Lifted Generative Parameter Learning

Guy Van den Broeck* and **Wannes Meert*** and **Jesse Davis**
Department of Computer Science, KU Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
{guy.vandenbroeck,wannes.meert,jesse.davis}@cs.kuleuven.be

Abstract

Statistical relational learning (SRL) augments probabilistic models with relational representations and facilitates reasoning over sets of objects. When learning the probabilistic parameters for SRL models, however, one often resorts to reasoning over individual objects. To address this challenge, we compile a Markov logic network into a compact and efficient first-order data structure and use weighted first-order model counting to *exactly* optimize the likelihood of the parameters in a lifted manner. By exploiting the relational structure in the model, it is possible to learn more accurate parameters and dramatically improve the run time of the likelihood calculation. This allows us to calculate the exact likelihood for models where previously only approximate inference was feasible. Results on real-world data sets show that this approach learns more accurate models.

Introduction

Statistical relational learning (SRL) (Getoor and Taskar 2007) seeks to develop representations that combine the benefits of probabilistic models, such as Markov or Bayesian networks, with those of relational representations, like first-order logic. Markov logic networks (MLNs), which combine first-order logic with Markov networks, are one of the most widely used SRL formalisms (Richardson and Domingos 2006). MLNs use a variant of first-order logic that attaches a weight to each formula in a theory. Then, given a set of objects, the theory specifies how to construct a propositional Markov network. In essense, Markov logic provides a language for compactly describing very large propositional Markov networks.

This paper addresses the problem of learning the weight associated with each formula in a MLN theory from data. A natural objective function is to learn weights that maximize the training set likelihood. Since, in general, this cannot be done in closed form, weight learning is addressed via convex optimization. However, each iteration of the optimization involves the (often) intractable task of running inference over the current model to compute the likelihood and its gradient. Consequently, people often optimize an approximate objective function such as pseudolikelihood (Koller and Friedman 2009).

*Both authors contributed equally to the paper.

Lifted inference improves the efficiency of inference by exploiting symmetries in SRL models to avoid repeated computations (Poole 2003; de Salvo Braz, Amir, and Roth 2005; Jaimovich, Meshi, and Friedman 2007; Milch et al. 2008; Gogate and Domingos 2011; Van den Broeck et al. 2011). In this paper, we investigate using lifted inference in order to efficiently and exactly learn maximum likelihood weights. First, we provide a generic overview of how lifted inference can be integrated into weight learning. A key insight from lifting is the possibility to identify indistinguishable groups of objects, which can be reasoned about as a whole. Grouping together indistinguishable objects can significantly reduce the number of inferences needed to compute gradients. Furthermore, lifting can compute gradients and likelihoods in time polynomial in the size of the data, as opposed to exponential when using classical methods.

Second, we describe a concrete weight learning algorithm for Markov logic based on weighted first-order model counting (WFOMC) (Van den Broeck et al. 2011; Van den Broeck 2011). WFOMC is a state-of-the-art inference algorithm that compiles the MLN into a circuit language which permits inference in polynomial time in the domain size of the MLN. Its most appealing property for weight learning is that compilation only needs to be performed once. On each iteration of convex optimization, the circuit can be reparametrized with updated weights to compute a new likelihood and its gradient.

We evaluate our approach on three standard real-world SRL data sets. We find that our algorithm learns models with better test-set likelihood than two competing approaches. We can employ *exact* lifted inference, because the generative setting does not require conditioning on evidence, which is #P-hard (Van den Broeck and Davis 2012), as it breaks the symmetries in the model. This is an exciting new application of exact lifted inference algorithms to *real-world* data.

Background: Markov Logic

Representation

Markov networks are undirected probabilistic graphical models that represent a joint probability distribution over a set of random variables $X_1, \ldots X_n$ (Della Pietra, Della Pietra, and Lafferty 1997). Each clique of variables $\mathbf{X}_k$ in the graph has a potential function, $\phi_k(\mathbf{X}_k)$, associated with

it. The probability of a possible world ω represented by a Markov network is $\Pr(\omega) = \frac{1}{Z}\prod_k \phi_k(\omega_k)$, where ω_k is the state of the kth clique (i.e., the state of the variables that appear in that clique), and Z is a normalization constant. Markov networks are often conveniently represented as *log-linear models*, where clique potentials are replaced by an exponentiated weighted sum of features of the state: $\Pr(\omega) = \frac{1}{Z}\exp\left(\sum_i w_i f_i(\omega)\right)$. A feature f_j may be any real-valued function of the state.

For the relational representation we use function-free first-order logic which consists of three types of symbols: (uppercase) *constants*, (lowercase) *variables*, and *predicates*. Constant symbols represent objects in the domain (e.g., people: `Alice`, `Bob`, etc.). Variable symbols range over the objects in the domain. Predicate symbols represent relations among objects in the domain (e.g., `Friends`) or attributes of objects (e.g., `Smokes`). A *term* is a variable or a constant. A predicate applied to a tuple of terms is an *atom*. A *literal* is an atom or its negation. A *formula* is constructed by connecting literals using logical connectives. Following the convention for lifted inference, we assume that all variables are free, that is, formulas have no quantifiers. A *ground* atom (*formula*) is an atom (formula) that contains no variables. A *database* (i.e., a *possible world*) assigns a truth value to each possible ground atom.

