Exploiting Causal Independence in Markov Logic Networks: Combining Undirected and Directed Models

Sriraam Natarajan*, Tushar Khot*, Daniel Lowd+, Prasad Tadepalli#,
Kristian Kersting&, Jude Shavlik*

* University of Wisconsin-Madison, + University of Oregon, # Oregon State University,
& Fraunhofer IAIS

Abstract. A new method is proposed for compiling causal independencies into Markov logic networks (MLNs). A MLN can be viewed as compactly representing a factorization of a joint probability into the product of a set of factors guided by logical formulas. We present a notion of causal independence that enables one to further factorize the factors into a combination of even smaller factors and consequently obtain a finer-grain factorization of the joint probability. The causal independence lets us specify the factor in terms of weighted, directed clauses and operators, such as "or", "sum" or "max", on the contribution of the variables involved in the factors, hence combining both undirected and directed knowledge. Our experimental evaluations shows that making use of the finer-grain factorization provided by causal independence can improve quality of parameter learning in MLNs.

1 Introduction

Most traditional AI methods were based on one of the two approaches: *first-order logic*, which excels at capturing the rich relationships among many objects, or *statistical* representations, which handle uncertain environments and noisy observations. Statistical relational learning (SRL) [5], an area of growing interest, seeks to unify these approaches in order to handle problems that are both complex and uncertain.

The principal attraction of SRL models is that they are more succinct than their propositional counterparts, leading to easier specification of their structure by the domain experts and faster learning of their parameters. However, different proposed models are good at expressing different kinds of knowledge, making it difficult to compare their empirical performance and to simultaneously exploit the strengths of each. The largest divide is between directed and undirected representations.

One of the primary advantages of the directed graphical models is the notion of "Independence of Causal Influence" (ICI) [6, 14] a.k.a "causal independence," i.e., there may be multiple independent causes for a target variable. Directed models can learn conditional distributions due to each of the causes separately and combine them using a (possibly stochastic) function, thus making the process of learning easier. This notion of ICI has been extended to the directed SRL models in two different ways: while PRMs [3] use aggregators such as `max`, `min`, and `average` to combine the influences due to several parents, other formalisms such as BLPs [10] and RBNs [8] use combination functions such as Noisy-OR, mean, or weighted mean to combine distributions.

One problem with the directed models is the need to keep the graph acyclic while preserving sparsity. This problem is avoided by undirected models such as Markov logic networks (MLNs) [1], which are based on Markov networks. Undirected models do not consider local models (i.e, do not treat each cause as independent from others) and hence do not model the notion of ICI explicitly. Bayesian Networks with tabular CPDs (one parameter for each configuration of the parent variables) can be directly translated to MLNs by introducing one formula for each BN parameter[1] . What is less clear is how combination functions from relational models can best be represented in an MLN. We consider a subset of combination functions called *decomposable combining rules* and derive a representation of MLNs that captures these rules. The important aspect of this representation is that we do not use the "ground" Bayesian network and instead use a "lifted" representation that avoids the grounding of the clauses, since grounding can produce an exponentially large number of (variable-free) clauses.

Representing combining rules using MLNs is a key step towards unifying directed and undirected SRL approaches. Such a unified view on SRL is not only of theoretical interest – It actually has many important practical implications such as more natural model specification and development of specialized, highly efficient inference and learning techniques that can be applied differently to different pieces of the model.

In this work, we make several major contributions: (1) A provable linear representation of decomposable combining functions within MLNs; (2) Explicit examples of average-based and noisy combination functions. (3) A formal description of the algorithm for converting from directed models with combining rules to MLNs. (4) A macro-definition that allows for succinct specification of the resulting MLNs. (5) Empirical proof that combining rules can improve the learning of MLNs when the domain knowledge available is minimal.

We proceed as follows. After introducing the necessary background, we derive the MLN clauses for representing decomposable combining rules and provide the clauses for two common cases of combining rules. Next, we derive a bound on the number of clauses and provide the pseudo-code for the compilation. Before the conclusion, we present empirical results in real-world tasks.

