Koller D (2002) Discriminative probabilistic models for relational data. In: Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI-02)

Taskar B, Guestrin C, Koller D (2003) Max-margin Markov networks. In: Proceedings of the 17th Annual Conference on Neural Information Processing Systems (NIPS-03)

Taskar B, Chatalbashev V, Koller D (2004) Learning associative Markov networks. In: Proceedings of the 21st International Conference on Machine Learning (ICML-04)

Tierney L (1994) Markov chains for exploring posterior distributions. Annals of Statistics 22(4):1701–1728

Tsochantaridis I, Hofmann T, Joachims T, Altun Y (2004) Support vector machine learning for interdependent and structured output spaces. In: Proceedings of the 21st International Conference on Machine Learning (ICML-04)

Van den Broeck G (2011) On the completeness of first-order knowledge compilation for lifted probabilistic inference. In: Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS-11)

Van den Broeck G, Darwiche A (2013) On the complexity and approximation of binary evidence in lifted inference. In: Proceedings of the 27th Annual

Conference on Neural Information Processing Systems (NIPS-13)
Van den Broeck G, Davis J (2012) Conditioning in first-order knowledge compilation and lifted probabilistic inference. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-12)
Van den Broeck G, Taghipour N, Meert W, Davis J, De Raedt L (2011) Lifted probabilistic inference by first-order knowledge compilation. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)
Van den Broeck G, Choi A, Darwiche A (2012) 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-12)
Van den Broeck G, Meert W, Darwiche A (2014) Skolemization for weighted first-order model counting. In: Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR-14)
Venugopal D, Gogate V (2012) On lifting the Gibbs sampling algorithm. In: Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS-12)
Wainwright MJ, Jaakkola T, Willsky AS (2005) MAP estimation via agreement on trees: message-passing and linear programming. IEEE Transactions on Information Theory 51(11):3697–3717
Wang J, Domingos P (2008) Hybrid Markov logic networks. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)
Wei W, Erenrich J, Selman B (2004) Towards efficient sampling: Exploiting random walk strategies. In: Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)
Weiss D, Taskar B (2010) Structured prediction cascades. In: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS-10)
Wellman MP, Breese JS, Goldman RP (1992) From knowledge bases to decision models. Knowledge Engineering Review 7(1):35–53
Wu M, Schölkopf B (2007) Transductive classification via local learning regularization. In: Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS-07)
Xu Z, Kersting K, Tresp V (2009) Multi-relational learning with Gaussian processes. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)
Yanover C, Meltzer T, Weiss Y (2006) Linear programming relaxations and belief propagation - an empirical study. Journal of Machine Learning Research 7:1887–1907
Yedidia JS, Freeman WT, Weiss Y (2001) Understanding belief propagation and its generalizations. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)
Zhang NL, Poole D (1994) A simple approach to Bayesian network computations. In: Proccedings of the Tenth Canadian Artificial Intelligence Conference