Markov logic (Richardson and Domingos 2006) combines Markov networks with first-order logic. Formally, a Markov logic network (MLN) is a set of pairs, (F_i, w_i), where F_i is a first-order formula and $w_i \in \mathbb{R}$. MLNs soften logic by associating a weight with each formula. Worlds that violate formulas become less likely, but not impossible. Intuitively, as w_i increases, so does the strength of the constraint F_i imposes on the world. Formulas with infinite weights represent a pure logic formula.

MLNs provide a template for constructing Markov networks. When given a finite set of constants (the domain), the formulas from an MLN define a Markov network. Nodes in the network, representing random variables, are the ground instances of the atoms in the formulas. Edges connect literals that appear in the same ground instance of a formula. An MLN induces the following probability distribution over logical interpretations, that is, relational databases db:

$$\Pr(db) = \frac{1}{Z}\exp\left(\sum_j^{|F|} w_j n_j(db)\right) \tag{1}$$

where F is the set of formulas in the MLN, w_i is the weight of the j^{th} formula, and $n_j(db)$ is the number of true groundings of formula F_j in database db.

Weight Learning

This paper focuses on the weight learning task for MLNs (Huynh and Mooney 2009; Lowd and Domingos 2007; Singla and Domingos 2005; Richardson and Domingos 2006). Weight learning uses data to automatically learn the weight associated with each feature (formula) by optimizing a given objective function. Ideally, each candidate model would be scored by its training set (log-)likelihood. For MLNs, the likelihood is a convex function of the weights and learning can be solved via convex optimization. The derivative of the log-likelihood (Richardson and Domingos 2006) with respect to the jth feature is:

$$\frac{\partial}{\partial w_j}\log \Pr_w(db) = n_j(db) - \mathbb{E}_w[n_j] \tag{2}$$

where $n_j(db)$ is the number of true groundings of F_i in training data and $\mathbb{E}_w[n_j]$ is computed using the current weight vector. The jth component of the gradient is simply the difference between the empirical counts of the jth feature in the data and its expectation according to the current model. Thus, each iteration of weight learning must perform inference on the current model to compute the expectations. This is often computationally infeasible.

Currently, the default generative weight learning approach for MLNs is to optimize the pseudo-likelihood (Besag 1975), which is more efficient to compute. The pseudo-likelihood is defined as

$$\Pr_w^{\bullet}(\omega) = \prod_{j=1}^{V} \Pr_w(X_j = \omega_j | MB_{X_j} = MB_{X_j}(\omega)),$$

where V is the number of random variables, ω_j is the value of the jth variable, MB_{X_j} is X_j's Markov blanket and $MB_{X_j}(\omega)$ is its state in the data. This can also be optimized via convex optimization. The (pseudo-) likelihood for a set of training examples is the product of the (pseudo-) likelihoods for the individual examples.

There has been work on optimizing the *conditional* likelihood and the most advanced work is by Lowd and Domingos (2007). They propose several approaches, the best of which is a second-order method called pre-conditioned scaled conjugate gradient (PCSG). They use MC-SAT (Poon and Domingos 2006), which is a slice-sampling Markov chain Monte Carlo method, to approximate the expected counts. Generative learning is a special case of discriminative learning where the query set contains all the variables in the domain and the evidence set is empty. Therefore this approach is suitable for learning maximum likelihood weights, although, to the best of our knowledge, this has yet to be attempted until this paper.

Inference

Probabilistic logic models combine aspects of first-order logic and probabilistic graphical models, enabling them to model complex logical and probabilistic interactions between large numbers of objects (Getoor and Taskar 2007; De Raedt et al. 2008). This level of expressivity comes at the cost of increased complexity of inference, motivating a new line of research in lifted inference algorithms (Poole 2003). These algorithms exploit logical structure and symmetries in probabilistic logics to perform efficient inference in these models. Large domains lead to very large graphical models causing inference to be intractable. Recent advances in lifted inference deal with a broad class of models that now can deal with large domains. We use the definition of Van den Broeck (2011) to formally define what lifting means:

Definition 1 (Domain-Lifted Probabilistic Inference)**.** A probabilistic inference procedure is *domain-lifted* for a model M, query q and evidence e iff the inference procedure runs in *polynomial* time in $|D_1|, \ldots, |D_k|$ with D_i the domain of the logical variable v_i appearing in M, q or e.

Lifted Generative Weight Learning

In this section, we provide a general overview about how lifted inference can be used in the context of weight learning. This approach yields *two benefits*:

First Benefit Leveraging insights from the lifted inference literature allows weight learning to compute a small number of marginals to compute the gradient,

Second Benefit Each query is computed more efficiently using lifted inference. Namely, it is polynomial in the size of the databases, that is, the number of objects in the databases, whereas propositional inference is in general exponential in this size.

