Mach Learn (2013) 92:91–132
DOI 10.1007/s10994-013-5385-0

Exploiting symmetries for scaling loopy belief propagation and relational training

Babak Ahmadi · Kristian Kersting · Martin Mladenov · Sriraam Natarajan

Received: 18 November 2012 / Accepted: 9 May 2013 / Published online: 30 May 2013

Abstract Judging by the increasing impact of machine learning on large-scale data analysis in the last decade, one can anticipate a substantial growth in diversity of the machine learning applications for "big data" over the next decade. This exciting new opportunity, however, also raises many challenges. One of them is scaling inference within and training of graphical models. Typical ways to address this scaling issue are inference by approximate message passing, stochastic gradients, and MapReduce, among others. Often, we encounter inference and training problems with symmetries and redundancies in the graph structure. A prominent example are relational models that capture complexity. Exploiting these symmetries, however, has not been considered for scaling yet. In this paper, we show that inference and training can indeed benefit from exploiting symmetries. Specifically, we show that (loopy) belief propagation (BP) can be lifted. That is, a model is compressed by grouping nodes together that send and receive identical messages so that a modified BP running on the lifted graph yields the same marginals as BP on the original one, but often in a fraction of time. By establishing a link between lifting and radix sort, we show that lifting is MapReduce-

Editors: Tijl De Bie and Peter Flach.

B. Ahmadi (✉)
Knowledge Discovery Department, Fraunhofer IAIS, Sankt Augustin, Germany
e-mail: babak.ahmadi@iais.fraunhofer.de

K. Kersting · M. Mladenov
Institute for Geodesy and Geoinformation, University of Bonn, Bonn, Germany

K. Kersting
e-mail: kersting@igg.uni-bonn.de

M. Mladenov
e-mail: mladenov@igg.uni-bonn.de

K. Kersting
Fraunhofer IAIS, Bonn, Germany

S. Natarajan
Translational Science Institute, Wake Forest University, Winston Salem, USA
e-mail: snataraj@wfubmc.edu

able. Still, in many if not most situations training relational models will not benefit from this (scalable) lifting: symmetries within models easily break since variables become correlated by virtue of depending asymmetrically on evidence. An appealing idea for such situations is to train and recombine local models. This breaks long-range dependencies and allows to exploit lifting within and across the local training tasks. Moreover, it naturally paves the way for the first scalable lifted training approaches based on stochastic gradients, both in an online and a MapReduced fashion. On several datasets, the online training, for instance, converges to the same quality solution over an order of magnitude faster, simply because it starts optimizing long before having seen the entire mega-example even once.

Keywords Statistical relational learning · Lifted inference · Lifted online training · MapReduce

1 Introduction

Machine learning thrives on large datasets and models, and one can anticipate substantial growth in the diversity and the scale of impact of machine learning applications over the coming decade. Such datasets and models originate for example from social networks and media, online books at Google, image collections at Flickr, and robots entering the real life. And as storage capacity, computational power, and communication bandwidth continue to expand, today's "large" is certainly tomorrow's "medium" and next week's "small".

This exciting new opportunity, however, also raises many challenges. One of them is scaling inference within and training of graphical models. Statistical learning provides a rich toolbox for scaling. Actually, statistical learning is unthinkable for many practical applications without techniques such as inference by approximate message passing, stochastic gradients, and MapReduce, among others. Often, however, we face inference and training problems with symmetries and redundancies in the graph structure. A prominent example are relational models, see De Raedt et al. (2008), Getoor and Taskar (2007) for overviews, that tackle a long standing goal of AI, namely unifying first-order logic—capturing regularities and symmetries—and probability—capturing uncertainty. They often encode large, complex models using few weighted rules only and, hence, symmetries and redundancies abound. However, symmetries, stemming from relational models or not, have not been considered for scaling inference and training yet.

The situation is detailed in Table 1. For instance, (loopy) belief propagation (Pearl 1991) has been proven successful in many real-world applications. Gonzalez et al. (2009a, 2009b) have presented parallel versions for large factor graphs in shared memory as well as the distributed memory setting of computer clusters. Unfortunately,

Limitation 1 *Loopy belief propagation does not exploit symmetries.*

Indeed, for relational models, lifted belief propagation has been proposed that exploits symmetries, see e.g. Singla and Domingos (2008). It often renders large, previously intractable probabilistic inference problems quickly solvable by employing symmetries to handle whole sets of indistinguishable random variables. However, symmetries are present in abundance in traditional, non-relational models, too. Moreover, although with the availability of affordable commodity hardware and high performance networking, we have increasing access to computer clusters,

Table 1 Machine learning thrives on large-scale datasets and models. Several approaches have been developed to scale traditional statistical learning such as approximate message passing, stochastic gradients, and MapReduce, among others (denoted by "x" if it is main stream or by a representative citation). Scaling by lifting, i.e., exploiting symmetries within the graphical model structure, however, has not received a lot of attention. In this paper, we aim at closing this gap (denoted as "**Sect.**") in order to boost cross-fertilization. Please note that the references are naturally not exhaustive but representatives of the two worlds

		Statistical Learning			Statistical Relational Learning		
		single core	online	MapReduce	single core	online	MapReduce
Loopy BP	standard	x	Acar et al. (2008)	Gonzalez et al. (2009a)	x	de Salvo Braz et al. (2009)	Gonzalez et al. (2009a)
	lifted	**Sect. 4**	Hadiji et al. (2011)	**Sect. 5**	Singla and Domingos (2008), **Sect. 4**	Nath and Domingos (2010), Hadiji et al. (2011)	**Sect. 5**
Training	standard	x	x	Zinkevich et al. (2010)	x	Huynh and Mooney (2011)	**Sect. 6**
	lifted	–	–	–	**Sect. 6**	**Sect. 6**	**Sect. 7**

Limitation 2 *Lifted belief propagation is still carried out on a single core*.

That is, there are no inference approaches that exploit both symmetries and MapReduce for scaling. But even if so, in many if not most situations, training relational models will not benefit from scalable lifting.

Limitation 3 *Symmetries within models easily break since variables become correlated by virtue of depending asymmetrically on evidence*,

so that lifting produces models that are often not far from propositionalized, therefore canceling the benefits of lifting for training. And, in relational learning we typically face a single mega-example (Mihalkova et al. 2007) only, a single large set of inter-connected facts. Consequently, many if not all standard statistical learning methods do not naturally carry over to the relational case. Consider, for example, stochastic gradient methods. Similar to the perceptron method (Rosenblatt 1962), stochastic gradient approaches update the weight vector in an online setting. We essentially assume that the training examples are given one at a time. The algorithms examine the current training example and then update the parameter vector accordingly. Stochastic gradient approaches often scale sub-linearly with the amount of training data, making them very attractive for large training data as targeted by statistical relational learning. Empirically, they are even often found to be more resilient to errors made when approximating the gradient. Unfortunately, stochastic gradient methods do not naturally carry over to the relational cases:

Limitation 4 *Stochastic gradients coincide with batch gradients in the relational case since there is only a single mega-example*.

Consequently, while there are efficient parallelized gradient approaches such as developed by Zinkevich et al. (2010) that impose very little I/O overhead and are well suited for a MapReduce implementation, these have not been used for lifted training.

In this paper, we demonstrate how to overcome all four limitations, that is we fill in most gaps in Table 1, and thereby move statistical and statistical relational learning closer together in order to boost cross-fertilization. Specifically, we make the following contributions:

- We introduce lifted loopy belief propagation that exploits symmetries and hence often scales much better than standard loopy belief propagation. Its underlying idea is rather simple: group together nodes that are indistinguishable in terms of messages received and sent given the evidence. The lifted graph is often significantly smaller and can be used to perform a modified loopy belief propagation yielding the same results as loopy belief propagation applied to the unlifted graph. This overcomes Limitation 1, and our experimental results show that considerable efficiency gains are obtainable.
- We present the first MapReduce lifted belief propagation approach. More precisely, we establish a link between color-passing, the specific way of lifting the graph, and radix sort, which is well-known to be MapReduce-able. Together with Gonzalez et al. (2009a, 2009b) MapReduce belief propagation approach, this overcomes Limitation 2. Our experimental results show that MapReduced lifting scales much better then single-core lifting.
- We develop the first lifted training approach. More specifically, we shatter a model into local pieces. In each iteration, we then train the pieces independently and re-combine the learned parameters from each piece. This breaks long-range dependencies and allows one to exploit lifting across the local training tasks. Hence, it overcomes Limitation 3.

– Based on the lifted piece-wise training, we introduce the first online training approach for relational models that can deal with partially observed training data. The idea is, we treat (mini-batches of) pieces as training examples and process one piece after the other. This overcomes Limitation 4. Our experimental results on several datasets demonstrate that the online training converges to the same quality solution over an order of magnitude faster than batch learning, simply because it starts optimizing long before having seen the entire mega-example even once. Moreover, it naturally paves the way to MapReduced lifted training. Indeed, the way we shatter the full model into pieces greatly effects the learning quality: important influences between variables might get broken. To overcome this, we randomly grow relational piece patterns that form trees. Our experimental results show that *tree pieces* can balance lifting and quality of the online training.

All experimental results on several datasets together demonstrate that scaling inference and training can greatly benefit from symmetries.

The present paper is a significant extension of the ECML/PKDD 2012 and the UAI 2009 conference papers (Ahmadi et al. 2012; Kersting et al. 2009). It provides the first coherent view on lifted inference and training using loopy belief propagation suggesting (piecewise) symmetries as a novel but promising dimension for scaling statistical machine learning. It develops the first MapReduce approaches to both tasks by establishing a link between color-passing and radix sort, which is well-known to be MapReduce-able. Additionally, the present paper provides a detailed description of the color-passing procedure and the resulting lifted equations for lifted BP as well as a characterization of the lifting in terms of colored computation trees (CCTs). Finally, and probably most importantly, it experimentally proves that exploiting symmetries and MapReduce together can scale much better than exploiting one of them alone.

We proceed as follows. After touching upon related work, we recap factor graphs, loopy belief propagation and Markov logic networks, the probabilistic relational framework we focus on for illustration purposes only. We then show how to scale message-passing inference in two dimensions, namely by exploiting symmetries and by MapReduce. Then, we develop stochastic relational gradients that naturally paves the way to MapReduce training. Each section is concluded by presenting the respective experimental results.

2 Related work

The paper aims at getting mainstream statistical and statistical relational learning closer together. As argued above, we do so by employing symmetries, MapReduce and stochastic gradients. Consequently, there are several related lines of research.

Lifted probabilistic inference Employing symmetries in graphical models for speeding up probabilistic inference, called lifted probabilistic inference, has recently received a lot of attention, see e.g. Kersting (2012) for an overview. The closest work to our approach for exploiting symmetries within message passing is the work by Singla and Domingos (2008). Actually, an investigation of their approach was the seed that grew into our proposal. Singla and Domingos' *lifted first-order belief propagation* (LFOBP) builds upon Jaimovich et al. (2007) and also groups random variables, i.e., nodes that send and receive identical messages. Lifted BP, the approach introduced in the present paper, differs from LFOBP on two important counts. First, lifted BP is conceptually easier than LFOBP. This is because efficient inference approaches for first-order and relational probabilistic models are typically rather complex. Second, LFOBP requires as input the specification of the probabilistic

model in first-order logical format. Only nodes over the same predicate can be grouped together to form so-called clusternodes. That means LFOBP coincides with standard BP for propositional MLNs, i.e., MLNs involving propositional variables only. The reason is that propositions are predicates with arity 0 so that the clusternodes are singletons. Hence, no nodes and no features are grouped together. In contrast, our lifting can directly be applied to any factor graph over finite random variables. In this sense, lifted BP is a generalization of LFOBP.

Sen et al. (2008) presented another "clustered" inference approach based on bisimulation. Like lifted BP, which can also be viewed as running a bisimulation-like algorithm on the factor graph, Sen et al.'s approach also does not require a first-order logical specification. In contrast to lifted BP, it is guaranteed to converge but is also much more complex. Its efficiency in dynamic relational domains, in which variables easily become correlated over time by virtue of sharing common influences in the past, is unclear and its evaluation is an interesting future work.

Others such as Poole (2003), Braz et al. (2005, 2006), Milch et al. (2008), Kisynski and Poole (2009), Choi et al. (2011) and Taghipour et al. (2012) have developed lifted versions of the variable elimination (VE) algorithm. They typically also employ a counting elimination operator that is equivalent to counting indistinguishable random variables and then summing them out immediately. The different variations of these algorithms improve upon Poole (2003) and Braz et al. (2005, 2006) work by introducing counting formulas (Milch et al. 2008), aggregation operators (Choi et al. 2011; Kisynski and Poole 2009) and handling arbitrary constraints (Taghipour et al. 2012). Choi et al. (2010) developed variable elimination algorithm when the underlying distributions are continuous random variables. These exact inference approaches are extremely complex, so far do not easily scale to realistic domains, and hence have only been applied to rather small artificial problems. Recently, Van den Broeck et al. (2012) proposed a method that relaxes first-order conditions to perform exact lifted inference on the model and then incrementally improve the approximation by adding more constraints back into the model. This can be viewed as bridging the lifted VE and the lifted BP methods presented above. Again, as for LFOBP, a first-order logical specification of the model is required.

An alternative to BP and VE is to use search-based methods based on recursive conditioning. That is, we decompose by conditioning on parameterized variables a lifted network into smaller networks that can be solved independently. Each of these subnetworks is then solved recursively using the same method, until we reach a simple enough network that can be solved (Darwiche 2001). Recently, several lifted search-based methods have been proposed (Gogate and Domingos 2010, 2011; Van den Broeck et al. 2011; Poole et al. 2011) that assume a relational model given. Gogate and Domingos (2011) reduced the problem of lifted probabilistic inference to weighted model counting in a lifted graph. Van den Broeck et al. (2011) employ circuits in first-order deterministic decomposable negation normal form to do the same, also for higher order marginals (Van den Broeck and Davis 2012). Both these approaches were developed in parallel and have promising potential to lifted inference. There are also sampling methods that employ ideas of lifting. Milch and Russell developed an MCMC approach where states are only partial descriptions of possible worlds (Milch and Russell 2006). Zettlemoyer et al. (2007) extended particle filters to a logical setting. Gogate and Domingos introduced a lifted importance sampling (Gogate and Domingos 2011). Recently, Niepert proposed permutation groups and group theoretical algorithms to represent and manipulate symmetries in probabilistic models, which can be used for MCMC (Niepert 2012). And Bui et al. (2012) have shown that for MAP inference we can exploit the symmetries of the model before evidence is obtained. It is an interesting question whether one can

characterize these symmetries more precisely. Work on symmetry in exponential families (Bui et al. 2012) shows that one can create clusternodes using the automorphism group of the graph, using the notion of orbit partitions. Mladenov et al. (2012) explored color-passing in the setting of linear programming and showed that symmetries can be found and exploited in linear programs.

As of today, none of these approaches have been shown to be MapReduce-able. Moreover, many of them have exponential costs in the treewidth of the graph, making them infeasible for most real-world applications, and none of them have been used for training relational models.

Local training Our lifted training approach is related to local training methods well known for propositional graphical models. Besag (1975) presented a pseudolikelihood (PL) approach for training an Ising model with a rectangular array of variables. PL, however, tends to introduce a bias and is not necessarily a good approximation of the true likelihood with a smaller number of samples. In the limit, however, the maximum pseudolikelihood coincides with that of the true likelihood (Winkler 1995). Hence, it is a very popular method for training models such as Conditional Random Fields (CRF) where the normalization can become intractable while PL requires normalizing over only one node. An alternative approach is to decompose the factor graph into tractable subgraphs (or pieces) that are trained independently (Sutton and Mccallum 2009), that the present paper also follows. This *piecewise training* can be understood as approximating the exact likelihood using a propagation algorithm such as BP. Sutton and Mccallum (2009) also combined the two ideas of PL and piecewise training to propose piecewise pseudolikelihood (PWPL) which in spite of being a double approximation has the benefit of being accurate like piecewise and scales well due to the use of PL. Another intuitive approach is to compute approximate marginal distributions using a global propagation algorithm like BP, and simply substitute the resulting beliefs into the exact ML gradient (Lee et al. 2007), which will result in approximate partial derivatives. Similarly, the beliefs can also be used by a sampling method such as MCMC where the true marginals are approximated by running an MCMC algorithm for a few iterations. Such an approach is called contrastive divergence (Hinton 2002) and is popular for training CRFs.