2 MLNs and Directed Models

A Bayesian network (BN) compactly represents a joint probability distribution over a set of variables $X = \{X_1, \ldots, X_n\}$ as a directed, acyclic graph and a set of conditional probability distributions (CPDs). The graph contains one node for each variable, and encodes the assertion that each variable is independent of its non-descendants given its parents in the graph. These conditional independence assertions allow us to represent the joint probability distribution as the product of the conditional probability of each variable, X_i, given its parents, parents(X_i): $P(X) = \prod_i P(X_i|\text{parents}(X_i))$

A Markov network (MN) (also called a Markov random field) specifies independencies using an undirected graph. The graph encodes the assertion that each variable is independent of all others given its neighbors in the graph. This set of independencies guarantees that the probability distribution can be factored into a set of potentials

[1] `http://alchemy.cs.washington.edu/faq/index.html`

functions defined over cliques in the graph. Unlike BNs, these factors are not constrained to be conditional probabilities. Instead, a potential function is allowed to take on any non-negative value. The joint probability distribution is therefore defined as follows:$P(X = x) = \frac{1}{Z} \prod_j \phi_j(\mathbf{D}_j)$, where ϕ_j is the jth potential function, $\mathbf{D}_j$ is the set of variables over which ϕ_j is defined, and Z is a normalization constant. MNs are often written as log-linear models, where the potential functions are replaced by a set of weighted features.

One of the most popular and general SRL representations is *Markov logic networks (MLNs)* [1]. An MLN consists of a set of formulas in first-order logic and their real-valued weights, $\{(w_i, f_i)\}$. Together with a set of constants, we can instantiate an MLN as a Markov network with a node for each ground predicate (atom) and a feature for each ground formula. All groundings of the same formula are assigned the same weight, leading to the following joint probability distribution over all atoms:$P(X = x) = \frac{1}{Z}$ $\exp\left(\sum_i w_i n_i(x)\right)$, where $n_i(x)$ is the number of times the ith formula is satisfied by possible world x and Z is a normalization constant (as in Markov networks). Intuitively, a possible world where formula f_i is true one more time than different possible world is e^{w_i} times as probable, all other things being equal.

Directed Models with Combining Rules: Our work does not assume any representation for the directed models. We merely use an abstract syntax called as First-Order Conditional Influence (FOCI) statements [13] to present the semantics of the directed models. We had earlier used this syntax to derive learning algorithms [13] and showed how most directed models such as BLPs [10], RBNs[7], PRMs[3], probabilistic relational language [4] and Logical Bayes nets [2] can be represented using this syntax. The goal of this work is not to convert from FOCI statements to MLNs but to show that the knowledge captured by directed models can be represented using MLNs. FOCI statements merely facilitate this conversion. We could replace the FOCI statements with any of the above directed models and still get the same result as all these models share the common syntax as shown in [13].

Each statement has the form: *If* $\langle condition \rangle$ *then* $\langle qualitative$ *influence*$\rangle$, where *condition* is a set of literals, each literal being a predicate symbol applied to the appropriate number of variables. The set of literals is treated as a conjunction. A $\langle qualitative$ *influence*$\rangle$ is of the form $X_1, \ldots, X_k$ $Qinf$ Y, where the X_i and Y are of the form $V.a$, and V is a variable that occurs in *condition* and a is an object attribute. Associated with each statement is a *conditional probability distribution* that specifies a probability distribution of the resultant conditioned on the influents, e.g. $P(Y|X_1, \ldots, X_k)$ for the above statement.

```
CR2{
  CR1: If {student(S), course(C), takes(T,S,C)} then
    T.grade Qinf S.satisfaction.
  CR1: If {student(S),paper(P,S)} then
    P.quality Qinf S.satisfaction.
}
```

The first rule specifies that the grade that a student obtains in a course influences his/her satisfaction. The CPD $P(S.\text{satisfaction} \mid T.\text{grade})$ associated with the first statement (partially) captures the quantitative relationships between the attributes. The second

states that if the student has authored a paper, then its quality influences the satisfaction of the student. The distributions due to multiple instantiations of the respective rules (the different course grades or the different paper qualities) are combined using the CR1 combining rule and the distributions due to different rules are combined using CR2 combining rule. We assume discrete values and hence the CPDs are represented using conditional probability tables (CPTs).

Note that there are two levels of combination functions - one for combining multiple instances of the same rule and the other for combining different rules. This idea of two-level combining rules is sufficient to capture the notion of ICI in SRL models and hence we address the 2-level combining rule in this work. The use of combining rules make learning in directed SRL models easier: multiple instances of the same rule share the same CPT and hence can be treated as individual examples while learning the CPTs. Similarly, the different CPTs can be learned *independently* of each other thus exploiting the notion of causal independence. Yet another advantage of the combining rules is that they allow for richer combination of probability distributions. MLNs in their default representation use an exponentiated weighted count as an (indirect) combination function of the different clauses. To express complex functions, a straightforward method would be to construct the grounded Bayes net for each rule and then construct the equivalent Markov net. Unfortunately, this leads to an exponential number of clauses in the MLN, making the twin problems of learning and inference computationally expensive. Instead we resort to a "lifted" method that avoids unrolling (grounding) all the clauses to create the MLN.