To illustrate the intuition behind the approach, assume that we want to learn the weight for a MLN that contains a single formula $w : F(x_1, \ldots, x_n)$. Furthermore, we have access to a lifted inference oracle that can efficiently compute the marginal probability of any random variable (ground atom) in the MLN, even for large domain sizes. Learning a weight that maximizes the likelihood of the MLN is challenging because it requires computing $\mathbb{E}_w[n_F]$ at each iteration of an optimization algorithm, by summing over the probability of each possible grounding of F:

$$\begin{aligned}\mathbb{E}_w[n_F] = &\Pr(F(x_1, \ldots, x_n)\theta_1) + \ldots \\ &+ \Pr(F(x_1, \ldots, x_n)\theta_m) \end{aligned} \tag{3}$$

where θ_i is a grounding substitution which replaces all the logical variables (x_i) in the atom by constants.

We now first review some techniques from the lifted inference literature. Second, we will show how to use these techniques to efficiently compute Equation 3.

Equiprobable Random Variables

A set of random variables V is called *equiprobable* w.r.t. a given MLN iff for all $v_1, v_2 \in V : \Pr(v_1) = \Pr(v_2)$. In the absence of evidence, many of the queries in an MLN will be equiprobable, because of the symmetries imposed by the model. Lifted inferences excels at answering these types of queries. In fact, one of the key insights from lifted inference is that we can partition the set of random variables into equiprobable sets by purely syntactic operations on the first-order model (Poole, Bacchus, and Kisynski 2011; Van den Broeck, Choi, and Darwiche 2012; Van den Broeck 2013).

Example 1. To illustrate this point, consider the model

$$w \quad \text{Smokes}(x) \wedge \text{Friends}(x, y) \Rightarrow \text{Smokes}(y).$$

which states that smokers are more likely to be friends with other smokers. Assuming a domain of three constants, $Alice$, Bob, and $Charlie$, $\text{Friends}(Alice, Bob)$ and $\text{Friends}(Bob, Charlie)$ are equiprobable, but $\text{Friends}(Alice, Alice)$ and $\text{Friends}(Bob, Charlie)$ are not. The first two queries have identical probabilities because they are indistinguishable w.r.t. the MLN. Intuitively, this can be seen by looking at a permutation of the constants, mapping $Alice$ into Bob and Bob into $Charlie$. This permutation turns $\Pr(\text{Friends}(Alice, Bob))$ into $\Pr(\text{Friends}(Bob, Charlie))$, whereas the MLN model is invariant under this permutation (it does not even explicitly mention the constants). Therefore, these atoms are indistinguishable and they must have the same marginal probability.

The question then is how to partition the queries into equiprobable sets. This can be done based on syntactic properties of the MLN by a technique called *preemptive shattering* (Poole, Bacchus, and Kisynski 2011). It is a conceptually simpler version of the influential *shattering* algorithm proposed by Poole (2003) and de Salvo Braz, Amir, and Roth (2005) in the context of lifted inference. This technique can be applied to a model with an arbitrary number of formulas. If the model does not mention specific constants, the algorithm enumerates all ways in which the logical variables in the same formula can be equal or different. More formally, in the single formula case, $\Pr(F(X_1, \ldots, X_n)\theta_k) = \Pr(F(X_1, \ldots, X_n)\theta_l))$ when two arguments $X_i\theta_k = X_j\theta_k$ iff $X_i\theta_l = X_j\theta_l$. If the model itself mentions certain unique information about specific constants, we can still partition the queries into equiprobable sets, but finding such sets gets slightly more complicated. Poole, Bacchus, and Kisynski (2011) contains the full details for this procedure.

Evaluating Expected Counts

We will now show how to use lifted inference techniques to compute the gradient.

First Benefit To achieve the *first benefit*, we identify equiprobable sets. This allows us to reduce the number of queries (Equation 3) that need to be answered during weight learning. For each set of equiprobable queries, only one representive query needs to be answered as each other query in the group will have the identical marginal probability. Answering these queries simply requires calculating the probability that each formula is true according to the model, and does not involve conditioning on the data. The only way in which the probabilities depend on the data is the domain size of each variable, that is, the number of objects in the world.

Let $\mathcal{P} = \{E_1, \ldots E_q\}$ denote the equiprobable partition found for formula $F(x_1, \ldots, x_n)$ by preemptive shattering, and let $F(x_1, \ldots, x_n)\theta_{E_i}$ be some element of E_i. We can then reduce the computation of Equation 3 to computing

$$\begin{aligned}\mathbb{E}_w[n_\phi] = &|E_1| \cdot \Pr(F(x_1, \ldots, x_n)\theta_{E_1}) + \ldots \\ &+ |E_q| \cdot \Pr(F(x_1, \ldots, x_n)\theta_{E_q}), \end{aligned} \tag{4}$$

involving as many queries as there are elements in the partition ($|\mathcal{P}| = q$).