Statistical relational learning All the above training methods were originally developed for propositional data while real-world data is inherently noisy and relational. Statistical Relational Learning (SRL) (De Raedt et al. 2008; Getoor and Taskar 2007) deals with uncertainty and relations among objects. The advantage of relational models is that they can succinctly represent probabilistic dependencies among the attributes of different related objects leading to a compact representation of learned models. While relational models are very expressive, learning them is a computationally intensive task. Recently, there have been some advances in learning SRL models, especially in the case of Markov Logic Networks (Khot et al. 2011; Kok and Domingos 2009, 2010; Lowd and Domingos 2007). Algorithms based on functional-gradient boosting (Friedman 2001) have been developed for learning SRL models such as Relational Dependency Networks (Natarajan et al. 2012), and Markov Logic Networks (Khot et al. 2011). Piecewise learning has also been pursued already in SRL. For instance, the work by Richardson and Domingos (2006) used pseudolikelihood to approximate the joint distribution of MLNs which is inspired from the local training methods mentioned above. Though all these methods exhibit good empirical performance, they apply the closed-world assumption, i.e., whatever is unobserved in the world is considered to be false. They cannot easily deal with missing information. To do so, algorithms based on classical EM (Dempster et al. 1977) have

been developed for ProbLog, CP-logic, PRISM, probabilistic relational models, Bayesian logic programs (Getoor et al. 2002; Gutmann et al. 2011; Kersting and De Raedt 2001; Sato and Kameya 2001; Thon et al. 2011), among others, as well as gradient-based approaches for relational models with complex combining rules (Natarajan et al. 2009; Jaeger 2007). Poon and Domingos (2008) extended the approach of Lowd and Domingos (2007) which is using a scaled conjugate gradient with preconditioner to handle missing data. All these approaches, however, assume a *batch* learning setting; they do not update the parameters until the entire data has been scanned. In the presence of large amounts of data such as relational data, the above method can be wasteful. Stochastic gradient methods as considered in the present paper, on the other hand, are online and scale sub-linearly with the amount of training data, making them very attractive for large data sets. Only Huynh and Mooney (2011) have recently studied online training of MLNs. Here, training was posed as an online max margin optimization problem and a gradient for the dual was derived and solved using incremental-dual-ascent algorithms. Huynh and Mooney's approach, however, is orthogonal to our approach in that they do discriminative learning as opposed to generative learning in the current paper. Also, they do not employ lifted inference for training and make the closed-world assumption. It would be interesting to see where the approaches can complement each other, e.g. by employing parallel lifted max-product belief propagation for the max margin computation.

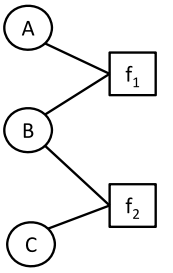
Distributed inference and training To scale probabilistic inference and training, Gonzalez et al. (2009a, 2009b) present algorithms for parallel inference on large factor graphs using belief propagation in shared memory as well as the distributed memory setting of computer clusters. Although, Gonzalez et al. report on map-reducing lifted inference within MLNs, they actually assume the lifted network to be given. We here demonstrate for the first time that lifting per se is MapReduce-able, thus putting scaling lifted SRL to "Big Data" within reach. As our experimental results illustrate, we achieve orders of magnitude improvement over existing methods using our approach. As far as we are aware, the only other method that has addressed scaling up of SRL algorithms is the work by Niu et al. (2011) that considered the problem of scaling up ground inference and learning over factor graphs that are multiple terabytes in size. They achieve these using database technology with two key observations: First, grounding of the entire SRL model into a factor graph is seen as a RDBMS join that is realized using distributed RDBMS. Second, they make learning I/O bound by using a storage manager to run inference efficiently over factor graphs that are larger than main memory. It remains a very interesting and exciting future work to implement our algorithm using this database technology.

3 Loopy belief propagation and Markov logic networks

Let $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ be a set of n discrete-valued random variables and let x_i represent the possible realizations of random variable X_i. Graphical models compactly represent a joint distribution over $\mathbf{X}$ as a product of factors (Pearl 1991), i.e.,

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_k f_k(\mathbf{x}_k). \tag{1}$$

Here, each factor f_k is a non-negative function of a subset of the variables $\mathbf{x}_k$, and Z is a normalization constant. As long as $P(\mathbf{X} = \mathbf{x}) > 0$ for all joint configurations $\mathbf{x}$, the distribution



A	B	f_1
True	True	1.2
True	False	1.4
False	True	2.0
False	False	0.4

C	B	f_2
True	True	1.2
True	False	1.4
False	True	2.0
False	False	0.4

Fig. 1 An example for a factor graph with associated potentials. *Circles* denote variables (binary in this case), *squares* denote factors

can be equivalently represented as a log-linear model:

$$P(\mathbf{X}=\mathbf{x}) = \frac{1}{Z}\exp\left[\sum_i w_i \cdot \varphi_i(\mathbf{x})\right], \tag{2}$$

where the features $\varphi_i(x)$ are arbitrary functions of (a subset of) the configuration $\mathbf{x}$.

Graphical models can be represented as factor graphs. A factor graph, as shown in Fig. 1, is a bipartite graph that expresses the factorization structure in Eq. (1). It has a variable node (denoted as a circle) for each variable X_i, a factor node (denoted as a square) for each f_k, with an edge connecting variable node i to factor node k if and only if X_i is an argument of f_k. We will consider one factor $f_i(\mathbf{x}) = \exp[w_i \cdot \varphi_i(\mathbf{x})]$ per feature $\varphi_i(\mathbf{x})$, i.e., we will not aggregate factors over the same variables into a single factor.

An important inference task is to compute the conditional probability of variables given the values of some others, the evidence, by summing out the remaining variables. The belief propagation (BP) algorithm is an efficient way to solve this problem that is exact when the factor graph is a tree, but only approximate when the factor graph has cycles. One should note that the problem of computing marginal probability functions is in general hard (#P-complete).

Belief propagation makes local computations only. It makes use of the graphical structure such that the marginals can be computed much more efficiently. We will now describe the BP algorithm in terms of operations on a factor graph. The computed marginal probability functions will be exact if the factor graph has no cycles, but the BP algorithm is still well-defined when the factor graph does have cycles. Although this loopy belief propagation has no guarantees of convergence or of giving the correct result, in practice it often does, and can be much more efficient than other methods (Murphy et al. 1999).

To define the BP algorithm, we first introduce messages between variable nodes and their neighboring factor nodes and vice versa. The message from a variable X to a factor f is

$$\mu_{X\to f}(x) = \prod_{h\in \mathrm{nb}(X)\setminus\{f\}} \mu_{h\to X}(x) \tag{3}$$

where $\mathrm{nb}(X)$ is the set of factors X appears in. The message from a factor to a variable is

$$\mu_{f\to X}(x) = \sum_{\neg\{X\}}\left(f(\mathbf{x}) \prod_{Y\in \mathrm{nb}(f)\setminus\{X\}} \mu_{Y\to f}(y)\right) \tag{4}$$

where $\mathrm{nb}(f)$ are the arguments of f, and the sum is over all the values of these except X, denoted as $\neg\{X\}$. The messages are usually initialized to 1.

Table 2 (*Top*) Example of Markov logic network inspired by Singla and Domingos (2008). Free variables are implicitly universally quantified. (*Center*) Grounding of the MLN for constants *Anna* and *Bob*. (*Bottom*) Additional clause about similar smoking habits of buddies instead of friends

English	First-Order Logic	Weight
Smoking causes cancer	`Smokes(x)` $\Rightarrow$ `Cancer(x)`	1.5
Friends have similar smoking habits	`Friends(x, y)` $\Rightarrow$ (`Smokes(x)` $\Leftrightarrow$ `Smokes(y)`)	2.0

1.5 : $Smokes(Anna) \Rightarrow Cancer(Anna)$

1.5 : $Smokes(Bob) \Rightarrow Cancer(Bob)$

2.0 : $Friends(Anna, Bob) \Rightarrow (Smokes(Anna) \Leftrightarrow Smokes(Bob))$

2.0 : $Friends(Bob, Anna) \Rightarrow (Smokes(Bob) \Leftrightarrow Smokes(Anna))$

English	First-Order Logic	Weight
Buddies have similar smoking habits	`Buddies(x, y)` $\Rightarrow$ (`Smokes(x)` $\Leftrightarrow$ `Smokes(y)`)	2.0

Now, the unnormalized belief of each variable X_i can be computed from the equation

$$b_i(x_i) = \prod_{f \in \mathrm{nb}(X_i)} \mu_{f \to X_i}(x_i) \tag{5}$$

Evidence is incorporated by setting $f(\mathbf{x}) = 0$ for states $\mathbf{x}$ that are incompatible with it. Different schedules may be used for message-passing.

Since, loopy belief propagation is efficient, it can directly be used for inference within statistical relational models that have recently gained attraction within the machine learning and AI communities. Statistical relational models provide powerful formalisms to compactly represent complex real-world domains. These formalisms enable us to effectively represent and tackle large problem instances for which inference and training is increasingly challenging. One of the most prominent examples of statistical relational models are Markov logic networks (Richardson and Domingos 2006).

A Markov logic network (MLN) F is defined by a set of first-order formulas (or clauses) F_i with associated weights w_i, $i \in \{1, \ldots, m\}$. Together with a set of constants $C = \{C_1, C_2, \ldots, C_n\}$ F can be grounded, i.e. the free variables in the predicates of the formulas F_i are bound to be constants in C, to define a Markov network. This ground Markov network contains a binary node for each possible grounding of each predicate, and a feature for each grounding f_k of each formula. The joint probability distribution of an MLN is given by

$$P(X = x) = Z^{-1} \exp\left(\sum_{i=1}^{|F|} \theta_i n_i(x) \right) \tag{6}$$

where for a given possible world x, i.e. an assignment of all variables X, $n_i(x)$ is the number groundings of the ith formula that evaluate to *true* and Z is a normalization constant.

An example of a simple Markov Logic network is depicted in Table 2 (top). It consist of two first order clauses with associated weights. Together with a set of constants, say *Anna* and *Bob*, it can be grounded to the ground clauses depicted in Table 2 (center). This set of ground clauses now defines a Markov network and in turn a factor graph with binary variables and factors. Each ground atom, e.g. *Smokes*(*Anna*), is represented by a variable

node in the factor graph and each ground clause is represented by a factor node. There is an edge between a variable and a factor iff the ground atoms appears in the corresponding ground clause and $f_i(x_{\{i\}}) = e^{\theta_i n_i(x)}$, where $x_{\{i\}}$ are the truth values of the variables appearing in f_i, w_i the weights of the clause and $n_i(x)$ is the truth value of the clause, given the assignment x. For example, $f_1(Smokes(Anna) = True, Cancer(Anna) = False) = e^{1.5*0} = 1$. For more details we refer to Richardson and Domingos (2006).

4 Scaling up inference: lifted belief propagation

Although already quite efficient, many graphical models produce inference problems with a lot of additional regularities reflected in the graphical structure but not exploited by BP. Probabilistic graphical models such as MLNs are prominent examples. As an illustrative example, reconsider the factor graph in Fig. 1. The associated potentials are identical. In other words, although the factors involved are different on the surface, they actually share quite a lot of information. Standard BP cannot make use of this information. In contrast, *lifted* BP—which we will introduce now—can make use of it and speed up inference by orders of magnitude.

Lifted BP performs two steps: Given a factor graph G, it first computes a compressed factor graph $\mathfrak{G}$ and then runs a modified BP on $\mathfrak{G}$. We will now discuss each step in turn using fraktur letters such as $\mathfrak{G}$, $\mathfrak{X}$, and $\mathfrak{f}$ to denote compressed graphs, nodes, and factors.

Step 1—compressing the factor graph Essentially, we simulate BP keeping track of which nodes and factors send the same messages, and group nodes and factors together correspondingly.

Let G be a given factor graph with Boolean variable and factor nodes. Initially, all variable nodes fall into three groups (one or two of these may be empty), namely known true, known false, and unknown. For ease of explanation, we will represent the groups by colored/shaded circles, say, magenta/white, green/gray, and red/black. All factor nodes with the same associated potentials also fall into one group represented by colored/shaded squares. For the factor graph in Fig. 1 the situation is depicted in Fig. 2. As shown on the left-hand side, assuming no evidence, all variable nodes are unknown, i.e., red/black. Now, each variable node sends a message to its neighboring factor nodes saying "I am of color/shade red/black". A factor node sorts the incoming colors/shades into a vector according to the order of the variables in its arguments. The last entry of the vector is the factor node's own color/shade, represented as light blue/gray square in Fig. 2. This color/shade signature is sent back to the neighboring variables nodes, essentially saying "I have communicated with these

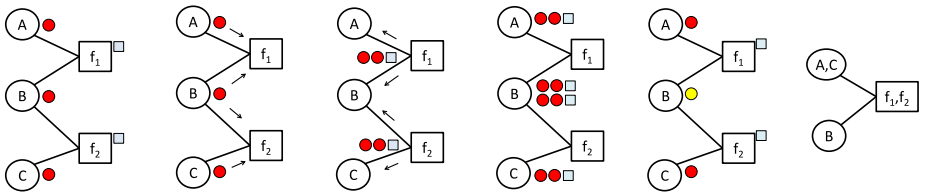


Fig. 2 From *left* to *right*, the steps of CFG compressing the factor graph in Fig. 1 assuming no evidence. The *shaded/colored small circles* and *squares* denote the groups and signatures produced running CFG. On the *right-hand* side, the resulting compressed factor graph is shown. For details we refer to Sect. 4 (Color figure online)

nodes". To lower communication cost, identical signatures can be replaced by new colors. The variable nodes stack the incoming signatures together and, hence, form unique signatures of their one-step message history. Variable nodes with the same stacked signatures, i.e., message history can now be grouped together. To indicate this, we assign a new color/shade to each group. In our running example, only variable node B changes its color/shade from red/black to yellow/gray. The factors are grouped in a similar fashion based on the incoming color/shade signatures of neighboring nodes. Finally, we iterate the process. As the effect of the evidence propagates through the factor graph, more groups are created. The process stops when no new colors/shades are created anymore.

The final compressed factor graph $\mathfrak{G}$ is constructed by grouping all nodes with the same color/shade into so-called *clusternodes* and all factors with the same color/shade signatures into so-called *clusterfactors*. In our case, variable nodes A, C and factor nodes f_1, f_2 are grouped together, see the right hand side of Fig. 2. Clusternodes (resp. clusterfactors) are sets of nodes (resp. factors) that send and receive the same messages at each step of carrying out BP on G. It is clear that they form a partition of the nodes in G.

Algorithm 1 summarizes our approach for computing the compressed factor graph $\mathfrak{G}$ Note that, we have to keep track of the position a variable appeared in a factor. Since factors in general are not commutative, it matters where the node appeared in the factor. Reconsider our example from Fig. 1, where $f_1(A = True, B = False) \neq f_1(A = False, B = True)$. However, in cases where the position of the variables within the factor does not matter, one can gain additional lifting by neglecting the variable positions in the node signatures and by sorting the colors within the factors' signatures. The ordering of the factors in the node signatures, on the other hand, should always be neglected, thus we perform a sort of the node color signatures (Algorithm 1, line 15).

The clustering we do here, groups together nodes and factors that are indistinguishable given the belief propagation computations. To better understand the color-passing and the resulting grouping of the nodes, it is useful to think of BP and its operations in terms of its *computation tree* (CT), see e.g. Ihler et al. (2005). The CT is the unrolling of the (loopy) graph structure where each level i corresponds to the i-th iteration of message passing. Similarly we can view color-passing, i.e., the lifting procedure as a *colored computation tree* (CCT). More precisely, one considers for every node X the computation tree rooted in X but now each node in the tree is colored according to the nodes' initial colors, cf. Fig. 3(a). For simplicity edge colors are omitted and we assume that the potentials are the same on all edges. Each CCT encodes the root nodes' local communication patterns that show all the colored paths along which node X communicates in the network. Consequently, color-passing groups nodes with respect to their CCTs: nodes having the same set of rooted paths of colors (node and factor names neglected) are clustered together. For instance, Fig. 3(b) shows the CCTs for the nodes X_1 to X_4. Because their set of paths are the same, X_1 and X_2 are grouped into one clusternode, X_3 and X_4 into another.[1]

Now we can run BP with minor modifications on the compressed factor graph $\mathfrak{G}$.

Step 2—BP on the compressed factor graph Recall that the basic idea is to simulate BP carried out on G on $\mathfrak{G}$. An edge from a clusterfactor $\mathfrak{f}$ to a cluster node $\mathfrak{X}$ in $\mathfrak{G}$ essentially represents multiple edges in G. Let $c(\mathfrak{f}, \mathfrak{X}, p)$ be the number of identical messages that would be sent from the factors in the clusterfactor $\mathfrak{f}$ to each node in the clusternode $\mathfrak{X}$ that

[1]The partitioning of the nodes obtained by color-passing corresponds to the so-called coarsest equitable partition of the graph (Mladenov et al. 2012). However, a formal characterization of the symmetries is beyond the scope of the current paper.