3 Combination Functions using MLNs

In this section, we present the equivalent MLN representation for decomposable combining functions.

3.1 Decomposable Combining Functions

In this work, we extend the definition of decomposable causal independence due to [6] to the relational setting. To understand the notion of decomposable combining rules, consider Figure 1 where y is the target and x's are the influents. The t_i^j's are the temporary y-values due to each instantiation of x_i^j. Note that there are n rules to predict y. The y's are deterministic nodes obtained using functions f's for the first level (that combines instances of the same rule) and g's for the second level (that combines different rules). The observed nodes are shown using solid circles while the dotted circles correspond to the hidden nodes. These hidden nodes are created when the functions are applied successively. In the figure, we present two levels of combining rules (f_i and g). Let us consider the first level combining rule. For simplicity, consider the set of functions f_1^i. The result y_1 can be represented as:

$$y_1 = f_{1,\sigma}^m(t_{1,\sigma}^m, f_{1,\sigma}^{m-1}(t_{1,\sigma}^{m-1}, ...f_{1,\sigma}^1(t_{1,\sigma}^1, p))) \tag{1}$$

for the given ordering σ of the different x_1^i. p is some prior on the value of y for rule j. Equation 1 can be written as

$$y_1 = f_{1,\sigma}(t_{1,\sigma}^m, t_{1,\sigma}^{m-1}, ...t_{1,\sigma}^1, p) \tag{2}$$

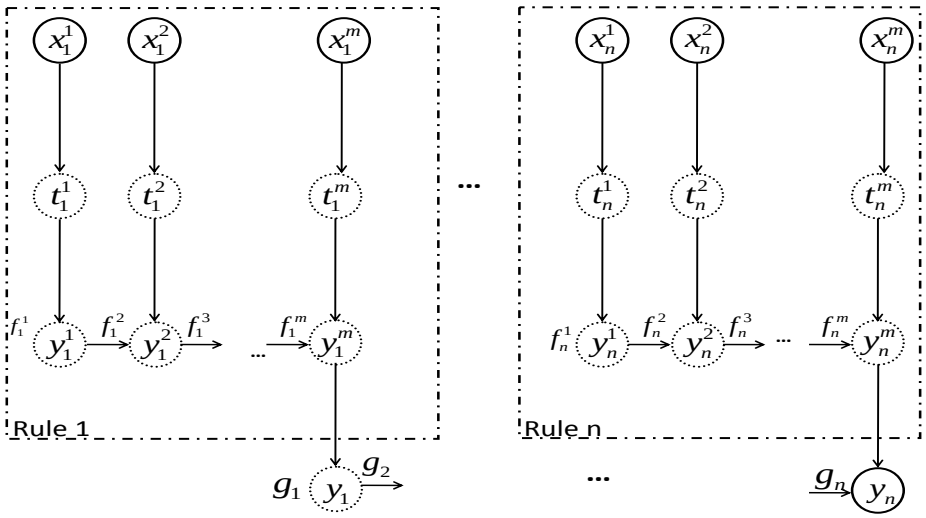

Fig. 1. Decomposable Combining Rules

where f_1 is a combining rule operating over all t_1^i.

Definition: A combining rule is called as *decomposable* if it satisfies equation 2 for **all** orderings (i.e., $\forall\sigma$). This is to say that for every possible ordering of the inputs, the combining rule can be decomposed into a set of functions that yield the same distribution.

In our setting, we require that the combining rules at both the levels are *decomposable*. i.e.,

$$y = g_{n,\sigma}(y_{n,\sigma}, g_{n-1,\sigma}(y_{n-1,\sigma}, ...g_{1,\sigma}(y_{1,\sigma}, p)))\forall\sigma \tag{3}$$

The above condition specifies that the rules can themselves be ordered differently but the resulting distribution always remains the same. Note that most common combination functions used in the literature [12, 13, 8, 6, 14] such as: *Noisy-Or*, *Noisy-And*, *Noisy-Existential*, average-based combination functions such as *mean*, *weighted-mean* and *context specific Independence*(CSI) can be represented using the above definition. In this work, we show how multi-level combining rules (rules that combine instances of the same clause and the ones that combine the distributions due to different clauses) can be represented and learned using MLNs.