In the multiple database setting (as arises when performing cross-validation), it is necessary to compute the gradient for each database, as the domain size and therefore the size of each E_i can vary according to the fold. The size of the partition $|\mathcal{P}|$, however, does not depend on the domain size, only on the structure of the MLN formulas.

Theorem 1. *Given b databases and an equiprobable partition $\mathcal{P}$, evaluating the gradient of the likelihood of the MLN (Equation 2) requires computing $b \cdot |\mathcal{P}|$ marginal probabilities.*

In the special case of a single formula with n logical variables, the number of queries we need to pose is the Bell number B_n (Bell 1934; Rota 1964).

Definition 2 (Bell Number)**.** Bell numbers are recursively defined as

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k \quad \text{with} \quad B_0 = B_1 = 1.$$

The Bell number B_n represents the number of partitions of n elements into non-empty sets. In our case, it is the number of equivalence relations on the n logical variables in the formula, which does not depend on the size of the domains or database. Assuming a domain size of D, that same formula will have D^n groundings and computing Equation 2 without using these insights from lifted inference would require answering D^n queries.

When more generally, we have multiple formulas that do not explicitly mention any constants from the database, the analysis is also easy, based on properties of the preemptive shattering algorithm.

Proposition 2. *A Markov logic network with k formulas $(w_1, F_1(x_1^1, \ldots, x_1^{n_1}))$ to $(w_k, F_k(x_k^1, \ldots, x_k^{n_k}))$ without constants has an equiprobable partition $\mathcal{P}$ such that $|\mathcal{P}| = \sum_{i=1}^{k} B_{n_i}$.*

For this case, Equation 3 requires computing $\sum_{i=1}^{k} D^{n_i}$ marginals per database, whereas lifted learning requires computing $\sum_{i=1}^{k} B_{n_i}$, essentially removing the dependency on the size of the database from Equation 3. This difference can be significant. Typically, formulas have a bounded number of variables n_i, on the order of between two and four, which gives Bell numbers $B_2 = 2$, $B_3 = 5$ and $B_4 = 15$. Databases in SRL, on the other hand, typically describe relations between hundreds of objects (D), resulting in models with tens of thousands of random variables. This is also true for the databases used in Section . In the most general case where the MLN being learned explicitly mentions certain constants, the analysis becomes more complex. However, the size of the partition found by preemptive shattering will still be polynomial in the number of such constants and independent of the number of constants in the entire domain and in the databases used for learning.

Second Benefit The *second benefit* of lifted learning over its proposition counterpart is the *complexity* of inference for each query. When using a lifted inference oracle that is *domain-lifted*, computing the probabilities in Equation 4 will be polynomial in the domain size, and therefore polynomial in the size of the databases. On the other hand, when doing propositional inference to compute the same numbers, inference is in general exponential in the domain size. Treewidth is a polynomial of domain size for most non-trivial MLNs and propositional inference is exponential in treewidth.

Lifted Weight Learning by First-Order Knowledge Compilation

The previous section assumed the presence of a lifted inference oracle. We will now look at the implications of choosing one particular algorithm, Weighted First-Order Model Counting (WFOMC) (Van den Broeck et al. 2011; Van den Broeck 2013). First, we give the necessary background on WFOMC and then describe its application to lifted learning.

Background on WFOMC

WFOMC's approach to lifted probabilistic inference consisting of the following three steps: (i) convert the MLN to a WFOMC problem, (ii) compile the WFOMC problem into a First-Order d-DNNF (FO d-DNNF) circuit, and (iii) evaluate the circuit for a given set of weights and domain sizes to compute probabilities.

WFOMC Representation. A WFOMC problem is similar to a Markov logic network. The difference is that in a WFOMC problem, weights can only be associated with predicates. For example, for the predicate Q, only weighted formulas of the form $(w, \mathrm{Q}(x_1, \ldots, x_n))$ are allowed. Formulas that are complex (containing logical connectives) must be *hard formulas*, with infinite weight. Any MLN can be transformed into a WFOMC problem by adding new atoms to the theory that represent the truth value of each weighted complex formula.

Example 2. Example 1 contains one weighted complex formula. Its WFOMC representation is

$$\begin{array}{ll} w & \mathrm{F}(x, y) \\ \infty & \mathrm{F}(x, y) \equiv [\mathrm{Smokes}(x) \wedge \mathrm{Friends}(x, y) \Rightarrow \mathrm{Smokes}(y)] \end{array}$$

where we introduce the additional atom $\mathrm{F}(x, y)$ to carry the weight of the MLN formula.

First-Order d-DNNF Circuits. First-order knowledge compilation compiles a first-order knowledge base into a target circuit language called FO d-DNNF (Van den Broeck et al. 2011), that represents theories in first-order logic with domain constraints. *Domain constraints* define a finite domain for each logical variable.

A FO d-DNNF circuit is a directed, acyclic graph, where the leaves represent first-order literals and the inner nodes represent formulas. A FO d-DNNF includes the following inner node types: *decomposable conjunction*, a conjunction of children that do not share any random variables, *deterministic disjunction*, a disjunction whose children cannot be true at the same time, and *first-order generalizations* of these types of operators.