Algorithm 1: CFG—CompressFactorGraph

Data: A factor Graph G with variable nodes X and factors f, Evidence E
Result: Compressed Graph $\mathfrak{G}$ with clustervariable nodes $\mathfrak{X}$ and clusterfactor nodes $\mathfrak{f}$

Compute initial clusters of the X_is w.r.t. E;
repeat
  `// Form color signature for each factor`
  foreach *factor* f_k **do**
    $signature_{f_k} = [\,]$;
    foreach *node* $X_i \in nb(f_k)$ **do in order of appearance in** f_k
      $signature_{f_k}$.append($X_i.color$);
    $signature_{f_k}$.append($f_k.color$);
  Group together all f_ks having the same signature;
  Assign each such cluster a unique color;
  Set $f_k.color$ correspondingly for all f_ks;
  `// Form color signature for each variable`
  foreach *node* $X_i \in X, i = 1, \ldots, n$ **do**
    $signature_{X_i} = [\,]$;
    foreach *factor* $f_k \in nb(X_i)$ **do**
      $signature_{X_i}$.append(($f_k.color$, $p(X_i, f_k)$));
    sort $signature_{X_i}$ according to ordering given by *color*;
    $signature_{X_i}$.append($X_i.color$);
  Group together all X_is having the same signature;
  Assign each such cluster a unique color;
  Set $X_i.color$ correspondingly for all X_is;
until *grouping does not change*;

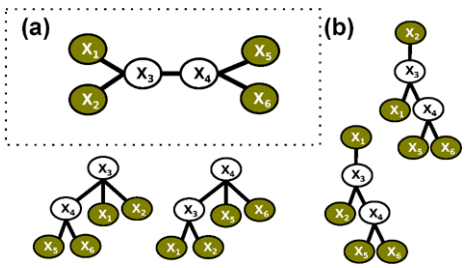


Fig. 3 (**a**) Original factor graph with *colored* nodes. (**b**) *Colored* computation trees for nodes X_1 to X_4. As one can see, nodes X_1 and X_2, respectively X_3 and X_4, have the same colored computation tree, thus are grouped together during color-passing (Color figure online)

appears at position p in $\mathfrak{f}$ if BP was carried out on G. The message from a clustervariable $\mathfrak{X}$ to a clusterfactor $\mathfrak{f}$ at position p is

$$\mu_{\mathfrak{X}\to\mathfrak{f},p}(x) = \mu_{\mathfrak{f},p\to\mathfrak{X}}(x)^{c(\mathfrak{f},\mathfrak{X},p)-1} \prod_{\mathfrak{h}\in \mathrm{nb}(\mathfrak{X})} \prod_{\substack{q\in P(\mathfrak{h},\mathfrak{X}) \\ (\mathfrak{h},q)\neq(\mathfrak{f},p)}} \mu_{\mathfrak{h},q\to\mathfrak{X}}(x)^{c(\mathfrak{h},\mathfrak{X},q)}, \tag{7}$$

where $\mathrm{nb}(\mathfrak{X})$ now denotes the neighbor relation in the compressed factor graph $\mathfrak{G}$ and $P(\mathfrak{h},\mathfrak{X})$ denotes the positions nodes from $\mathfrak{X}$ appear in $\mathfrak{f}$. The $c(\mathfrak{f},\mathfrak{X},p)-1$ exponent reflects the fact that a clustervariable's message to a clusterfactor excludes the corresponding factor's message to the variable if BP was carried out on G. The message from the factors

to neighboring variables essentially remains unchanged. The difference is that we now only send one message per clusternode and position, given by

$$\mu_{\mathfrak{f},p\to\mathfrak{X}}(x)=\sum_{\neg\{\mathfrak{X}\}}\left(\mathfrak{f}(\mathbf{x})\prod_{\mathfrak{Y}\in\mathrm{nb}(\mathfrak{f})}\prod_{q\in P(\mathfrak{f},\mathfrak{Y})}\mu_{\mathfrak{Y}\to\mathfrak{f},q}(y)^{c(\mathfrak{f},\mathfrak{Y},q)-\delta_{\mathfrak{X}\mathfrak{Y}}\delta_{pq}}\right), \tag{8}$$

where $\delta_{\mathfrak{X}\mathfrak{Y}}$ and δ_{pq} are one iff $\mathfrak{X}=\mathfrak{Y}$ and $p=q$ respectively. The unnormalized belief of $\mathfrak{X}_i$, i.e., of any node X in $\mathfrak{X}_i$ can be computed from the equation

$$b_i(x_i)=\prod_{\mathfrak{f}\in\mathrm{nb}(\mathfrak{X}_i)}\prod_{p\in P(\mathfrak{f},\mathfrak{X}_i)}\mu_{\mathfrak{f},p\to\mathfrak{X}_i}(x_i)^{c(\mathfrak{f},\mathfrak{X}_i,p)}. \tag{9}$$

Evidence is incorporated either on the ground level by setting $f(\mathbf{x})=0$ or on the lifted level by setting $\mathfrak{f}(\mathbf{x})=0$ for states $\mathbf{x}$ that are incompatible with it.[2] Again, different schedules may be used for message-passing. If there is no compression possible in the factor graph, i.e. there are no symmetries to exploit, there will be only a single position for a variable $\mathfrak{X}$ in factor $\mathfrak{f}$ and the counts $c(\mathfrak{f},\mathfrak{X},1)$ will be 1. In this case the equations simplify to Eqs. (3)–(5).

To conclude the section, the following theorem states the correctness of *lifted* BP.

Theorem 1 *Given a factor graph G, there exists a unique minimal compressed $\mathfrak{G}$ factor graph, and algorithm* CFG(G) *returns it. Running BP on $\mathfrak{G}$ using Eqs.* (7) *and* (9) *produces the same results as BP applied to G.*

The theorem generalizes the theorem of Singla and Domingos (2008) but can essentially be proven along the same ways. Although very similar in spirit, lifted BP has one important advantage: not only can it be applied to first-order and relational probabilistic models, but also directly to traditional, i.e., propositional models such as Markov networks.

Proof We prove the uniqueness of $\mathfrak{G}$ by contradiction. Suppose there are two minimal lifted networks $\mathfrak{G}_1$ and $\mathfrak{G}_2$. Then there exists a variable node X that is in clusternode $\mathfrak{X}_1$ in $\mathfrak{G}_1$ and in clusternode $\mathfrak{X}_2$ in $\mathfrak{G}_2$, $\mathfrak{X}_1\neq\mathfrak{X}_2$; or similarly for some clusterfactor $\mathfrak{f}$. Since all nodes in $\mathfrak{X}_1$, and $\mathfrak{X}_2$ respectively, send and receive the same messages $\mathfrak{X}_1=\mathfrak{X}_2$. Following the definition of clusternodes, any pair of nodes $\mathfrak{X}$ and $\mathfrak{Y}$ in $\mathfrak{G}$ send and receive different messages, therefore no further grouping is possible. Hence, $\mathfrak{G}$ is a unique minimal compressed network.

Now we show that algorithm CFG(G) returns this minimal compressed network. The following arguments are made for the variable nodes in the graph, but can analogously be applied to factor nodes. Reconsider the colored computation trees (CCT) which resemble the paths along which each node communicates in the network. Variables nodes are being grouped if they send and receive the same messages. Thus nodes X_1 and X_2 are in they same clusternode iff they have the same colored computation tree. Unfolding the computation tree to depth k gives the exact messages that the root node receives after k BP iterations. CFG(G) finds exactly the similar CCTs. Initially all nodes are colored by the evidence we have, thus for iteration $k=0$ we group all nodes that are similarly colored at the level k in the CCT. The signatures at iteration $k+1$ consist of the signatures at depth k (the nodes own color in the previous iteration) and the colors of all direct neighbors. That is, at iteration $k+1$ all

[2]Note that, the variables have been grouped according to evidence and their local structure. Thus all factors within a clusterfactor are indistinguishable and we can set the states of the whole clusterfactor $\mathfrak{f}$ at once.

nodes that have a similar CCT up to the $(k+1)$-th level are grouped. $\mathrm{CFG}(G)$ is iterated until the grouping does not change. The number of iterations is bounded by the longest path connecting two nodes in the graph. The proof that modified BP applied to $\mathfrak{G}$ gives the same results as BP applied to G also follows from CCTs, Eqs. (7) and (8), and the count resembling the number of identical messages sent from the nodes in G. □

In contrast to the color-passing procedure, Singla and Domingos (2008) work on the relational representation and lift the Markov logic network in a top-down fashion. Color-passing on the other hand starts from the ground network and groups together nodes and factor bottom-up. While a top-down construction of the lifted network has the advantage of being more efficient for liftable relational models since the model does not need to be grounded, a bottom-up construction has the advantage that we do not rely on a relational model such as Markov logic networks. Color-passing can group ground instances of similar clauses and atoms even if they are named differently. Reconsider the two clause example from Table 2 (top). Now, we add the clause from Table 2 (bottom) which has the same weight as the second clause and is similar in the structure, i.e. it has the same neighbors. Starting top-down, these two clauses would never be grouped by Singla and Domingos (2008) whereas color-passing would initially give these clauses the same color. More importantly, as long as we can initially color the nodes and factors, based on the evidence and the potentials respectively, color-passing is applicable to relational as well as propositional data such as Boolean formulae shown in the following.

4.1 Evaluation: lifted belief propagation

Our intention here is to investigate the following question:

(Q1) Can we scale inference in graphical models by exploiting symmetries?

To this aim, we implemented lifted belief propagation (LBP) (Ahmadi et al. 2011; Kersting et al. 2009) in C++ based on libDAI[3] with bindings to Python. We will evaluate the gain for inference by presenting significant showcases for the application of lifted BP, namely approximate inference for dynamic relational models and model counting of Boolean formulae.

Showing highly impressive lifting ratios for inference is commonly done by restricting to the classical symmetrical relational models without evidence. The two showcases for which we demonstrate lifted inference in the following, however, are particularly suited to not only show the benefits but also the shortcomings of lifted inference. Our first showcase, dynamic relational domains consists of long chains that destroy the indistinguishability of variables which might exist in a single time-step. Due to long chains, within and across time-steps, variables become correlated by virtue of sharing some common influence. The second showcase, the problem of model counting of Boolean formulae, as we will see later, is an iterative procedure that repeatedly runs inference. In each iteration new asymmetrical evidence is introduced and lifted inference is run on the modified model. Both are very challenging tasks for inference and in particular for lifting. Thus, in this section, we already address random evidence and randomness in the graphical structure that are major issues for lifted inference and training, as we will learn in the following sections.

[3] http://cs.ru.nl/~jorism/libDAI/.

Table 3 (*Top*) Example of a social network Markov logic network inspired by Singla and Domingos (2008). Free variables are implicitly universally quantified. (*Bottom*) Dynamic extension of the static social network model

English	First-Order Logic	Weight
Most people do not smoke	$\neg$`Smokes(x)`	1.4
Most people do not have cancer	$\neg$`Cancer(x)`	2.3
Most people are not friends	$\neg$`Friends(x, y)`	4.6
Smoking causes cancer	`Smokes(x)` $\Rightarrow$ `Cancer(x)`	2.0
Friends have similar smoking habits	`Friends(x, y)` $\Rightarrow$ (`Smokes(x)` <=> `Smokes(y)`)	2.0
Apriori most people do not smoke	$\neg$`Smokes(x, 0)`	1.4
Apriori most people do not have cancer	$\neg$`Cancer(x, 0)`	2.3
A priori most people are not friends	$\neg$`Friends(x, y, 0)`	4.6
Smoking causes cancer	`Smokes(x, t)` $\Rightarrow$ `Cancer(x, t)`	2.0
Friends have similar smoking habits	`Friends(x, y, t)` $\Rightarrow$ (`Smokes(x, t)` <=> `Smokes(y, t)`)	2.0
Most friends stay friends	`Friends(x, y, t)` $\Leftrightarrow$ `Friends(x, y, succ(t))`	5.0
Most smokers stay smokers	`Smokes(x, t)` $\Leftrightarrow$ `Smokes(x, succ(t))`	5.0

Lifted inference in dynamic relational domains Stochastic processes evolving over time are widespread. The truth values of relations depend on the time step t. For instance, a smoker may quit smoking tomorrow. Therefore, we extend MLNs by allowing the modeling of time. The resulting framework is called dynamic MLNs (DMLNs). Specifically, we introduce *fluents*, a special form of predicates whose last argument is time.

Here, we focus on discrete time processes, i.e., the time argument is non-negative integer valued. Furthermore, we assume a successor function `succ(t)`, which maps the integer `t` to `t + 1`. There are two kinds of formulas: *intra-time* and *inter-time* ones. Intra-time formulas specify dependencies within a time slice and, hence, do not involve the `succ` function. To enforce the Markov assumption, each term in the formula is restricted to at most one application of the `succ` function, i.e., terms such as `succ(succ(t))` are disallowed. A dynamic MLN is now a set of weighted intra- and inter-time formulas. Given the domain constants, in particular the time range $0, \ldots, T_{\max}$ of interest, a DMLN induces a MLN and in turn a Markov network over time.

As an example consider the social network DMLN shown in Table 3 (Bottom). The first three clauses encode the initial distribution at `t = 0`. The next two clauses are intra-time clauses that talk about the relationships that exist within a single time-step. They say that smoking causes cancer and that friends have similar smoking habits. Of course, these are not hard clauses as with the case of first-order logic. The weights presented in the right column serve as soft-constraints for the clauses. The last two clauses are the inter-time clause and talk about friends and smoking habits persisting over time.

Assume that there are two constants `Anna` and `Bob`. Let us say that `Bob` smokes at time `0` and he is friend with `Anna`. Then the ground Markov network will have a clique corresponding to the first two clauses for every time-step starting from 0. There will also be edges between `Smokes(Bob)` (correspondingly `Anna`) and between `Friends(Bob, Anna)` for consecutive time-steps.

To perform inference, we could employ any known MLN inference algorithm. Unlike the case for static MLNs, however, we need approximation even for sparse models: Random

variables easily become correlated over time by virtue of sharing common influences in the past.

Classical approaches to perform approximate inference in dynamic Bayesian networks (DBN) are the Boyen-Koller (BK) algorithm (Boyen and Koller 1998) and Murphy and Weiss's factored frontier (FF) algorithm (Murphy and Weiss 2001). Both approaches have been shown to be equivalent to one iteration of BP but on different graphs (Murphy and Weiss 2001). BK, however, involves exact inference, which for probabilistic logic models is extremely complex, so far does not scale to realistic domains, and hence has only been applied to rather small artificial problems. In contrast, FF is a more aggressive approximation. It is equivalent to (loopy) BP on the regular factor graph with a *forwards-backwards* message protocol: each node first sends messages from "left" to "right" and then sends messages from "right" to "left". Therefore a *frontier set* is maintained. Starting from time-step $t = 0$ we first send local messages then messages to the next time-step. A node is included in the *frontier set* iff all of its parents, that is all neighbors from time-step $t - 1$, and its neighbors from the same time-step are included. Only then it receives a message from its neighbors. The basic idea of lifted first-order factored frontier (LFF) is to *plug in lifted BP in place of BP* in FF. The local structure is replicated for all time-step in the dynamic network. Thus, the initial coloring of the nodes is the same for all time-steps. However, the communication patterns of the instantiation from different time-steps are different. Therefore, when we compress such a dynamic network nodes and factors from different time-steps end up being in different clusternodes and clusterfactors respectively. To see this, suppose we have a network consisting of a single node over three time-steps X^t, $t \in \{0, 1, 2\}$, i.e. a chain of length three. Initially all nodes get the same color. However, the signatures are different. Node X^0 only has a right neighbor, node X^1 has two neighbors (left and right) and node X^2 has a left neighbor. The *frontier set* for the lifted network is still well defined and we can run the lifted factored frontier algorithm.

We used the social network DMLN in Table 3 (bottom). There were 20 people in the domain. For fractions $r \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$ of people we randomly choose whether they smoke or not and who 5 of their friends are, and randomly assigned a time step to the information. Other friendship relations are still assumed to be unknown. `Cancer(x, t)` is unknown for all persons `x` and all time steps. The "observed" people were randomly chosen. The query predicate was `Cancer`.

In the first experiment, we investigated the compression ratio between standard FF and LFF for 10 and 15 time steps. For HMM's one iteration of the "forwards-backwards" is guaranteed to give the correct marginals, in loopy graphs, however, more iterations are necessary to propagate local as well as global information. Thus all experiments have been run to convergence with a threshold of 10^{-8} maximum of 1000 iterations. Figure 4 (left) shows the results for 10 time steps. The results for 15 were similar and therefore omitted here. As one can see, the size of the factor graph as well as the number of messages sent is much smaller for LFF.

In the second experiment, we compared the "forwards-backwards" message protocol with the "flooding" protocol, the most widely used and generally best-performing method for static networks. Using the "flooding" protocol, messages are passed from each variable to each corresponding factor and back at each iteration. Again, we considered 10 time steps. The results shown in Fig. 4 (Middle) clearly favor the FB protocol.