Example 3. Figure 1 illustrates a FO d-DNNF for the formula of Example 1. The circuit introduces a new domain S, which is a subset of $People$. It states that there exists such a S for which (i) all people in S are smokers (ii) no other people are smokers and (iii) smokers are not friends with non smokers. Or alternatively, someone who is friends with a smoker is also a smoker.

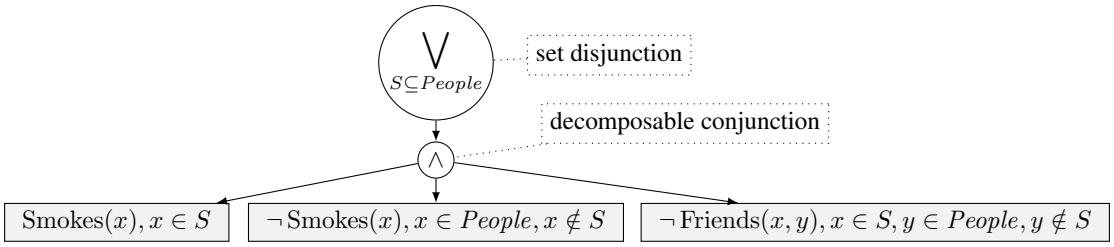

Figure 1: First-Order d-DNNF Circuit for the Formula of Example 1.

A top-down compilation algorithm transforms a logical theory into a FO d-DNNF by applying a sequence of operations that simplify the logical theory (see Van den Broeck et al. (2011) for algorithmic details). Van den Broeck (2011) proved that any theory where each formula has at most two logical variables and no quantifiers are guaranteed to be domain-liftable and can be compiled to a fully lifted circuit. Furthermore, many commonly used MLN theories outside this class can be compiled to a fully lifted circuit. In other cases, the theory can be compiled by grounding parts of the theory.[1]

Inference. The compiled circuit for a WFOMC problem can be used to answer probabilistic queries. The marginal probability of a query q for a model M, weight vector $\bar{w}$ and domain size D is

$$\mathrm{P}(q|M) = \frac{\mathrm{WMC}(q \wedge M, \bar{w}, D)}{\mathrm{WMC}(M, \bar{w}, D)} \tag{5}$$

where WMC stands for the weighted model count. We will drop the arguments w and D when they are clear from the context. The $\mathrm{WMC}(\mathrm{q} \wedge \mathrm{M})$ is simply the weight of all possible worlds where q is true. The $\mathrm{WMC}(\mathrm{M})$ is simply the partition function Z for the model. Darwiche (2009) gives a more detailed overview of the weighted model counting approach to propositional probabilistic inference. The FO d-DNNF circuit for a WFOMC problem is independent of the domain size of the WFOMC problem. Furthermore, computing weighted model counts is polynomial in the size of the domains. One of the advantages of using knowledge compilation for inference is that it exploits context-specific independence and determinism in the MLN.

Lifted Weight Learning

To illustrate the benefits of using a knowledge compilation approach, we first consider computing the gradient of the likelihood for a single formula. Then we present our full lifted weight learning algorithm.

In the single formula case, computing the expected number of groundings of $F(x_1, \ldots, x_n)$ requires estimating $\Pr(F(x_1, \ldots, x_n)\theta_{E_i})$ once for each equiprobable partition. WFOMC solves this by computing the ratio:

$$\frac{\mathrm{WMC}(F(x_1, \ldots, x_n)\theta_{E_i} \wedge M)}{\mathrm{WMC}(M)}$$

Notice that each partition has the same denominator $\mathrm{WMC}(M)$, which corresponds to the partition function of the MLN. If we have q equiprobable partitions, evaluating the weighted model counts requires compiling $q+1$ circuits: one for each equiprobable partition, and one for the partition function. These circuits are *independent* of the weights and domains of the first-order model and can therefore be used to compute Equation 4 for any database size and weight vector. Thus, each circuit can be *reused on each iteration of weight learning*. This exemplifies the idea behind the knowledge compilation approach to inference: transform the model from a representation where a certain task is hard to a representation where the task is easy, and reuse that representation to solve multiple problems.

Algorithm 1 outlines our Lifted Weight Learning (LWL) approach. It takes a MLN M and a set of databases DB as inputs and returns a weight vector $\bar{w}$. The algorithm works as follows. First, it builds all the circuits needed to compute the likelihood and its gradient. It compiles one circuit for M to compute the partition function. Then it preemptively shatters each weighted formula F in M to identify its equiprobable partition. It compiles one circuit for every equiprobable partition of the formula. Second, it runs an iterative procedure to learn the weights. During each iteration i of the convex optimization, it computes the gradient of the likelihood given the current weights $\bar{w}_i$. First it computes the partition function. Then, for each of the b databases, the expected counts for each formula are calculated by reevaluating the compiled circuit associated with every one of the formula's equiprobable partitions. Traditionally, this is the most challenging step in computing Equation 2 (the gradient). The algorithm terminates when a stop condition is met (e.g., after a predefined number of iterations).

The computational saving of employing WFOMC during inference can be characterized as follows. Over t iterations of the convex optimization algorithm, instead of answering $t \cdot b \cdot |\mathcal{P}|$ hard inference queries, as done by a vanilla lifted learning algorithm, the knowledge compilation approach performs $1+|\mathcal{P}|$ hard compilation steps and reuses each circuit $t \cdot b$ times by reevaluating it for different weight vectors w_i and domain sizes D.

For the special case of a single formula with n logical variables, by using both knowledge compilation and lifted

[1] Any other exact lifted inference technique could be used (e.g., FOVE or PTP). WFOMC is chosen because it can lift a larger variety of symmetries and offer circuit reuse which is unique to knowledge compilation.

Algorithm 1 LIFTEDWEIGHTLEARNING(M, DB)

Input.
M: A set of MLN formulas with initial weights
DB: A set of databases.

Supporting Functions.
COMPILE Compile to FO d-DNNF circuit
SHATTER Partition into equiprobable sets
WFOMC Compute weighted FO model count
LBFGS Optimization algorithm

Function.
let D^{db} be the domain sizes in database db
let n_F^{db} be the number of true groundings of formula F in database db
let $\bar{w}$ be the initial weight vector of M
$C_Z \leftarrow \text{COMPILE}(M)$
for each $F \in M$ **do**
  $\mathcal{P}_F \leftarrow \text{SHATTER}(M, F)$ // Partition
  for each $E \in \mathcal{P}_F$ **do**
    for some $F(x_1, \ldots, x_n)\theta_E \in E$ **do**
      $C_E \leftarrow \text{COMPILE}(M \wedge F(x_1, \ldots, x_n)\theta_E)$
repeat
  $\mathcal{L} \leftarrow 0$ // Log-likelihood
  $\nabla\mathcal{L} \leftarrow \bar{0}$ // Log-likelihood gradient vector
  for each $db \in DB$ **do**
    $Z \leftarrow \text{WFOMC}(C_Z, \bar{w}, D^{db})$
    $\mathcal{L} \leftarrow \mathcal{L} - \log(Z)$
    for each $F_i \in M$ **do**
      $\mathcal{L} \leftarrow \mathcal{L} + \bar{w}_i \cdot n_{F_i}^{db}$
      $\nabla\mathcal{L}_i \leftarrow \nabla\mathcal{L}_i + n_{F_i}^{db}$
      for each $E \in \mathcal{P}_{F_i}$ **do**
        $p \leftarrow \text{WFOMC}(C_E, \bar{w}, D^{db})/Z$
        $\nabla\mathcal{L}_i \leftarrow \nabla\mathcal{L}_i - |E| \cdot p$
  $\bar{w} \leftarrow \text{LBFGS}(\mathcal{L}, \bar{w}, \nabla\mathcal{L})$
until convergence
return $\bar{w}$

inference, we went from answering $t \cdot b \cdot D^n$ queries that are exponential in the size of the databases to $1 + B_n$ compilations that are independent of the training examples and $t \cdot b$ circuit evaluations that are polynomial in the size of the databases.

Empirical Evaluation

We evaluate our approach on one synthetic domain and three real-world datasets.

Synthetic Data: Scaling Behavior

We use the friends and smokers model from Example 1 (Singla and Domingos 2005) as a controlled enviroment to explore how our algorithm scales with respect to domain size. We vary the number of people in the domain from 100 to 30,000 and randomly generate a training database for each size. The results are in Fig. 2. This model is fully liftable and training time is polynomial in the domain size. The largest database, for domain size $30,000$, assigns truth values to $30,000^2 + 30,000$ or approximately 900 million ground atoms.

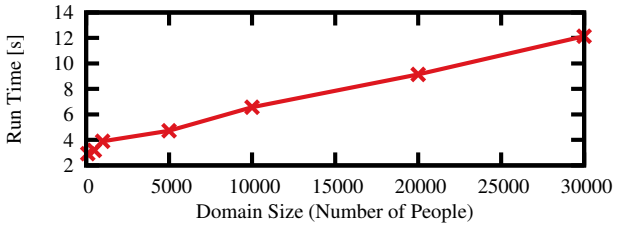

Figure 2: Timings for friends-smokers.

Real-World Data: Test-Set Likelihood

We use three real-world datasets to compare the following three MLN weight learning algorithms:

PLL optimizes the pseudo-likelihood of the MLN given the data via the LBFGS algorithm (Liu and Nocedal 1989) and is the default generative weight learning algorithm in the Alchemy package (Kok et al. 2008).

PSCG is a weight learning algorithm (Lowd and Domingos 2007) in Alchemy that uses approximate inference to estimate the gradient of the objective function. It optimizes the likelihood of the MLN when all predicates are treated as query atoms (i.e., the evidence set is empty).