For a qualitative comparison, we finally computed the probability estimates for `cancer(A, t)` using LFF and MC-SAT, the default inference of the ALCHEMY system.[4]

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

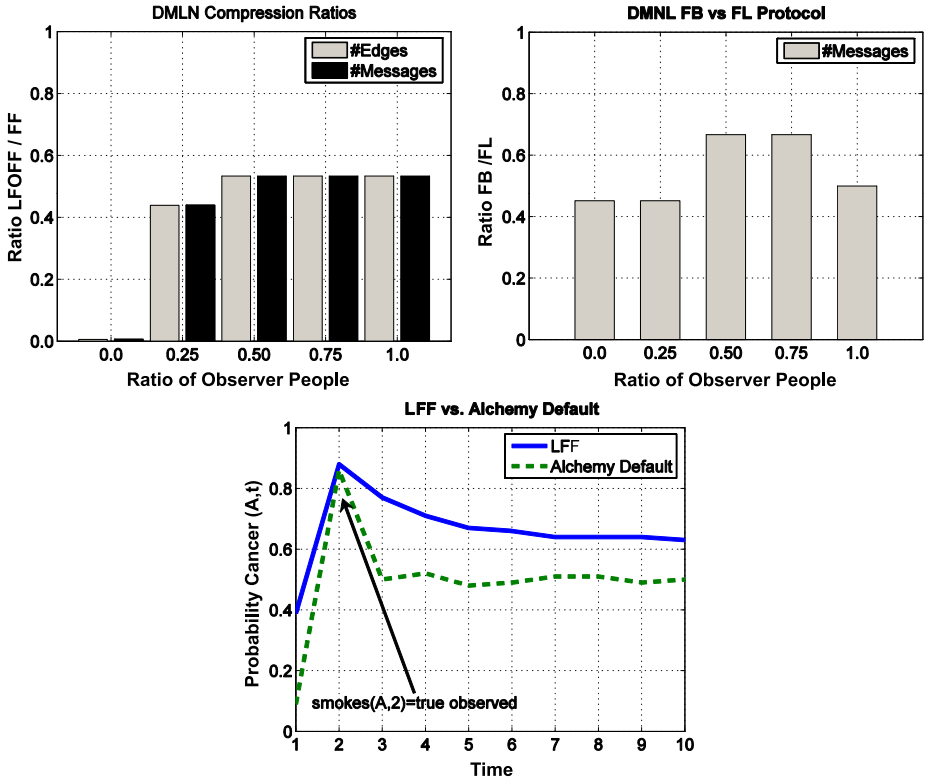


Fig. 4 (*Left*) Ratios (LFF / FF) of number of edges and messages computed. The lower the value, the greater the speed-up when using LFF in place of FF. (*Middle*) Ratios (Forwards-Backwards/Flooding protocol) of number of messages computed. The lower the value, the greater the speed-up when using the FB protocol in place of the FL protocol. (*Right*) Probability estimates for `cancer(A, t)` over time

For MC-SAT, we used default parameters. There were four persons (`A`, `B`, `C`, and `D`) and we observed that `A` smokes at time step 2. All other relations where unobserved for all time steps. We expect that the probability of `A` having cancer has a peak at $t = 2$ smoothly fading out over time. Figure 4 (right) shows the results. In contrast to LFF, MC-SAT does not show the expected behavior. The probabilities drop irrespective of the distance to the observation.

So far **Q1** is affirmatively answered. The results clearly show that by lifting we can exploit symmetries for inference in the graphical model. A compression and thereby a speed-up, however, is not guaranteed. If there are no symmetries—such as the random 3-CNF in the next section—lifted BP essentially coincides with BP.

Model counting using (lifted) belief propagation Model counting is the classical problem of computing the number of solutions of a given propositional formula. It vastly generalizes the NP-complete problem of propositional satisfiability, and hence is both highly useful and extremely expensive to solve in practice. Interesting applications include multi-agent reasoning, adversarial reasoning, and graph coloring, among others.

Our approach, called LBPCOUNT, is based on BPCOUNT for computing a probabilistic lower bound on the model count of a Boolean formula F, which was introduced by Kroc et al. (2008). The basic idea is to efficiently obtain a rough estimate of the "marginals"

of propositional variables using belief propagation with damping. The marginal of variable u in a set of satisfying assignments of a formula is the fraction of such assignments with $u = true$ and $u = false$ respectively. If this information is computed accurately enough, it is sufficient to recursively count the number of solutions of only one of "F with $u = true$" and "F with $u = false$", and scale the count up accordingly. Kroc et al. have empirically shown that BPCOUNT can provide good quality bounds in a fraction of the time compared to previous, sample-based methods.

The basic idea of LBPCOUNT now is to *plug in lifted BP in place of BP*. However, we have to be a little bit more cautious: propositional variables can appear at any position in the clauses. This makes high compression rates unlikely because, for each clusternode (set of propositional variables) and clusterfeature (set of clauses) combination, we carry a count for each position the clusternode appears in the clusterfeature. Fortunately, however, we deal with disjunctions only (assuming the formula f is in CNF). Propositional variables may appear negated or unnegated in the clauses which is the only distinction we have to make. Therefore, we can safely assume two positions (negated, unnegated) and besides sorting the node color signatures we can now also sort the factor color signatures by position. Reconsider the example from Fig. 2 and assume that the potentials associated with f_1, f_2 encode disjunctions. Indeed, assuming `B` to be the first argument of f_1 does not change the semantics of f_1. As our experimental results will show this can result in huge compression rates and large efficiency gains.

We have implemented (L)BPCOUNT based on SAMPLECOUNT[5] using our (L)BP implementation. We ran BPCOUNT and LBPCOUNT on the circuit synthesis problem `2bit-max_6` with damping factor 0.5 and convergence threshold 10^{-8}. The formula has 192 variables, 766 clauses and a true count of 2.1×10^{29}. The resulting factor graph has 192 variable nodes, 766 factor nodes, and 1800 edges.

The statistics of running (L)BPCount are shown in Fig. 5 (left). As one can see, a significant improvement in efficiency is achieved when the marginal estimates are computed using LIFTED BP instead of BP: LIFTED BP reduces the messages sent by 88.7 % when identifying the first, most balanced variable; in total, it reduces the number of messages sent by 70.2 %. Both approaches yield the same lower bound of 5.8×10^{28}, which is in the same range as Kroc et al. report. Getting exactly the same lower bound was not possible because of the randomization inherent to BPCOUNT. Constructing the compressed graph took 9 % of the total time of LIFTED BP. Overall, LBPCOUNT was about twice as fast as BPCOUNT, although our LIFTED BP implementation was not optimized.

Unfortunately, such a significant efficiency gain is not always obtainable. We ran BPCOUNT and LBPCOUNT on the random 3-CNF `wff-3-100-150`. The formula has 100 variables, 150 clauses and a true count of 1.8×10^{21}. Both approaches yield again the same lower bound, which is in the same range as Kroc et al. report. The statistics of running (L)BPCount are shown in Fig. 5 (middle). LIFTED BP is not able to compress the factor graph at all. In turn, it does not gain any efficiency but actually produces a small overhead due to trying to compress the factor graph and to compute the counts.

In real-world domains, however, there is often a lot of redundancy. As a final experiment, we ran BPCOUNT and LBPCOUNT on the Latin square construction problem `ls8-norm`. The formula has 301 variables, 1601 clauses and a true count of 5.4×10^{11}. Again, we got similar estimates as Kroc et al. The statistics of running (L)BPCount are shown in Fig. 5 (right). In the first iteration, Lifted BP sent only 0.6 % of the number of messages

[5] www.cs.cornell.edu/~sabhar/#software/.

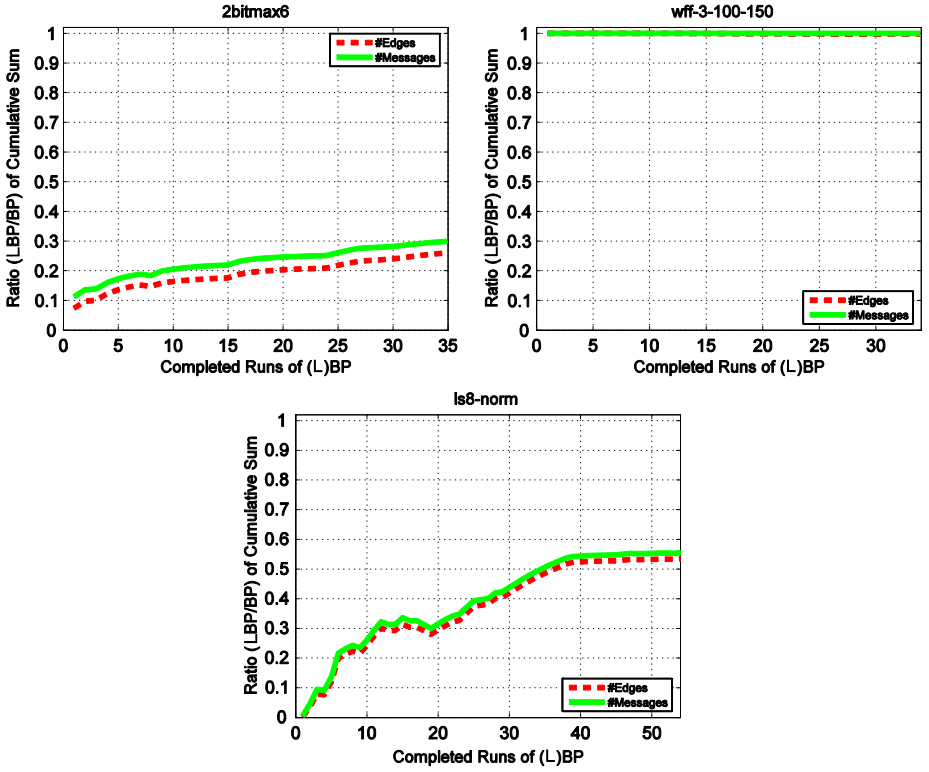


Fig. 5 Ratios LBPCOUNT/BPCOUNT between 0.0 and 1.0 of the cumulative sum of edges computed respectively messages sent. A ratio of 1.0 means that LIFTED BP sends exactly as many messages as BP; a ratio of 0.5 that it sends half as many messages. (*Left*) `2bitmax_6`: Using LIFTED BP saved 88.7 % of the messages BP sent in the first iteration of LBPCOUNT; in total, it saved 70.2 % of the messages. (*Middle*) Random 3-CNF `wff-3-100-150`: No efficiency gain. The small difference in number of edges is due to a differently selected proposition due to tie breaking. (*Right*) `ls8-norm`: In the first iteration of LBPCOUNT, using LIFTED BP saved 99.4 % of the messages BP sent. In total, this value dropped to 44.6 %

BP sent. This corresponds to 162 times fewer messages sent than BP. The result on model counting and lifted inference in dynamic relational domains clearly affirm **Q1** and show that lifted belief propagation can exploit symmetries and thus scale inference.

5 Scaling up inference: MapReduced lifting

Indeed lifted belief propagation as introduced is an attractive avenue to scaling inference. The empirical evaluation has shown lifting can render large, previously intractable probabilistic inference problems quickly solvable by employing symmetries to handle whole sets of indistinguishable random variables.

With the availability of affordable commodity hardware and high performance networking, however, we have increasing access to computer clusters providing an additional dimension to scale lifted inference. We now show that this is indeed the case. That is, we can distribute the inference and in particular the color-passing procedure for lifting message-passing using the MapReduce framework (Dean and Ghemawat 2008).

The *MapReduce* programming model allows parallel processing of massive data sets inspired by the functional programming primitives map and reduce and is basically divided into these two steps, the *Map*- and the *Reduce*-step. In the *Map*-step the input is taken, divided into smaller sub-problems and then distributed to all worker nodes. These smaller sub problems are then solved by the nodes independently in parallel. Alternatively, the sub-problems can be further distributed to be solved in a hierarchical fashion. In the subsequent *Reduce*-step all outputs of the sub-problems are collected and combined to form the output for the original problem.

If one takes a closer look at color-passing, see Algorithm 1, one notices that each iteration of color-passing basically consists of three steps, namely

1. Form the colorsignatures
2. Group similar colorsignatures
3. Assign a new color to each group

for the variables and the factors, respectively. We now show how each step can be carried out within the MapReduce framework.

Algorithm 2: MR-CP: MapReduce Color-passing

repeat
  `// Color-passing for the factor nodes`
  forall $f \in G$ **do in parallel**
    $s(f) = (s(X_1), s(X_2), \ldots, s(X_{d_f}))$, where $X_i \in nb(f)$, $i = 1, \ldots, d_f$
  Sort all of the signatures $s(f)$ in parallel using **MapReduce**;
  Map each signature $s(f)$ to a new color, s.t. $col(s(f_i)) = col(s(f_j))$ iff $s(f_i) = s(f_j)$
  `// Color-passing for the variable nodes`
  forall $X \in G$ **do in parallel**
    $s(X) = ((s(f_1), p(X, f_1)), (s(f_2), p(X, f_2)), \ldots, (s(f_{d_X}), p(X, f_{d_X})))$, where $f_i \in nb(X)$, $i = 1, \ldots, d_X$ and $p(X, f_i)$ is the position of X in f_i
  Sort all of the signatures $s(X)$ in parallel using **MapReduce**;
  Map each signature $s(X)$ to a new color, s.t. $col(s(X_i)) = col(s(X_j))$ iff $s(X_i) = s(X_j)$
until *grouping does not change*;

The resulting MapReduce color-passing is summarized in Algorithm 2 and has to be repeated in each iteration for the factors and the nodes respectively. Specifically, recall that color-passing is an iterative procedure so that we only need to take care of the direct neighbors in every iteration, i.e. building the colorsignatures for the variables (factors) requires the variables' (factors') own color and the color of the neighboring factors (variables). *Step-1* can thus be distributed by splitting the network into k parts and forming the colorsignatures within each part independently in parallel. However, care has to be taken at the borders of the splits. Figure 6 shows the factor graph from Fig. 1 and how it can be split into parts for forming the colorsignatures for the variables (Fig. 6 (left)) and the factors (Fig. 6 (right)). We see that to be able to correctly pass the colors in this step we have to introduce copynodes

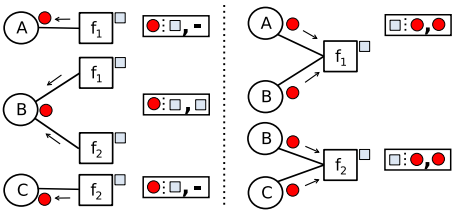


Fig. 6 The partitions when passing colors to build the signatures for the variables (*left*) and factors (*right*) and the corresponding colorsignatures that have to be sorted. Copynodes (-factors) have to be introduced at the borders to form the correct neighborhood (Color figure online)

at the borders of the parts.[6] In the case of the node signatures, for example (Fig. 6 (left)), both f_1 and f_2 have to be duplicated to be present in all signatures of the variables. The structure of the network does not change during color-passing, only the colors may change in two subsequent iterations and have to be communicated. To reduce communication cost we thus split the network into parts before we start the color-passing procedure and use this partitioning in all iterations.[7] The next step that has to be carried out is the sorting of the signatures of the variables (factors) within the network (*Step-2*). These signatures consist of the node's own color and the colors of the respective neighbors. Figure 6 also shows the resulting colorsignatures after color-passing in the partitions of the example from Fig. 1. For each node (Fig. 6 (left)) and factor (Fig. 6 (right)) we obtain the colorsignatures given the current coloring of the network. These colorsignatures have to be compared and grouped to find nodes (factors) that have the same one-step communication pattern. Finding similar signatures by sorting them can be efficiently carried out by using radix sort (Cormen et al. 2001) which has been shown to be MapReduce-able (Zhu et al. 2009). Radix sort is linear in the number of elements to sort if the length of the keys is fixed and known. It basically is a non-comparative sorting algorithm that sorts data with grouping keys by the individual digits which share the same significant position and value. The signatures of our nodes and factors can be seen as the keys we need to sort and each color in the signatures can be seen as a digit in our sorting algorithm. Radix sort's efficiency is $\mathcal{O}(kn)$ for n keys which have k or fewer digits. The close connection of color-passing to radix sort paves the way for an efficient MapReduce implementation of the lifting procedure.

Indeed, although radix sort is very efficient—the sorting efficiency is in the order of edges in the ground network—one may wonder whether it is actually well suited for an efficient implementation of lifting within a concrete MapReduce framework such as Hadoop. The Hadoop framework performs a sorting between the *Map*- and the *Reduce*-step. The key-value pair that is returned by the mappers has to be grouped and is then sent to the respective reducers to be processed further. This sorting is realized within Hadoop using an instance of quick sort with an efficiency $\mathcal{O}(n \log n)$. If we have a bounded degree of the nodes in the graph as in our case, however, this limits the length of the signatures and radix sort is still the algorithm of choice.

[6]Note that in the shared memory setting this is not necessary. Here we use the MapReduce framework, thus we have to introduce copynodes.

[7]Note that how we partition the model greatly affects the efficiency of the lifting. Finding an optimal partitioning that balances communication cost and CPU-load, however, is out of the scope of this paper. In general, the partitioning problem for parallelism is well-studied (Chamberlain 1998), there are efficient tools, e.g. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/. We show that color-passing is MapReduce-able and dramatically improves scalability even with a naive partitioning.

Moreover, our main goal is to illustrate that—in contrast to many other lifting approaches—color-passing is naturally MapReduce-able. Consequently, for the sake of simplicity, we stick to the Hadoop internal sorting to group our signatures and leave a highly efficient MapReduce realization for future work. Algorithms 3 and 4 show the map and the reduce function respectively. In the map phase we form the signatures of all nodes (factors) in a split of the graph and return a key-value pair of $(s(X_i), X_i)$ which are the signature and the id of the node (factor). These are then grouped by the reduce function and a pair $(s(L), L)$ for all nodes having the same signature, i.e. $s(L) = s(X_i)$ for all $X_i \in L$. In practice one could gain additional speed-up the if the sorting of Hadoop is avoided, for example by realizing a *Map*-only task using a distributed ordered database like *HBase*.[8] The signatures that are formed in *Step-1* would be written directly into the database as $(s(X_i)\_X_i, X_i)$, i.e. for each node an entry with a key that is composed of the signature and the node's id is inserted and the groupings could be found by sequentially reading out the values for the next step.

Algorithm 3: Map Function for Hadoop Sorting

Input: Split S of the network
Output: Signatures of nodes $X_i \in S$ as key value pairs $(s(X_i), X_i)$
forall $X_i \in S$ **do in parallel**
  $s(X_i) = (s(f_1), s(f_2), \ldots, s(f_{d_X}))$, where $f_j \in nb(X_i)$, $j = 1, \ldots, d_{X_i}$
return *key-value pairs* $(s(X_i), X_i)$

Algorithm 4: Reduce Function for Hadoop Sorting

Input: Key-value pairs $(s(X_i), X_i)$
Output: Signatures with corresponding list of nodes
Form list L of nodes X_i having same signature
return *key-value pairs* $(s(L), L)$

The *Map*-phase of (*Step-2*) could also be integrated into the parallel build of the signatures (*Step-1*), such that we have one single *Map*-step for building and sorting the signatures.

Finally, reassigning a new color (*Step-3*) is an additional linear time operation that does one pass over the nodes (factors) and assigns a new color to each of the different groups. That is, we first have to build the color signatures. The splits of the network are the input and are distributed to the mappers. In the example shown in Fig. 7 there is one line for every node and its local neighborhood, i.e. ids of the factors it is connected to. The mappers take this input and form the signatures of the current iteration independently in parallel. These signatures are then passed on to MapReduce sort. Figure 7 shows the sorting procedure for the signatures. The mappers in the sorting step, output a tuple of key-value pairs consisting of the signature and the *id* of the respective node (Algorithm 3).[9] These are then grouped by the reduce operation, such that all nodes having the same signatures are returned in a list (Algorithm 4). The internal sorting of the Hadoop framework takes place between the map- and the reduce-phase, such that the key-value pairs can be sent to the right reducers.

[8] http://hbase.apache.org/.

[9] Since this is essentially the output of the previous step that is passed through the two steps can easily be integrated. We keep them separate for illustration purposes of the two distinct steps of color-passing.

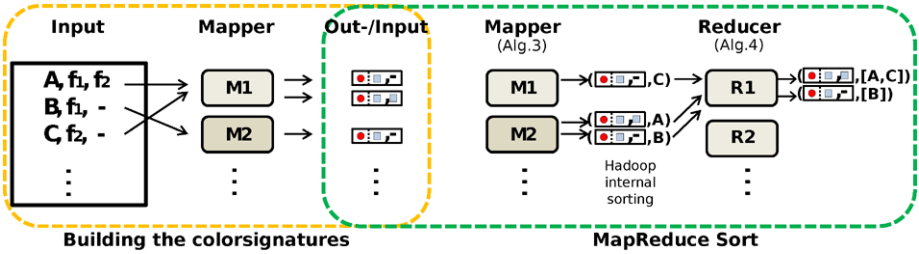


Fig. 7 MapReduce jobs for *Step-1* (*left*) and *Step-2* (*right*) of color-passing. For building the signatures the local parts are distributed to the mappers that form the signature based on the colorings of the current iteration. The output is sorted and for each signature a list of nodes with the same signature is returned (Color figure online)

Taking the MapReduce arguments for (*Step-1*) to (*Step-3*) together this proves that color-passing itself is MapReduce-able. Together with the MapReduce-able results of Gonzalez et al. (2009a, 2009b) for the modified belief propagation, this proves the following Theorem.

Theorem 2 *Lifted belief propagation is MapReduce-able.*

Moreover, we have the following time complexity, which essentially results from running radix sort h times.

Theorem 3 *The runtime complexity of the color-passing algorithm with h iterations is $\mathcal{O}(hm)$, where m is the number of edges in the graph.*

Proof Assume that every graph has n nodes (ground atoms) and m edges (ground atom appearances in ground clauses). Defining the signatures in step 1 for all nodes is an $\mathcal{O}(m)$ operation. The elements of a signature of a factor are $s(f) = (s(X_1), s(X_2), \ldots, s(X_{d_f}))$, where $X_i \in nb(f)$, $i = 1, \ldots, d_f$. Now there are two sorts that have to be carried out. The first sort is within the signatures. We have to sort the colors within a node's signatures and in the case where the position in the factor does not matter, we can safely sort the colors within the factor signatures while compressing the factor graph. Sorting the signatures is an $\mathcal{O}(m)$ operation for all nodes. This efficiency can be achieved by using counting sort, which is an instance of bucket sort, due to the limited range of the elements of the signatures. The cardinality of this signature is upper-bounded by n, which means that we can sort all signatures in $\mathcal{O}(m)$ by the following procedure. We assign the elements of all signatures to their corresponding buckets, recording which signature they came from. By reading through all buckets in ascending order, we can then extract the sorted signatures for all nodes in a graph. The runtime is $\mathcal{O}(m)$ as there are $\mathcal{O}(m)$ elements in the signatures of a graph in iteration i. The second sorting is that of the resulting signatures to group similar nodes and factors. This sorting is of time complexity $\mathcal{O}(m)$ via radix sort. The label compression requires one pass over all signatures and their positions, that is $\mathcal{O}(m)$. Hence all these steps result in a total runtime of $\mathcal{O}(hm)$ for h iterations. □

5.1 Evaluation: lifting with MapReduce

We have just shown for the first time that lifting per se can be carried out in parallel. Thus, we can now combine the gains we get from the two orthogonal approaches to scaling infer-

ence, putting scaling lifted SRL to Big Data within reach. That is, we investigate the second question:

(Q2) Does the MapReduce lifting additionally improve scalability?

We compare the color-passing procedure with a single-core Python/C++ implementation and a parallel implementation using MRJob[10] and Hadoop.[11] We partitioned the network per node (factor) and introduced copynodes (copyfactors) at the borders. The grouping of the signatures was found by a MapReduce implementation of Algorithm 3 and Algorithm 4 and mappers and reducers were distributed to four cores. Note that for this experiment the amount of lifting is not important. Here, we compare how the lifting methods scale. Whether we achieve lifting or not does not change the runtime of the lifting computations. As Theorem 3 shows, the runtime of color-passing depends on the number of edges present in the graph. Thus, we show the time needed for lifting on synthetic grids of varying size to be able to precisely control their size and the number of edges in the network. Figure 8 shows the results on grids of varying size. We scaled the grids from 5×5 to 500×500, resulting in networks with 25 to 250,000 variables. The ratio is 1 at the black horizontal line which indicates that at this point both methods are equal. Figure 8 (left) shows the ratio of the runtime for single-core color-passing and MapReduce color-passing (MR-CP) using four cores. We see that due to the overhead of MapReduce the single core methods is faster for smaller networks. However, as the number of variables grows, this changes rapidly and MR-CP scales to much larger instances. Figure 8 (right) shows the running time of single-core color-passing and MapReduce color-passing. One can see that MR-CPs scales orders of magnitude better than single-core color-passing. The results clearly favor MR-CP for larger networks and affirmatively answers (**Q2**)

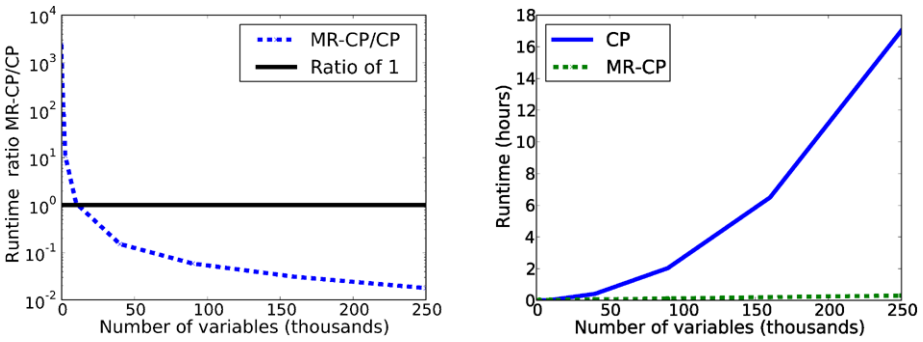


Fig. 8 (*Left*) The ratio of the runtime for single-core color-passing and MapReduce color-passing (MR-CP) on grids of varying size. The ratio is 1 at the *black horizontal line* which indicates that at this point both methods are equal. We see that due to the overhead of MapReduce for smaller networks the single core methods is a few orders of magnitude faster. However, as the number of variables grows, this changes rapidly and the MR-CP scales to much larger instances. (*Right*) Runtimes for color-passing and MR-CP. One can clearly see that MR-CP scales lifting to much larger instances

[10] https://github.com/Yelp/mrjob.

[11] http://hadoop.apache.org/.

6 Scaling up training of relational models: lifted online training

Now, we have everything together for scaling up training of relational probabilistic models. Relational models are a prominent example where symmetries abound since they combine aspects of first-order logic and probability to encode large, complex models using few probabilistic rules only. The standard parameter learning task for Markov logic networks[12] can be formulated as follows. Given a set of training instances $D = \{D_1, D_2, \ldots, D_M\}$ each consisting of an assignment to the variables in X, the goal is to output a parameter vector θ specifying a weight for each $F_i \in F$. Typically, however, only a single mega-example (Mihalkova et al. 2007) is given, a single large set of inter-connected facts. For the sake of simplicity we will sometimes denote the mega-example simply as E. To train the model, we can seek to maximize the log-likelihood or equivalently minimize the negative log-likelihood function $-\log P(D \mid \theta)$ given by

$$\ell(\theta, D) = -\frac{1}{M} \sum_{D} \log P_\theta(X = x_{D_i}). \tag{10}$$

The likelihood, however, is computationally hard to obtain. A widely-used alternative is to maximize the pseudo-log-likelihood instead i.e.,

$$\log P^*(X = x \mid \theta) = \sum_{l=1}^{n} \log P_\theta\big(X_l = x_l | MB_x(X_l)\big) \tag{11}$$

where $MB_x(X_l)$ is the state of the Markov blanket of X_l in the data, i.e. the assignment of all variables neighboring X_l. In this paper, we minimize the negative log-likelihood (Eq.(10)). No matter which objective function is used, one typically runs a gradient-descent to train the model. That is, we start with some initial parameters θ_0—typically initialized to be zero or at random around zero—and update the parameter vector using $\theta_{t+1} = \theta_t - \eta_t \cdot g_t$. Here g_t denotes the gradient of the negative log-likelihood function and is given by:

$$\partial \ell(\theta, D)/\partial \theta_k = M \mathbf{E}_{\mathbf{x} \sim P_\theta}\big[n_k(\mathbf{x})\big] - n_k(D). \tag{12}$$

This gradient expression has a particularly intuitive form: the gradient attempts to make the feature counts in the empirical data equal to their expected counts relative to the learned model. Note that, to compute the expected feature counts, we must perform inference relative to the current model. This inference step must be performed at every step of the gradient process. In the case of partially observed data, we cannot simply read-off the feature counts in the empirical data and have to perform inference there as well. Consequently, there is a close interaction between the training approach and the inference method employed for training such as lifted belief propagation.

When training a relational model for a given set of observations, however, the presence of evidence on the variables mostly destroys the symmetries. This makes lifted approaches virtually of no use if the evidence is asymmetric. In the fully observed case, this may not be a major obstacle since we can simply count how often a clause is true. Unfortunately, in many real-world domains, the mega-example available is incomplete, i.e., the truth values of some ground atoms may not be observed. For instance in medical domains, a patient rarely gets

[12] We develop our lifted training approach within the framework of Markov logic networks for illustration purposes only. We would like to stress that it naturally carries over to other relational frameworks.

all of the possible tests. In the presence of missing data, however, the maximum likelihood estimate typically cannot be written in closed form. It is a numerical optimization problem, and typically involves nonlinear, iterative optimization and multiple calls to a relational inference engine as subroutine.

Since efficient lifted inference is troublesome in the presence of partial evidence and most lifted approaches easily fall back to the ground case, we need to seek a way to make the learning task tractable. An appealing idea for efficiently training large models is to break the asymmetries, i.e., to divide the model into pieces that are trained independently and to exploit symmetries across multiple pieces for lifting.

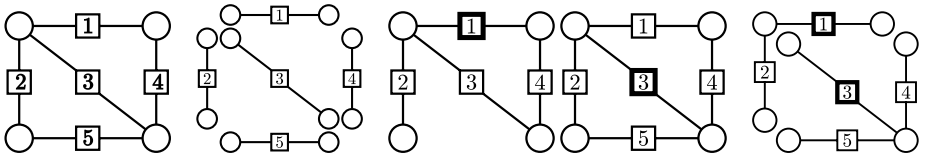
6.1 Breaking asymmetries: piecewise shattering

In piecewise training, we decompose the mega-example and its corresponding factor graph into tractable but not necessarily disjoint subgraphs (or pieces) $\mathcal{P} = \{p_1, \ldots, p_k\}$ that are trained independently (Sutton and McCallum 2009). Intuitively, the pieces turn the single mega-example into a set of many training examples and hence pave the way for online training. This is a reasonable idea since in many applications, the local information in each factor alone is already enough to do well at predicting the outputs. The parameters learned locally are then used to perform global inference on the whole model.

More formally, at training time, each piece from $\mathcal{P} = \{p_1, \ldots, p_k\}$ has a local likelihood as if it were a separate graph, i.e., training example and the global likelihood is estimated by the sum of its pieces: $\hat{\ell}(\theta, D) = \sum_{p_i \in \mathcal{P}} \ell(\theta|_{p_i}, D|_{p_i})$. Here $\theta|_{p_i}$ denotes the parameter vector containing only the parameters appearing in piece p_i and $D|_{p_i}$ the evidence for variables appearing in the current piece p_i. The standard piecewise decomposition breaks the model into a separate piece for each factor. Intuitively, however, this discards dependencies of the model parameters when we decompose the mega-example into pieces. Although the piecewise model helps to significantly reduce the cost of training, the way we shatter the full model into pieces greatly effects the learning and lifting quality. Strong influences between variables might get broken. Consequently, we next propose a shattering approach that aims at keeping strong influence but still features lifting.

6.2 Breaking asymmetries: relational tree shattering

Assume that the mega-example has been turned into a single factor graph for performing inference, cf. Fig. 9(a). Now, starting from each factor, we extract networks of depth d rooted



(a) Orig. model (b) Depth $d = 0$ (c) Depth $d = 1$ for f_1 and f_3 (d) Trees $d = 1$

Fig. 9 Schematic factor-graph depiction of the difference between likelihood (**a**), standard piecewise (**b**, **c**) and treewise training (**d**). Likelihood training considers the whole mega-example, i.e., it performs inference on the complete factor graph induced over the mega-example. Here, *circles* denote random variables, and *boxes* denote factors. Piecewise training normalizes over one factor at a time (**b**) or higher-order, complete neighborhoods of a factor (**c**) taking longer dependencies into account, here shown factors f_1 and f_3. Treewise training (**d**) explores the spectrum between (**b**) and (**c**) in that it also takes longer dependencies into account but does not consider complete higher neighborhoods; shown for tree features for factors f_1 and f_3. In doing so it balances complexity and accuracy of inference

Algorithm 5: RELTREEFINDING: Relational Treefinding

Input: Set of clauses F, a mega example E, depth d, and discount $t \in [0, 1]$
Output: Set of tree pieces T

```
// Tree-Pattern Finding
```
Initialize the dictionary of tree patterns to be empty, i.e., $P = \emptyset$;
for each *clause* $F_i \in F$ **do**
  Select a random ground instance f_j of F_i;
  Initialize tree pattern for F_i, i.e., $P_i = \{f_j\}$;
  `// perform random walk in a breadth-first manner starting in` f_j
  for $f_k =$ BFS.*next*() **do**
    if *current_depth* $> d$ **then** break;
    sample p uniformly from [0, 1];
    if $p > t^{|P_i|}$ **or** f_k *would induce a cycle* **then**
      skip branch rooted in f_k in *BFS*;
    else
      add f_k to P_i;
  Variabilize P_i and add it to dictionary P;

```
// Construct tree-based pieces using the relational tree patterns
```
for each $f_j \in E$ **do**
  Find $P_k \in P$ matching f_j, i.e., the tree pattern rooted in the clause F_k corresponding to factor f_j;
  Unify P_k with f_j to obtain piece T_j if possible and add T_j to T;
return T;

in this factor. A local network of depth $d = 0$ thus corresponds to the standard piecewise model as shown in Fig. 9(b), i.e. each factor is isolated in a separate piece. Networks of depth $d = 1$ contain the factor in which it is rooted and all of its direct neighbors, Fig. 9(c). Thus when we perform inference in such local models using say belief propagation (BP) the messages in the root factor of such a network resemble the BP messages in the global model up to the d-th iteration. Longer range dependencies are neglected. A small value for d keeps the pieces small and makes inference and hence training more efficient, while a large d is more accurate. However, it has a major weakness since pieces of densely connected networks may contain considerably large subnetworks, rendering the standard piecewise learning procedure useless.