LWL is our proposed approach. It uses the WFOMC package (Van den Broeck et al. 2011) to compute the likelihood and its gradient and the limited-memory BFGS optimizer in the Factorie system (McCallum, Schultz, and Singh 2009). Our implementation of LWL is available as open-source software.[2]

The goal of our empirical evaluation is to compare whether exactly optimizing the likelihood is better than optimizing the approximated likelihood. We evaluate all models based on their test set likelihood, which is a standard metric for generative learning. Additionally, we are interested in whether it is possible to apply exact lifted learning to learned theories.

Datasets and Methodology

We first briefly describe the datasets we use.[3] The **IMDB** data comes from the IMDB.com website. The data set contains information about attributes (e.g., gender) and relationships among actors, directors, and movies. The data is divided into five different folds. The **UWCSE** data contains information about the University of Washington CSE Department. The data contains information about students, professors, and classes and models relationships (e.g., TAs and Advisor) among these entities. The data consists of five folds, each one corresponding to a different group in the CSE Department. The **WebKB** data consists of Web pages

[2] http://dtai.cs.kuleuven.be/wfomc/
[3] Available on http://alchemy.cs.washington.edu

UWCSE			
	PSCG	PLL	LWL
BUSL	-1671	-2564	**-1523**
	-684	-909	**-541**
	-1702	-2871	**-1283**
	-3291	-2660	**-2565**
	-3399	-5280	**-2362**
MSL	-25745	-1519	**-1497**
	-1070	-535	**-524**
	-21936	-1213	**-1209**
	-2756	-2472	**-2471**
	-51903	**-2261**	-2274

IMDB			
	PSCG	PLL	LWL
BUSL	-379	-1190	**-377**
	-580	-1241	**-558**
	-907	-1581	-968
	-291	-660	**-284**
	-608	-539	**-267**
MSL	-1641792	-9672	**-831**
	-1690616	-12471	**-766**
	-32286	-18242	**-1307**
	-229063	-1354	**-698**
	-16250	-982	**-700**

WebKB			
	PSCG	PLL	LWL
MSL	-9453	-5860	**-5655**
	-27628	-5129	**-5047**
	-26548	-4135	**-3917**
	-18052	-4367	**-4280**

Table 1: Log-likelihoods for models learned in the UWCSE, IMDB and WebKB domains.

from the computer science departments of four universities (Craven and Slattery 2001). The data has information about words that appear on pages, labels of pages and links between Web pages. There are four folds, one for each university. We limit the number of words considered on each fold to the ten with the highest information gain with respect to a page's class.

To generate a set of models, we use two standard MLN structure learning algorithms: BUSL (Mihalkova and Mooney 2007), which works bottom-up, and MSL (Kok and Domingos 2005), which works top-down. During structure learning, we limit each clause to contain at most four literals and three variables.[4] In all domains, we perform cross-validation and hold out one fold as the test set, and use the remaining folds to learn the model. Each fold serves as a test set once. Given the learned structure, each weight learning algorithm uses the same data that produced the structure to learn weights. Then we compute the test-set likelihood for each learned model. We always use WFOMC to compute the test-set likelihood so the only difference among the three approaches is *how* the weights were learned.

Results and Discussion

Table 1 reports results for all learned models. LWL consistenly outperforms both PLL and PSCG in terms of test-set likelihood. It loses once to PLL and once to PSCG. The magnitude of differences varies. LWL does better than PLL if the pseudo-likelihood assumption is violated (e.g., long chains of interactions). For some of the learned theories this results in large differences between LWL and PLL. For models that contain mostly nearly deterministic formulas, LWL and PLL tend to have similar test-set likelihoods. For example, MSL learns a number of such formulas for the UWCSE domain, which express statements like 'all people are either students or professors'. PSCG performance is quite variable. PSCG performs very poorly for some models as MC-SAT can sometimes give very bad estimates of the gradient. This can occur if MC-SAT fails to converge (e.g., because it gets stuck in a single mode of the distribution).

LWL is able to compile all the learned theories, except for those learned by BUSL on WebKB. These theories, while theoretically compilable, are very large as they contain more than 50 learned complex, first order formulas. The compilation ran out of 25 GB of memory. In terms of run time, on average PLL takes 1 second, LWL takes 2 minutes, and PSCG takes 40 minutes. For LWL, about 25% of the time is spent on compilation.

[4]The objective function for structure learning is pseudo-likelihood.

Related Work and Conclusions

Jaimovich, Meshi, and Friedman (2007) described a formalism similar to Markov logic and proposed a form of lifted belief propagation for generative parameter learning in that language. Our work provides a much more detailed and formal treatment of the subject. Also, we looked at using *exact* inference as opposed to approximate. Our empirical results provide some initial evidence that sampling based approximations can provide sub-optimal results. Ahmadi, Kersting, and Natarajan (2012) recently proposed the use of *approximate* lifted inference, namely lifted belief propagation (Singla and Domingos 2008; Kersting, Ahmadi, and Natarajan 2009), in a stochastic gradient optimization approach to piecewise *discriminative* weight learning.