To overcome this, we now present a shattering approach that randomly grows piece patterns forming trees. Formally, a tree is defined as a set of factors such that for any two factors f_1 and f_n in the set, there exists one and only one ordering of (a subset of) factors in the set $f_1, f_2, \ldots, f_n$ such that f_i and f_{i+1} share at least one variable, i.e. there are no loops. A tree of factors can then be generalized into a tree pattern, i.e., conjunctions of relational "clauses" by variabilizing their arguments. For every clause of the MLN we thus form a tree by performing a random walk rooted in one ground instance of that clause. This process can be viewed as a form of relational pathfinding (Richards and Mooney 1992).

Table 4 Example of a Markov Logic network about promotions and a resulting raise in income. Grounding this example leads to the ground model in Fig. 10. Figure 10 also shows the tree shattering for this network

English	First-Order Logic	Weight
Only one person can be promoted	`Promotion(x)` ⇔`!Promotion(y)`	2.0
A promotion comes with an increased income	`Promotion(x)` ⇒ `HighIncome(x)`	1.5
	`Promotion(x)` ⇒`!HighIncome(y)`	1.1

The relational treefinding is summarized in Algorithm 5. For a given set of Clauses F and a mega example E the algorithm starts off by constructing a tree pattern for each clause F_i (lines 1–13). Therefore, it first selects a random ground instance f_j (line 3) from where it grows the tree. Then it performs a breadth-first traversal of the factor's neighborhood and samples uniformly whether they are added to the tree or not (line 7). If the sample p is larger than $t^{|P_i|}$, where $t \in [0, 1]$ is a discount threshold and $|P_i|$ the size of the current tree, or the factor would induce a cycle, the factor and its whole branch are discarded and skipped in the breadth-first traversal, otherwise it is added to the current tree (lines 8–11). A small t basically keeps the size of the tree small while larger values for t allow for more factors being included in the tree. The procedure is carried out to a depth of at most d, and then stops growing the tree. This is then generalized into a piece-pattern by variablizing its arguments (line 13). All pieces are now constructed based on these piece patterns. For f_j we apply the pattern P_k of clause F_k which generated the factor (lines 15–18). In case a pattern is not applicable we apply it to the root clause, which is always possible, and as far down the tree as we can, sacrificing lifting potential for that particular piece.

These *tree-based pieces* can balance efficiency and quality of the parameter estimation well. To see this, consider the MLN in Table 4. It has three rules stating that there is only one position available and if a person gets promotion the income will be high. Grounding this example for a domain of two constants *Anna* and *Bob* leads to the ground model in Fig. 10 (left).[13] Figure 10 (center) shows the set of clauses that corresponds to the tree rooted in the factor f_3 where green colors show that the factors have been included in the piece while all red factors have been discarded. The neighborhood of the factor f_3 which corresponds to the ground clause "`Promotion(Anna)` ⇔ `!Promotion(Bob)`" is traversed in a breadth-first manner, i.e., first its direct neighbors in random order. Assume we have reached the clause factor $f_4 =$ "`Promotion(Bob)` ⇒ `HighIncome(Bob)`" first. The two ground clauses share the ground atom `Promotion(Bob)`. We uniformly sample $p \in [0, 1]$. It was small enough, e.g. $p = 0.3 < 0.9^1$ so f_4 is added to the tree. For the ground clause $f_2 =$" `Promotion(Anna)` ⇒ `HighIncome(Anna)`" we sample $p = 0.85 > 0.9^2$ so f_2 and all of its branches are discarded. Continuing, for the next ground clause $f_1 =$ "`Promotion(Anna)` ⇒ `HighIncome(Bob)`" we sample $p = 0.5 < 0.9^2$ so f_1 could be added. If we added f_1, however, it would together with f_3 and f_4 form a cycle, so its branch is discarded. For $f_5 =$ "`Promotion(Bob)` ⇒ `HighIncome(Anna)`" we sample $p = 0.4 < 0.9^2$ so it is added to the tree. Note that now we cannot add any more edges without including cycles. The set of clauses is then variablized, as shown in Fig. 10(right), to obtain the tree pattern P_k that can be applied to all groundings of the clause the root originated from. "`Promotion(Anna)` ⇔ `!Promotion(Bob)`" was the root clause of the

[13] In fact grounding the MLN would lead to more factors however for illustration purposes we assume e.g. "`Promotion(Anna)` ⇔ `!Promotion(Bob)`" and "`Promotion(Bob)` ⇔ `!Promotion(Anna)`" are simplified to a single factor.

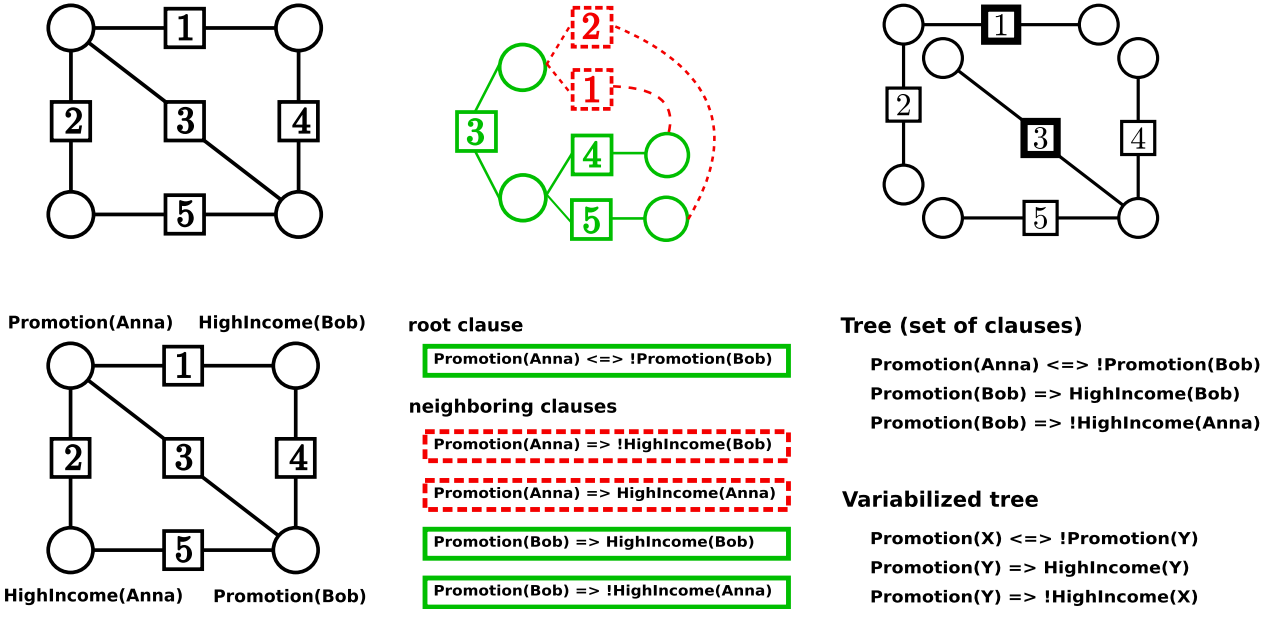


Fig. 10 Illustration of tree shattering for a factor graph (*top row*) and the Markov logic network in Table 4 for a domain with two persons (*bottom row*): from the original model (*left*) we compute a tree piece (*center*). Starting from factor f_3 (the root clause), we randomly follow the tree-structured "unrolling" of the graphical model rooted at f_3. *Green* (*solid*) shows that the factor has been included in the random walk while all *red* (*dashed*) factors have been discarded. This results in the tree pattern for f_3 shown on the *right* hand side. A similar random walk generated the other shown tree pattern for f_1 (*top right*). The resulting tree is variablized (*bottom right*) to be applied to other instantiations of the MLN clause (Color figure online)

tree pattern. Thus if we encounter another ground instance of the same clause, in this case "`Promotion(Bob)` $\Leftrightarrow$ `!Promotion(Anna)`", we find the substitution to apply this pattern, e.g. $P_k\{X \mapsto Bob, Y \mapsto Anna\}$. If the substitution is not unique, for example when we have more than two co-workers seeking for a promotion, we choose a substitution at random. as we can, sacrificing lifting potential for that particular piece. The connectivity of a piece and thereby its size can be controlled via the discount t. In this way, we include longer range dependencies in our pieces without sacrificing efficiency. And more importantly, by forming tree patterns and applying them to all factors we ensure that we have a potentially high amount of lifting: *Since we have decomposed the model into smaller pieces, the influence of the evidence is limited to a shorter range and hence allows lifting the local models.* Maximum-likelihood learning can be phrased in terms of maximizing the log-partition function $A(\theta)$ of a graphical model, and we can actually use a network decomposed into trees to maximize an upper bound on $A(\theta)$, i.e., we introduce a piecewise approximation of maximum likelihood training of relational models.

This follows from the convexity of $A(\theta)$. To see why this is the case, we first write the original parameter vector θ as a mixture of parameter vectors θ_{T_t} induced by the non-overlapping tractable subgraphs.[14] For each edge in our mega-example E, we add a tree T_t which contains all the original vertices but only the edges present in t. With each tree T_t we associate an exponential parameter vector θ_{T_t}. Let μ be a strictly positive probability distribution over the non-overlapping tractable subgraphs of each clause, such that the original parameter vector θ can be written as a combination of per-tree clause parameter vectors

$$\theta = \sum_F \sum_{t \in F} \mu_{t,F} \theta_{T_t},$$

where we have expressed parameter sharing among the ground instance of the clauses. Now using Jensen's inequality, we can state the following upper bound to the log partition function:

$$A(\theta) = A\left(\sum_F \sum_{t \in F} \mu_{t,F} \theta_{T_t}\right) = A\left(\sum_t \mu_t \theta_{T_t}\right) \leq \sum_t \mu_t A(\theta_{T_t}) \tag{13}$$

with $\mu_t = \sum_F \mu_{t,F}$. Since the $\mu_{t,F}$ are convex, the μ_t are convex, too, and applying Jensen's inequality is safe. So we can follow Sutton and Mccallum (2009) arguments. Namely, for tractable subgraphs and a tractable number of models the right-hand side of (13) can be computed efficiently. Generally, it forms an optimization problem, which according to Wainwright et al. (2002) can be interpreted as free energy and depends on a set of marginals and edge appearance probabilities, in our case the probability that an edge appears in a tree, i.e. is visited in the random walk. Also, it is easy to show that standard pieces, i.e. one factor per piece, are an upper bound to this bound since, we can apply Jensen's inequality again when breaking the trees into independent paths from the root to the leaves.

Now, we show how to turn this upper bound into a lifted online training for relational models.

6.3 Lifted online training via stochastic meta-descent

Stochastic gradient descent algorithms update the weight vector in an online setting. We essentially assume that the pieces are given one at a time. The algorithms examine the current

[14] The subgraphs produced by Algorithm 5 may overlap. We will show how to account for this in Sect. 6.3.

piece and then update the parameter vector accordingly. They often scale sub-linearly with the amount of training data, making them very attractive for large training data as targeted by statistical relational learning. To reduce variance, we may form *mini-batches* consisting of several pieces on which we learn the parameters locally. In contrast to the propositional case, however, mini-batches have another important advantage: we can now make use of the symmetries within and *across* pieces for lifting. The pieces within a mini-batch can be seen as disconnected parts of a larger network. Now if we perform inference for the whole mini-batch we naturally exploit the symmetries within each piece and across the pieces in the mini-batch.

The bound of Eq. (13) holds for tree pieces that partition the graph into disjoint sets of factors. The relational tree shattering finds a pattern for each clause and applies it to all the ground instances. To cope for the multiplicity introduced we normalize the gradient by the number of appearances of a clause. More formally, the gradient in (12) is approximated by

$$\sum_i \#_i^{-1} \cdot \frac{\partial \ell(\theta, \boldsymbol{D}_i)}{\partial \theta_k}, \tag{14}$$

where the mega-example $\boldsymbol{D}$ is broken up into pieces respectively mini-batches of pieces $\boldsymbol{D}_i$. Here the vector $\#_i$ denotes a per-clause normalization that counts how often each clause appears in mini-batch $\boldsymbol{D}_i$ and $\cdot$ is the component-wise multiplication. This is a major difference to the propositional case and avoids "double counting" parameters which otherwise would be the case when training from the tree pieces due to potential multiplicity of factors. Recall that we build a piece from every grounding of each clause. During the random walk ground clauses can be visited repeatedly such that they appear in multiple different pieces. The normalization by $\#_i$ accounts for this fact. For example, let g_i be a gradient over the mini-batch $\boldsymbol{D}_i$. For a single piece we count how often a ground instance of each clause appears in the piece $\boldsymbol{D}_i$. If $\boldsymbol{D}_i$ consists of more than one piece we add the count vector of all pieces together. For example, if for a model with 4 clauses the single piece mini-batch $\boldsymbol{D}_i$ has counts $(1, 3, 0, 2)$ the gradient is normalized by the respective counts. If the mini-batch, however, has an additional piece with counts $(0, 2, 1, 0)$ we normalize by the sum, i.e. $(1, 5, 1, 2)$.

Since the gradient involves inference per batch only, inference is again feasible and more importantly liftable as we will show in Sect. 8.1. Consequently, we can scale to problem instances traditional relational methods can not easily handle. However, the asymptotic convergence of first-order stochastic gradients to the optimum can often be painfully slow if e.g. the step-size is too small. One is tempted to just employ standard advanced gradient techniques such as L-BFGS. As the gradient is stochastically approximated by random subsamples, the measurements are inherently noisy. This confuses the line searches of conjugate gradient and quasi-Newton methods as conjugacy of the search direction over multiple iterations can not be maintained (Schraudolph and Graepel 2003). Gain adaptation methods like Stochastic Meta-Descent (SMD) overcome these limitations by using second-order information to adapt a per-parameter step size (Vishwanathan et al. 2006). However, while SMD is very efficient in Euclidean spaces, Amari (1998) showed that the parameter space is actually a Riemannian space of the metric C, the covariance of the gradients. Consequently, the ordinary gradient does not give the steepest direction of the target function which is instead given by the natural gradient, that is by $C^{-1}\boldsymbol{g}$. Intuitively, the natural gradient is more conservative and does not allow large variances. If the gradients highly disagree in one direction, one should not take the step. Thus, whenever we have computed a new gradient $\boldsymbol{g}_t$ we integrate its information and update the covariance at time step t by the following expression:

$$C_t = \gamma C_{t-1} + \boldsymbol{g}_t \boldsymbol{g}_t^T \tag{15}$$

where $C_0 = \mathbf{0}$, and γ is a parameter that controls how much older gradients are discounted. Now, let each parameter θ_k have its own step size η_k. We update the parameter by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{\eta}_t \cdot \boldsymbol{g}_t, \tag{16}$$

where $\cdot$ denotes component-wise multiplication. The gain vector $\boldsymbol{\eta}_t$ thus serves as a diagonal conditioner. The vector $\boldsymbol{\eta}$ containing the individual per-parameter step sizes is adapted with the meta-gain μ:

$$\boldsymbol{\eta}_{t+1} = \boldsymbol{\eta}_t \cdot \exp(-\mu \boldsymbol{g}_{t+1} \cdot \boldsymbol{v}_{t+1}) \approx \boldsymbol{\eta}_t \cdot \max\left(\frac{1}{2}, 1 - \mu \boldsymbol{g}_{t+1} \cdot \boldsymbol{v}_{t+1}\right) \tag{17}$$

where $\boldsymbol{v} \in \Theta$ characterizes the long-term dependence of the system parameters on gain history. The time scale is determined by the decay factor $0 \leq \lambda \leq 1$. The vector $\boldsymbol{v}$ is iteratively updated by

$$\boldsymbol{v}_{t+1} = \lambda \boldsymbol{v}_t - \boldsymbol{\eta} \cdot \left(\boldsymbol{g}_t + \lambda C^{-1} \boldsymbol{v}_t\right). \tag{18}$$

To ensure a low computational complexity and a good stability of the computations, one can maintain a low rank approximation of C, see Le Roux et al. (2007) for more details. Using per-parameter step-sizes considerably accelerates the convergence of stochastic natural gradient descent.

Algorithm 6: Lifted Online Training of Relational Models

Input: Markov Logic Network M, mega-example E, decay factors t, γ, and λ, initial step-size η
Output: Parameter vector θ

```
// Generate mini-batches
```
1 Generate set of tree pieces $\mathcal{T}$ using RelTreeFinding;
2 Randomly form mini-batches $\mathcal{B} = \{B_1, \ldots, B_m\}$ each consisting of l pieces;
```
// Perform lifted stochastic meta-descent
```
3 Initialize θ and v_0 with zeros and the covariance matrix C to the zero matrix;
4 **while** *not converged* **do**
5 Shuffle mini-batches $\mathcal{B}$ randomly;
6 **for** $i = 1, 2, \ldots, m$ **do**
7 Compute gradient g for B_i using lifted belief propagation;
8 Update covariance matrix C using (15);
9 Update parameter vector θ using (16) and the involved equations;
10
11 **return** θ;

Putting everything together, we arrive at the lifted online learning for relational models as summarized in Algorithm 6. We form mini-batches of tree pieces (lines 1–2). After initialization (line 3), we then perform lifted stochastic meta-descent (lines 4–9). That is, we randomly select a mini-batch, compute its gradient using lifted inference, and update the parameter vector. Note that pieces and mini-batches can also be computed on the fly and thus their construction be interweaved with the parameter update. We iterate these steps until convergence, e.g. by considering the change of the parameter vector in the last l steps. If

the change is small enough, we consider it as evidence of convergence. To simplify things, we may also simply fix the number of times we cycle through all mini-batches. This also allows to compare different methods.

7 Scaling up training of relational models: MapReduced stochastic gradient

So far, we have shown how the model can be shattered into smaller pieces to efficiently learn the parameters. This shattering makes training large models tractable and improves on the speed of the convergence, as we will show in the experimental section. Even more importantly, as each lifted piece is processed one after the other it naturally paves the way for a MapReduce approach. The gradients of the shattered pieces can be computed locally in a distributed fashion which in turn allows a *MapReduce* friendly parallel approach without bandwidth constrains and considerable latency, see e.g. Zinkevich et al. (2010), Langford et al. (2009). This proves the following theorem.

Theorem 4 *Lifted approximate training of relational models is MapReduce-able.*

Now, we have everything together to investigate scalable lifted inference and training.

8 Scalable lifted inference and training: experimental evaluation

Our intention here is to investigate the following questions:

(Q3) Does piecewise lifted inference help in non-symmetric cases?
(Q4) Can we efficiently train relational models using stochastic gradients?
(Q5) Are there symmetries within mini-batches that result in lifting?
(Q6) Can relational treefinding produce pieces that balance accuracy and lifting well?
(Q7) Is it even possible to achieve one-pass relational training?

To this aim, we implemented lifted online learning for relational models in Python and C++. As a batch learning reference, we used *scaled conjugate gradient (SCG)* (Møller 1993). SCG chooses the search direction and the step size by using information from the second order approximation. For inference we used our lifted belief propagation (LBP) (Ahmadi et al. 2011; Kersting et al. 2009) implementation. Inference as a subroutine for the training methods was also carried out by LBP to convergence with a threshold of 10^{-8} maximum of 1000 iterations.

For the evaluation of the training approaches, we computed the *conditional marginal log-likelihood* (CMLL) (Lee et al. 2007), which is defined with respect to marginal probabilities. More precisely, we first divide the variables into two groups: X_{hidden} and $X_{observed}$. Then, we compute

$$\text{CMLL} = \sum_{X \in X_{hidden}} \log P(X | X_{observed}) \tag{19}$$

for the given mega-example. To stabilize the metric, we divided the variables into four groups and calculated the average CMLL when observing only one group and hiding the rest. Instead of the global log-partition function it is defined in terms of the marginal probabilities. CMLL measures the ability to predict each variable separately. All experiments were conducted on a single machine with 2.4 GHz and 64 GB of RAM.

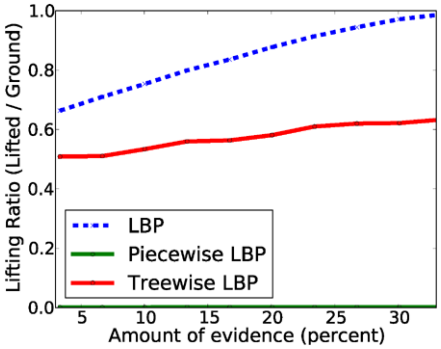


Fig. 11 Lifting Ratio for CORA entity resolution MLN with varying amount of evidence. Although there is some lifting initially for LBP, with a certain amount of evidence lifted belief propagation is basically ground. Piecewise lifted inference, on the other hand, achieves networks that are orders of magnitude smaller than standard lifting. The trees keep some dependencies compared to the standard pieces and still manage to achieve compression for LBP

8.1 Lifted piecewise inference (Q3)

The experiments in the previous sections have shown that we can considerably speed up inference by lifting. However, in cases where there are no symmetries or symmetries are broken by asymmetric evidence we do not gain and lifted belief propagation basically falls back to the propositional case. This is the case when naively applying lifted inference to the training of relational models. An appealing idea for such situations is to run inference locally. By breaking the model into pieces the influence is limited and we can gain significant lifting of the model. Figure 11 shows the results for the Cora entity resolution MLN, which we will also train later in Sect. 8.4. We sampled 10 bibliography entries and extracted all facts corresponding to these bibliography entries. Then we varied the amount of evidence on the facts. One can see that lifted BP achieves some compression at the beginning. However, with a certain amount of evidence lifted belief propagation is basically ground and achieves very little compression. Piecewise lifted inference, on the other hand, achieves networks that are orders of magnitude smaller compared to standard lifting. However, by shattering the model into pieces all dependencies are broken. Tree-pieces balance this trade-off. The trees keep some dependencies and still manage to achieve compression for LBP. As we will see in the training experiments in Sect. 8 these additional dependencies will help to train the model faster. Lifted Piecewise inference clearly enables lifting in non-symmetric cases where standard lifting fails.

8.2 Training of friends-and-smokers MLN (Q4, Q5)

In our first training experiment we learned the parameters for the "Friends-and-Smokers" MLN (Singla and Domingos 2008), which basically defines rules about the smoking behavior of people, how the friendship of two people influences whether a person smokes or not, and that a person is more likely to get cancer if he smokes. The "Friends-and-Smokers" MLN, however, is an oversimplification of the effects and rich interactions in social networks. Thus we enriched the network by adding two clauses: if someone is stressed he is more likely to smoke and people having cancer should get medical treatment. For a given set of parameters we sampled 5 datasets from the joint distribution of the MLN with 10 persons. For each dataset we learned the parameters on this dataset and evaluated on the other

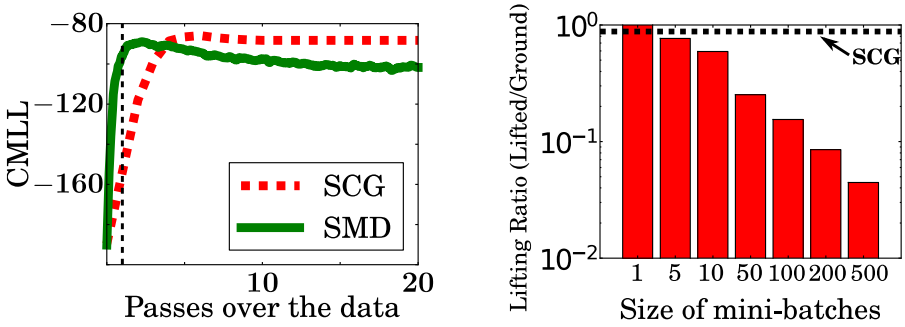


Fig. 12 "Passes over mega-example" vs. Test-CMLL for the Friends-and-Smokers (*left*) (the higher the better). Lifted online learning has already learned before seeing the mega example even once (*dashed vertical line*). (*Right*) Benefit of local training for lifting. Lifting ratio for varying mini-batch size versus the full batch model on the Friends-and-Smokers MLN. Clearly for a batch size of 1 there is no lifting but with larger mini-batch sizes there is more potential to lift the pieces within each batch; the size can be an order of magnitude smaller (Color figure online)

four. The ground network of this MLN contains 380 factors and 140 variables. The batch size was 10 and we used a stepsize of 0.2. Other parameters for SMD were chosen to be $\lambda = .99$, $\mu = 0.1$, and γ the discount for older gradients as 0.9. Figure 12 (left) shows the CMLL averaged over all of the 5 folds.

As one can see, the lifted SMD using single factor pieces has a steep learning curve and has already learned the parameters before seeing the mega example even once (indicated by the dashed vertical line). Note that we learned the models without stopping criterion and for a fixed number of passes over the data thus the CMLL on the test data can decrease. SCG on the other hand requires four passes over the entire training data to have a similar result in terms of CMLL. Thus **Q4** can be answered affirmatively. Moreover, as Fig. 12 (right) shows, piecewise learning greatly increases the lifting compared to batch learning, which essentially does not feature lifting at all. Thus, **Q5** can be answered affirmatively.

8.3 Voting MLN (Q5)

To investigate whether tree pieces although more complex can still yield lifting, we considered the Voting MLN from the Alchemy repository. The network contains 3230 factors and 3230 variables. Note that it is a propositional Naive Bayes (NB) model. Hence, depth 0 pieces will yield greater lifting but hamper information flow among attributes if the class variable is unobserved. Tree pieces intuitively couple depth 0 hence will indeed yield lower lifting ratios. However, with larger mini-batches they should still yield higher lifting than the batch case. This is confirmed by the experimental results summarized in Fig. 13 (left) that shows the lifting ratio for standard pieces vs. tree pieces with depth $d = 1$ with a threshold of $t = 0.9$. Thus, (**Q5**) can be answered affirmatively.

8.4 Training of CORA entity resolution MLN (Q6, Q7)

Here we learned the parameters for the Cora entity resolution MLN,[15] one of the standard datasets for relational learning. In the current paper, however, it is used in a non-standard,

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

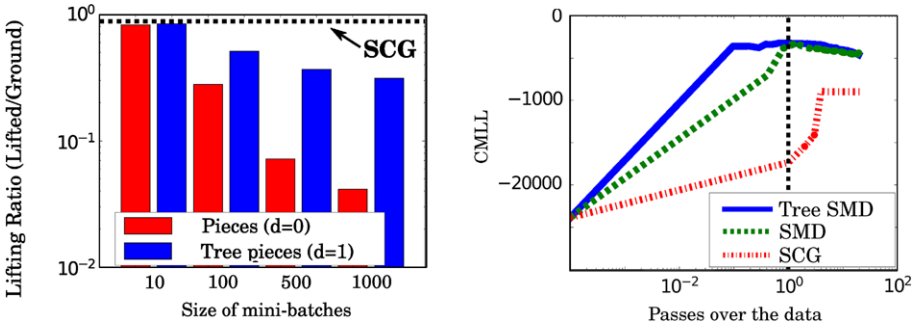


Fig. 13 Experimental results. (*Left*) Lifting ratio for standard pieces vs. tree pieces on the Voting MLN. Due to rejoining of pieces, additional symmetries are broken and the lifting potential is smaller. However, the sizes of the models per mini-batch still gradually decrease with larger mini-batch sizes. (*Right*) "passes over mega-example" vs. Test-CMLL for the CORA MLN (the higher the better) (Color figure online)

more challenging setting. For a set of bibliography entries (papers) the Cora MLN has facts, e.g., about word appearances in the titles and in author names, the venue a paper appeared in, its title, etc. The task is now to infer whether two entries in the bibliography denote the same paper (predicate *samePaper*), two venues are the same (*sameVenue*), two titles are the same (*sameTitle*), and whether two authors are the same (*sameAuthor*). We sampled 20 bibliography entries and extracted all facts corresponding to these bibliography entries. We constructed five folds then trained on four folds and tested on the fifth. The mega-example E is composed of the four folds we train on. We employed a transductive learning setting for this task. The MLN was parsed with all facts for the bibliography entries from the five folds, i.e., the queries were hidden for the test fold. The query consisted of all four predicates (*sameAuthor*, *samePaper*, *sameBib*, *sameVenue*). The resulting ground network consisted of 36,390 factors and 11,181 variables. We learned the parameters using SCG, lifted stochastic meta-descent with standard pieces as well as pieces using relational treefinding with a depth d of 1 and a threshold t of 0.9. The trees consisted of around ten factors on average. So we updated with a batch size of 100 for the trees and 1000 for standard pieces with a stepsize of 0.05. Furthermore, other parameters were chosen to be $\lambda = .99$, $\mu = 0.9$, and $\gamma = 0.9$. Figure 13 (right) shows the averaged learning results for this entity resolution task. Again, online training does not need to see the whole mega-example; it has learned long before finishing one pass over the entire data. Thus, (**Q6**) can be answered affirmatively.

Moreover, Fig. 13 also shows that by building tree pieces one can considerably speed-up the learning process. They convey a lot of additional information such that one obtains a better solution with a smaller amount of data. This is due to the fact that the Cora dataset contains a lot of strong dependencies which are all broken if we form one piece per factor. The trees on the other hand preserve parts of the local structure which significantly helps during learning. Thus, (**Q7**) can be answered affirmatively.

8.5 Lifted imitation learning in the Wumpus domain (Q6, Q7)

To further investigate (**Q6**) and (**Q7**), we considered imitation learning in a relational domain for a Partially Observed Markov Decision Process (POMDP). We created a simple version of the Wumpus task (Russell and Norvig 2003) where the location of Wumpus is partially observed. We used a 5×5 grid with a Wumpus placed in a random location in every training trajectory. The Wumpus is always surrounded by stench on all four sides. We do not have

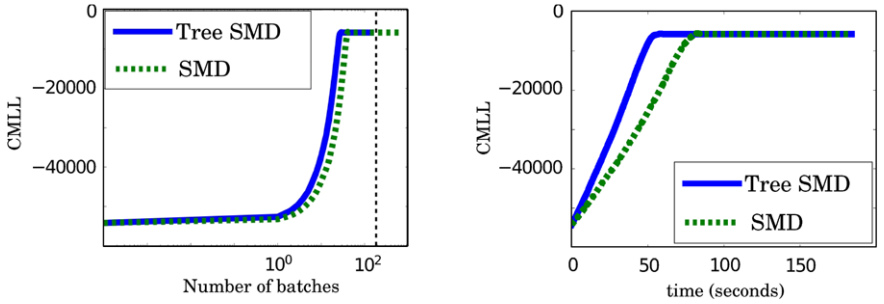


Fig. 14 Experimental results for the Wumpus MLN. (*Left*) "number of batches" vs. Test-CMLL (the higher the better). (*Right*) Runtime vs. CMLL. As one can see, lifted online learning has already converged before seeing the mega example even once (*black vertical line*). For the Wumpus MLN, SCG did not converge within 72 hours (Color figure online)

any pits or breezes in our task. The agent can perform 8 possible actions: 4 move actions in each direction and 4 shoot actions in each direction. The agent's task is to move to a cell so that he can fire an arrow to kill the Wumpus. The Wumpus is not observed in all the trajectories although the stench is always observed. Trajectories were created by real human users who play the game.

The cells of the 5×5 grid were numbered and we use predicates like $cellAtRow(cell, row)$ and $cellAbove(cell, cell)$ to define the structure of the grid and were the cell is located. These facts were always given. Other predicates were $wumpus(cell)$, $stench(cell)$, $agent(cell, t)$ and actions move/shoot for all for directions, e.g. $shootUp(t)$. The rules we learn the weights for describe the state or whether an action should be performed. Two examples of such rules are:

$$w_1: \quad stench(scell) \wedge cellAbove(scell, wcell) => wumpus(wcell)$$
$$w_2: \quad wumpus(wcell) \wedge agent(acell, t) \wedge cellCol(acell, acol) \wedge cellCol(wcell, wcol) \wedge less(acol, wcol) => shootRight(t)$$

The resulting network contains 182400 factors and 4469 variables. We updated with a batch size of 200 for the trees (depth $d = 0$, threshold $t = 0.9$) and 2000 for standard pieces with a stepsize of 0.05. As for the Cora dataset used $\lambda = .99$, $\mu = 0.9$, and $\gamma = 0.9$. Figure 14 shows the result on this dataset for lifted SMD with standard pieces as well as pieces using relational treefinding with a threshold t of 0.9. For this task, SCG did not converge within 72 hours. Note that this particular network has a complex structure with lots of edges and large clauses. This makes inference on the global model intractable. Figure 14 shows the learning curve as a function of the total number of batches seen (left) as well as the runtime (right) for one pass over the data. As one can see, tree pieces actually yield faster convergence, again long before having seen the dataset even once. Thus, (**Q6**) and (**Q7**) can be answered affirmatively.

Taking all experimental results together, all questions **Q1**–**Q7** can be clearly answered affirmatively.

9 Conclusions

Symmetries can be found almost everywhere, in arabesques and French gardens, in the rose windows and vaults in Gothic cathedrals, in the meter, rhythm, and melody of music, in the

metrical and rhyme schemes of poetry as well as in the patterns of steps when dancing. Symmetric faces are even said to be more beautiful to humans. Actually, symmetry is both a conceptual and a perceptual notion often associated with beauty-related judgments (Zaidel and Hessamian 2010). Or, to quote Hermann Weyl "Beauty is bound up with symmetry" (Weyl 1952). This link between symmetry and beauty is often made by scientists. In physics, for instance, symmetry is linked to beauty in that symmetry describes the invariants of nature, which, if discerned, could reveal the fundamental, true physical reality (Zaidel and Hessamian 2010). In mathematics, as Herr and Bödi note, "we expect objects with many symmetries to be uniform and regular, thus not too complicated" (Herr and Bödi 2010). Therefore, it is not surprising that symmetries have also been explored in many AI tasks. For instance, there are symmetry-aware approaches in (mixed-)integer programming (Bödi et al. 2011; Margot 2010), linear programming (Herr and Bödi 2010; Mladenov et al. 2012), SAT and CSP (Sellmann and Van Hentenryck 2005) as well as MDPs (Dean and Givan 1997; Ravindran and Barto 2001).