We investigated the effect of lifted inference for parameter learning in the statistical relational learning setting. Specifically, we proposed a weight learning algorithm for Markov logic based on weighted first-order model counting. Calculating the gradient of the likelihood is the crucial step in parameter learning. Applying insights from lifted inference allows us to compute the gradient exactly while querying fewer marginals and answering each query more efficiently. WFOMC yields a further benefit in that its compiled circuits are independent of the database and can be reused over all databases and iterations during optimization. Our proposed approach was evaluated on learned models from three real-world data sets. Our approach consistently resulted in better test-set likelihoods than two approximate weight learning algorithms.

Acknowledgments Guy Van den Broeck is supported by the Research Foundation-Flanders (FWO Vlaanderen).

References

Ahmadi, B.; Kersting, K.; and Natarajan, S. 2012. Lifted online training of relational models with stochastic gradient methods. In *Proceedings of ECML/PKDD*.

Bell, E. 1934. Exponential numbers. *American Mathematical Monthly* 411–419.

Besag, J. 1975. Statistical Analysis of Non-Lattice Data. *The Statistician* 24:179–195.

Craven, M., and Slattery, S. 2001. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning Journal* 43(1/2):97–119.

Darwiche, A. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

De Raedt, L.; Frasconi, P.; Kersting, K.; and Muggleton, S., eds. 2008. *Probabilistic inductive logic programming: theory and applications*. Berlin, Heidelberg: Springer-Verlag.

de Salvo Braz, R.; Amir, E.; and Roth, D. 2005. Lifted First-Order Probabilistic Inference. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 1319–1325.

Della Pietra, S.; Della Pietra, V.; and Lafferty, J. 1997. Inducing Features of Random Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19:380–392.

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

Gogate, V., and Domingos, P. 2011. Probabilistic Theorem Proving. In *Proceedings of 27th Conference on Uncertainty in Artificial Intelligence*.

Huynh, T. N., and Mooney, R. J. 2009. Max-margin weight learning for markov logic networks. In *Proceedings of ECML/PKDD*, 564–579.

Jaimovich, A.; Meshi, O.; and Friedman, N. 2007. Template based inference in symmetric relational Markov random fields. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, 191–199.

Kersting, K.; Ahmadi, B.; and Natarajan, S. 2009. Counting belief propagation. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, 277–284.

Kok, S., and Domingos, P. 2005. Learning the structure of Markov logic networks. In *Proceedings of the International Conference on Machine Learning*, 441–448.

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

Koller, D., and Friedman, N. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

Liu, D. C., and Nocedal, J. 1989. On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming* 45(3):503–528.

Lowd, D., and Domingos, P. 2007. Efficient weight learning for Markov logic networks. In *Proceedings of the 11th Conference on the Practice of Knowledge Discovery in Databases*, 200–211.

McCallum, A.; Schultz, K.; and Singh, S. 2009. FACTORIE: Probabilistic programming via imperatively defined factor graphs. In *Neural Information Processing Systems (NIPS)*.

Mihalkova, L., and Mooney, R. J. 2007. Bottom-Up Learning of Markov Logic Network Structure. In *Proceedings of the 24th International Conference on Machine Learning*, 625–632.

Milch, B.; Zettlemoyer, L. S.; Kersting, K.; Haimes, M.; and Kaelbling, L. P. 2008. Lifted Probabilistic Inference with Counting Formulas. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, 1062–1068.

Poole, D.; Bacchus, F.; and Kisynski, J. 2011. Towards completely lifted search-based probabilistic inference. *CoRR* abs/1107.4035.

Poole, D. 2003. First-Order Probabilistic Inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, 985–991.

Poon, H., and Domingos, P. 2006. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, 458.

Richardson, M., and Domingos, P. 2006. Markov Logic Networks. *Machine Learning* 62(1):107–136.

Rota, G. 1964. The number of partitions of a set. *The American Mathematical Monthly* 71(5):498–504.

Singla, P., and Domingos, P. 2005. Discriminative training of markov logic networks. In *20th National Conference on Artificial Intelligence*, 868–873.

Singla, P., and Domingos, P. 2008. Lifted First-Order Belief Propagation. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, 1094–1099.

Van den Broeck, G., and Davis, J. 2012. Conditioning in first-order knowledge compilation and lifted probabilistic inference. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*.

Van den Broeck, G.; Taghipour, N.; Meert, W.; Davis, J.; and De Raedt, L. 2011. Lifted Probabilistic Inference by First-Order Knowledge Compilation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2178–2185.

Van den Broeck, G.; Choi, A.; and Darwiche, A. 2012. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*.

Van den Broeck, G. 2011. On the Completeness of First-Order Knowledge Compilation for Lifted Probabilistic Inference. In *Annual Conference on Neural Information Processing Systems (NIPS)*.

Van den Broeck, G. 2013. *Lifted Inference and Learning in Statistical Relational Models*. Ph.D. Dissertation, KU Leuven.