Surprisingly, however, symmetries have not been the subject of great interest within statistical learning. In this paper, we have shown that scaling inference and relational training of graphical models can actually greatly benefit from symmetries. We have introduced lifted belief propagation and shown that lifting is MapReduce-able. However, already in 1848, Louis Pasteur recognized "Life as manifested to us is a function of the asymmetry of the universe". This remark characterizes somehow one of the main challenges we are facing: Not only are almost all large graphs asymmetric (Erdös and Rényi 1963), but *even if there are symmetries within a model, they easily break when it comes to inference and training since variables become correlated by virtue of depending asymmetrically on evidence*. This, however, does not mean that lifted inference and training is hopeless. We have demonstrated that breaking long-term dependencies via piece-wise inference and training naturally breaks asymmetries and paves the way to lifted online respectively MapReduced relational training.

The symmetry-aware framework for learning outlined in the present paper puts many interesting research goals into reach. For instance, one should tackle one-pass relational learning by investigating different ways of gain adaption and scheduling of pieces for updates. Since piecewise training is a simple form of dual decomposition, further exploration of dual decomposition methods is an attractive future direction. One should also investigate budget constraints on both the number of examples and the computation time per iteration. Another interesting avenue for future work is to use sequences of increasingly finer approximations to control the trade-off between lifting and accuracy (Kiddon and Domingos 2011). Besides belief propagation, lifted message passing approaches have been introduced for Gaussian belief propagation (Ahmadi et al. 2011), warning propagation and survey propagation (Hadiji et al. 2011). The definition of an abstract lifted message passing framework that unifies these and other message passing algorithms remains to be done and is very interesting future work. Finally, one should start investigating symmetries in general machine learning approaches such as support-vector machines and Gaussian processes. So, while there have been considerable advances, there are more than enough problems, in particular asymmetric ones, to go around to really establish symmetry-aware machine learning.

Acknowledgements The authors would like to thank the anonymous reviewers for their helpful comments. Thanks to Roman Garnett, Fabian Hadiji and Mirwaes Wahabzada for helpful discussion on map-reduce, and Tushar Khot for kindly providing the Wumpus dataset. The authors also would like to thank Tuyen N. Huynh and Raymond J. Mooney for communicating their idea and code on online discriminative learning. Babak Ahmadi and Kristian Kersting were supported by the Fraunhofer ATTRACT fellowship STREAM and by the European Commission under contract number FP7-248258-First-MM. Martin Mladenov and Kristian Kersting were supported by the German Research Foundation DFG, KE 1686/2-1, within the SPP 1527. Sriraam Natarajan gratefully acknowledges the support of the DARPA Machine Reading Program under

AFRL prime contract no. FA8750-09-C-0181. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of DARPA, AFRL, or the US government.

References

Acar, U., Ihler, A., Mettu, R., & Sumer, O. (2008). Adaptive inference on general graphical models. In *UAI-08*, Corvallis, Oregon: AUAI Press.

Ahmadi, B., Kersting, K., & Sanner, S. (2011). Multi-evidence lifted message passing, with application to pagerank and the Kalman filter. In *IJCAI*.

Ahmadi, B., Kersting, K., & Natarajan, S. (2012). Lifted online training of relational models with stochastic gradient methods. In *Proceedings ECML-PKDD*, Bristol, UK, September 24–28. Berlin: Springer.

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, *10*, 251–276.

Besag, J. (1975). Statistical analysis of non-lattice data. *Journal of the Royal Statistical Society. Series D. The Statistician*, *24*(3), 179–195.

Bödi, R., Herr, K., & Joswig, M. (2011). Algorithms for highly symmetric linear and integer programs. In *Mathematical Programming, Series A*. Online First.

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic processes. In *Proc. of the conf. on uncertainty in artificial intelligence (UAI-98)* (pp. 33–42).

Bui, H. H., Huynh, T., & de Salvo Braz, R. (2012). Exact lifted inference with distinct soft evidence on every object. In *Proc. of the 26th AAAI conf. on artificial intelligence (AAAI 2012)*.

Bui, H.H., Huynh, T. N., & Riedel, S. (2012). Automorphism groups of graphical models and lifted variational inference. In *CoRR*.

Chamberlain, B. L. (1998). *Graph partitioning algorithms for distributing workloads of parallel computations* (Technical report).

Choi, J., Hill, D., & Amir, E. (2010). Lifted inference for relational continuous models. In *UAI*.

Choi, J., de Salvo Braz, R., & Bui, H. (2011). Efficient methods for lifted inference with aggregate factors. In *AAAI 2011*.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms* (2nd ed.). Cambridge: MIT Press.

Darwiche, A. (2001). Recursive conditioning. *Artificial Intelligence*, *126*(1–2), 5–41.

De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. H. (eds.) (2008). *Lecture notes in computer science: Vol. 4911. Probabilistic inductive logic programming*. Berlin: Springer.

de Salvo Braz, R., Amir, E., & Roth, D. (2005). Lifted first order probabilistic inference. In *Proc. of the 19th international joint conference on artificial intelligence (IJCAI-05)* (pp. 1319–1325).

de Salvo Braz, R., Amir, E., & Roth, D. (2006). MPE and partial inversion in lifted probabilistic variable elimination. In *Proc. of the 21st AAAI conf. on artificial intelligence (AAAI-06)*.

de Salvo Braz, R., Natarajan, S., Bui, H., Shavlik, J., & Russell, S. (2009). Anytime lifted belief propagation. In *Proc. SRL-09*.

Dean, J., & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113.

Dean, T., & Givan, R. (1997). Model minimization in Markov decision processes. In *Proc. of the fourteenth national conf. on artificial intelligence (AAAI-97)* (pp. 106–111).

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39.

Erdös, P., & Rényi, A. (1963). Asymmetric graphs. *Acta Mathematica Academiae Scientiarum Hungaricae*, *14*, 295–315.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 1189–1232.

Getoor, L., & Taskar, B. (2007). *Introduction to statistical relational learning*. Cambridge: 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.

Gogate, V., & Domingos, P. (2010). Exploiting logical structure in lifted probabilistic inference. In *Working note of the AAAI-10 workshop on statistical relational artificial intelligence*.

Gogate, V., & Domingos, P. (2011). Probabilistic theorem proving. In *Proc. ot the 27th conf. on uncertainty in artificial intelligence (UAI)*.

Gonzalez, J. E., Low, Y., & Guestrin, C. (2009b). Residual splash for optimally parallelizing belief propagation. In *Artificial intelligence and statistics (AISTATS)* (pp. 177–184).

Gonzalez, J., Low, Y., Guestrin, C., & O'Hallaron, D. (2009a). Distributed parallel inference on large factor graphs. In *UAI*, Montreal, Canada, July 2009.
Gutmann, B., Thon, I., & De Raedt, L. (2011). Learning the parameters of probabilistic logic programs from interpretations. In *ECML-PKDD* (pp. 581–596).
Hadiji, F., Ahmadi, B., & Kersting, K. (2011). Efficient sequential clamping for lifted message passing. In *Proceedings of the 34th annual German conference on artificial intelligence (KI–11)*. Berlin: Springer.
Herr, K., & Bödi, R. (2010). Symmetries in linear and integer programs. In *CoRR*. 0908.3329
Hinton, G. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, *14*.
Huynh, T., & Mooney, R. (2011). Online max-margin weight learning for Markov logic networks. In *SDM*.
Ihler, A. T., Fisher, J. W. III, & Willsky, A. S. (2005). Loopy belief propagation: convergence and effects of message errors. *Journal of Machine Learning Research*, *6*, 905–936.
Jaeger, M. (2007). Parameter learning for Relational Bayesian networks. In *ICML*.
Jaimovich, A., Meshi, O., & Friedman, N. (2007). Template-based inference in symmetric relational Markov random fields. In *Proc. of the conf. on uncertainty in artificial intelligence (UAI-07)* (pp. 191–199).
Kersting, K. (2012). Lifted probabilistic inference. In L. De Raedt, C. Bessiere, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, & P. Lucas (Eds.), *Proceedings of 20th European conference on artificial intelligence (ECAI–2012)*, Montpellier, France, August 27–31, 2012. Amsterdam: ECCAI IOS Press. (Invited talk at the frontiers of AI track).
Kersting, K., & De Raedt, L. (2001). Adaptive Bayesian logic programs. In *ILP*.
Kersting, K., Ahmadi, B., & Natarajan, S. (2009). Counting belief propagation. In *UAI*, Montreal, Canada.
Khot, T., Natarajan, S., Kersting, K., & Shavlik, J. (2011). Learning Markov logic networks via functional gradient boosting. In *ICDM*.
Kiddon, C., & Domingos, P. (2011). Coarse-to-fine inference and learning for first-order probabilistic models. In *Proc. of the 25th AAAI conf. on artificial intelligence (AAAI 2011)*.
Kisynski, J., & Poole, D. (2009). Constraint processing in lifted probabilistic inference. In *UAI* (pp. 293–302).
Kok, S., & Domingos, P. (2009). Learning Markov logic network structure via hypergraph lifting. In *ICML*.
Kok, S., & Domingos, P. (2010). Learning Markov logic networks using structural motifs. In *ICML*.
Kroc, L., Sabharwal, A., & Selman, B. (2008). Leveraging belief propagation, backtrack search, and statistics for model counting. In *Proc. of the 5th int. conf. on the integration of AI and OR techniques in constraint programming for combinatorial optimization problems (CPAIOR-08)* (pp. 127–141).
Langford, J., Smola, A. J., & Zinkevich, M. (2009). Slow learners are fast. In *NIPS* (pp. 2331–2339).
Le Roux, N., Manzagol, P.-A., & Bengio, Y. (2007). Topmoumoute online natural gradient algorithm. In *NIPS*.
Lee, S.-I., Ganapathi, V., & Koller, D. (2007). Efficient structure learning of Markov networks using L1-regularization. In *NIPS*.
Lowd, D., & Domingos, P. (2007). Efficient weight learning for Markov logic networks. In *Proceedings of the eleventh European conference on principles and practice of knowledge discovery in databases*.
Margot, F. (2010). Symmetry in integer linear programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, & L. A. Wolsey (Eds.), *50 years of integer programming 1958–2008: from the early years to the state-of-the-art* (pp. 1–40). Berlin: Springer.
Mihalkova, L., Huynh, T. N., & Mooney, R. J. (2007). Mapping and revising Markov logic networks for transfer learning. In *AAAI* (pp. 608–614).
Milch, B., & Russell, S. J. (2006). General-purpose mcmc inference over relational structures. In *Proc. of the 22nd conf. in uncertainty in artificial intelligence (UAI-2006)*.
Milch, B., Zettlemoyer, L., Kersting, K., Haimes, M., & Pack Kaelbling, L. (2008). Lifted probabilistic inference with counting formulas. In *Proc. of the 23rd AAAI conf. on artificial intelligence (AAAI-08)* (pp. 13–17).
Mladenov, M., Ahmadi, B., & Kersting, K. (2012). Lifted linear programming. In *JMLR: W&CP: Vol. 22. 15th int. conf. on artificial intelligence and statistics (AISTATS 2012)* (pp. 788–797).
Møller, M. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, *6*(4), 525–533.
Murphy, K. P., & Weiss, Y. (2001). The factored frontier algorithm for approximate inference in DBNs. In *Proc. of the conf. on uncertainty in artificial intelligence (UAI-01)* (pp. 378–385).
Murphy, K. P., Weiss, Y., & Jordan, M. I. (1999). Loopy belief propagation for approximate inference: an empirical study. In *Proc. of the conf. on uncertainty in artificial intelligence (UAI-99)* (pp. 467–475).
Natarajan, S., Tadepalli, P., Dietterich, T. G., & Fern, A. (2009). Learning first-order probabilistic models with combining rules. *Annals of Mathematics and AI*.
Natarajan, S., Khot, T., Kersting, K., Guttmann, B., & Shavlik, J. (2012). Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*.

Nath, A., & Domingos, P. (2010). Efficient lifting for online probabilistic inference. In *AAAI*.
Niepert, M. (2012). Markov chains on orbits of permutation groups. In *Proc. of the 28th conf. on uncertainty in artificial intelligence (UAI)*.
Niu, F., Ré, C., Doan, A., & Shavlik, J. (2011). Tuffy: scaling up statistical inference in Markov logic networks using an rdbms. *Proceedings of the VLDB Endowment*, *4*(6), 373–384.
Pearl, J. (1991). *Reasoning in intelligent systems: networks of plausible inference* (2nd ed.). San Meteo: Morgan Kaufmann.
Poole, D. (2003). First-order probabilistic inference. In *Proc. of the 18th international joint conference on artificial intelligence (IJCAI-05)* (pp. 985–991).
Poole, D., Bacchus, F., & Kisyński, J. (2011). Towards completely lifted search-based probabilistic inference. In *CoRR*. 1107.4035.
Poon, H., & Domingos, P. (2008). Joint unsupervised coreference resolution with Markov logic. In *Proceedings of the conference on empirical methods in natural language processing*.
Ravindran, B., & Barto, A. G. (2001). *Symmetries and model minimization in Markov decision processes* (Technical Report 01-43). University of Massachusetts, Amherst, MA, USA.
Richards, B. L., & Mooney, R. J. (1992). Learning relations by pathfinding. In *AAAI*.
Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning*, *62*(1–2), 107–136.
Rosenblatt, F. (1962). *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. New York: Spartan.
Russell, S. J., & Norvig, P. (2003). *Artificial intelligence: a modern approach*. Upper Saddle River: Pearson Education.
Sato, T., & Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *The Journal of Artificial Intelligence Research*, *15*, 391–454.
Schraudolph, N., & Graepel, T. (2003). Combining conjugate direction methods with stochastic approximation of gradients. In *AISTATS* (pp. 7–13).
Sellmann, M., & Van Hentenryck, P. (2005). Structural symmetry breaking. In *Proc. of 19th international joint conf. on artificial intelligence (IJCAI-05)*.
Sen, P., Deshpande, A., & Getoor, L. (2008). Exploiting shared correlations in probabilistic databases. In *Proc. of the intern. conf. on very large data bases (VLDB-08)*.
Singla, P., & Domingos, P. (2008). Lifted first-order belief propagation. In *Proc. of the 23rd AAAI conf. on artificial intelligence (AAAI-08)*, Chicago, IL, USA, July 13–17, 2008 (pp. 1094–1099).
Sutton, C., & Mccallum, A. (2009). Piecewise training for structured prediction. *Machine Learning*, *77*(2–3), 165–194.
Sutton, C., & McCallum, A. (2009). Piecewise training for structured prediction. *Machine Learning*, *77*(2–3), 165–194.
Taghipour, N., Fierens, D., Davis, J., & Blockeel, H. (2012). Lifted variable elimination with arbitrary constraints. In *JMLR: workshop and conference proceedings* (Vol. 22, pp. 1194–1202).
Thon, I., Landwehr, N., & De Raedt, L. (2011). Stochastic relational processes: efficient inference and applications. *Machine Learning*, *82*(2), 239–272.
Van den Broeck, G., & Davis, J. (2012). Conditioning in first-order knowledge compilation and lifted probabilistic inference. In *Proc. of the 26th AAAI conf. on artificial intelligence (AAAI-2012)*.
Van den Broeck, G., Taghipour, N., Meert, W., Davis, J., & De Raedt, L. (2011). Lifted probabilistic inference by first-order knowledge compilation. In *Proc. of the 22nd int. joint conf. on artificial intelligence (IJCAI)* (pp. 2178–2185).
Van den Broeck, G., Choi, A., & Darwiche, A. (2012). Lifted relax, compensate and then recover: from approximate to exact lifted probabilistic inference. In *UAI* (pp. 131–141).
Vishwanathan, S. V. N., Schraudolph, N. N., Schmidt, M. W., & Murphy, K. P. (2006). Accelerated training of conditional random fields with stochastic gradient methods. In *ICML* (pp. 969–976).
Wainwright, M., Jaakkola, T., & Willsky, A. (2002). A new class of upper bounds on the log partition function. In *UAI* (pp. 536–543).
Weyl, H. (1952). *Symmetry*. Princeton: Princeton University Press.
Winkler, G. (1995). *Image analysis, random fields and dynamic Monte Carlo methods*. Berlin: Springer.
Zaidel, D. W., & Hessamian, M. (2010). Asymmetry and symmetry in the beauty of human faces. *Symmetry*, *2*, 136–149.
Zettlemoyer, L.S., Pasula, H. M., & Kaelbling, L.P. (2007). Logical particle filtering. In *Proc. of the dagstuhl seminar on probabilistic, logical, and relational learning*.
Zhu, S., Xiao, Z., Chen, H., Chen, R., Zhang, W., & Zang, B. (2009). Evaluating splash-2 applications using mapreduce. In *Proceedings of the 8th international symposium on advanced parallel processing technologies, APPT '09* (pp. 452–464). Berlin: Springer.
Zinkevich, M. A., Smola, A., Weimer, M., & Li, L. (2010). Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems* (Vol. 23, pp. 2595–2603).