3.2 Decomposable Combining Functions using MLNs

The notion of decomposability is crucial to deriving the representation of combining rules using MLNs. This allows us to consider the combining rules as multiplexers (the f_i^j and g_i in the figure above) on Bayesian networks. *The key idea in our work is to view the combination function as choosing a value among several values proposed by the parents.* For instance, taking the average of distributions corresponds to choosing the target value using an uniform distribution among the values proposed by the parents. Weighted mean can be understood as choosing a value based on the distribution given by the weights. The Bayes net representation is used only for the sake of presentation

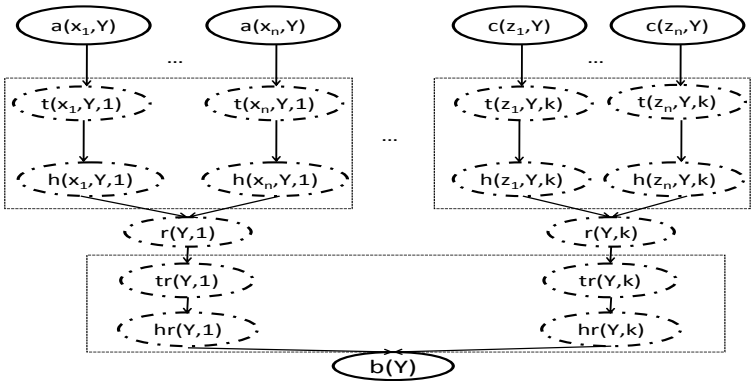

Fig. 2. Understanding combining rules using multiplexers. The dashed nodes are the hidden nodes and the multiplexer nodes, while the solid nodes are observed in the data.

and is not used in its full form during translation. The translation occurs at the logical level (i.e., on the variables rather than the groundings of these variables).

Consider the following two FOCI statements:

$$a(X,Y) \;\; Qinf \;\; b(Y)$$
$$c(Z,Y) \;\; Qinf \;\; b(Y)$$

where $\langle a, b, c\rangle$ are predicates and $\langle X, Y, Z\rangle$ are variables. Associated with each clause is a conditional probability distribution $P(b(y)|parent(b))$ where the parent for the first statement is $a(x, y)$ and the second is $c(z, y)$. Note that there could be several possible instantiations for X and Z in the above rules. For simplicity, let us assume that the distributions due to the different instances of the same rule are combined using CR_1 and the resulting distributions due to the different rules are combined using CR_2.

Consider the Bayesian network presented in Figure 2. For ease of explanation, assume that there are n instantiations of each rule and k such rules (we present only two of them for brevity). In addition to the a, b and c predicates, we introduce two more types of predicates indicated using dashed nodes: *hidden* (temporary) value predicates (t and tr) and *multiplexer* predicates (h and hr). Since there are two levels of combining functions, there are two different sets of multiplexers and hidden nodes represented by two different boxes in the figure. The first box corresponds to choosing a value from a single rule (given by $r(y, i)$, where i is the rule index) and in the next level the final value of the target is chosen from one among the different r-values. We now explain the multiplexers inside the same rule (the top box) and the same idea is extended for different rules (bottom box).

The hidden predicates t's can be understood as choosing a value of the target given the instantiation of the parent based on the CPD. The multiplexers (h-nodes) serve to choose one of the n t-values for the target. The idea is that if a particular h is activated, the value of the corresponding t node is chosen to be the value of the target for the current rule (i.e, $r(y, i)$ is set to be that particular t-value). Given the different values of $r(y, I)$ for all I, the final value of the target b is chosen using the next level of the multiplexer.

In our formalism, there is no restriction on the equality of CR_1 and CR_2, i.e., they need not be similar combination functions as long as they are decomposable. For instance, it is possible to use a mean combining rule to combine the instances of a single rule while a Noisy-Or could be used to combine the different rules themselves. It can be easily observed from our translation to MLNs (presented later) that the only change for the different cases would be the encoding of the multiplexers. Note that it is possible to imagine developing specialized clauses for each combination of the combining functions. In this work, we aim to derive a general representation that covers all decomposable combining functions. Our translation consists for four different kinds of clauses:

1. **CPT Clauses:** This follows the standard translation of Bayesian Networks to MLNs. Each independent parameter in the CPT of the Bayes net becomes a clause in the MLN. An example of such a clause is

$$\begin{aligned} w_i^1 : a(X,Y) &\Rightarrow t(X,Y,i) \\ w_i^0 : \neg a(X,Y) &\Rightarrow t(X,Y,i) \end{aligned} \tag{4}$$

   where $w_i^j = log\frac{p_i^j}{(1-p_i^j)}$, $p_i^j = P(b(Y) = 1|a(X,Y) = j)$[2]. Hence, for each independent parameter of the original CPT in the directed model, there is a clause in the MLN with the weight as a function of the parameter. In general, the set of arguments in the temporary predicate t is the union of all the arguments in the body of the clause and an argument for the rule index.
2. **Multiplexer Clauses:** These are the clauses that choose a particular value of the target given a set of parent values. For the first-level multiplexer (h in the figure), this set corresponds to the set of values due to different instantiations of the same rule. For the second-level multiplexer, this set consists of the values due to different rules. For the first level, the MLN clauses are of the form

$$\infty : h(X,Y,I) \Rightarrow (t(X,Y,I) \Leftrightarrow r(Y,I)) \tag{5}$$

   The above clause is a hard clause (i.e., infinite weight) that specifies that for a particular value of X, if $h(X,Y,i)$ is true for a rule i, then the value of the target for that rule ($r(Y,i)$) must be chosen to be the corresponding $t(X,Y,i)$. Note that the multiplexer always has the same number of variables as that of t. Similarly, for the next level, the multiplexer clause would be,

$$\infty : hr(Y,I) \Rightarrow (tr(Y,I) \Leftrightarrow b(Y)) \tag{6}$$

3. **Stochastic Function Clauses:** These are the clauses that specify the stochastic function to be employed on the values. These are essentially the "prior" on the h predicates. For mean, the idea is to choose a target value from the set of h-values uniformly. In the case of Noisy-Or, the target is chosen from using an Or function over the hidden variables (t and tr). We present the stochastic function clauses for two different cases later in the section.

[2] The CPT clauses are defined for rule 1 that uses predicate a. All the other rules will have similar clauses.

4. **Integrity Constraints:** These are the constraints that are used to specify that among the different multiplexer nodes, only **one** of them can be true for any particular example. These are of the form:

$$\begin{aligned} &\infty : h(X_1, Y, I) \wedge h(X_2, Y, I) \Rightarrow (X_1 = X_2) \\ &\infty : \qquad\qquad\qquad \exists X.h(X, Y, I) \end{aligned} \tag{7}$$

The above set of clauses specifies that if h is true for 2 values of X, they should be identical and there exists a grounding of X to make h-true. These constraints are exactly similar for the second level as well[3].

4 Transformation of Combining Rules

As mentioned earlier, the Bayesian Network representation is used **only** to explain the translation by the use of multiplexers. The translation itself is independent of the number of groundings (note that all the predicates in the clauses are variablized and not grounded). We now present two most common types of combination functions from literature: (1) *average*-based and (2) *Noisy* combination functions. Let us consider just a single clause $a(X, Y) \Rightarrow b(Y)$ for ease of explanation. Associated with this clause is a conditional probability distribution $P(b|a)$ (we use a and b as shorthand notations for the predicates). As we mentioned, the differences between different combination functions lie mainly in the stochastic function clauses. For each case, we first present the translation and prove the correctness of the resulting distribution. We then present the worked example corresponding to the student satisfaction rules presented earlier.

4.1 Average-Based Combining Rules:

Assume that the different instantiations of the above rule are combined using the weighted-mean combining rule. Then the posterior over the target b given the different sets of parents is given by

$$P(b|a_1, ..., a_n) = \frac{1}{\sum w_i} \sum_i w_i \times P(b|a_i) \tag{8}$$

where a_i denotes $a(x_i, y)$. For the case of mean, all $w_i = 1$. The CPT clauses will be of the form presented in the earlier section, where the weights are log functions of the CPT parameters ($log \frac{p_i}{(1-p_i)}$) . The multiplexer clause is again a hard clause that specifies the value of the target based on the value of the multiplexer ($h(X, Y, I)$). The integrity constraints are also the same as the ones presented above. The stochastic function is the weighted-mean. This specifies the prior on the multiplexer nodes i.e., defines the prior probability with which each multiplexer node is true. Hence, they are of the form: $u_i : h(x_i, y, i)$, where $u_i = log(w_i)$ is the log-odds of the given x_i. Actually, any weight of the form $u_i = log(const * w_i) = log(const) + log(w_i)$ would work. For mean, the

[3] We realize that we could implement these constraints using "!" in alchemy. We found some issues when we used ! for learning weights and hence explicitly present the constraints.

log-odds would imply $u_i = log(1/n)$, where n is the number of instantiations. From the previous equation, it follows that any u_i are acceptable as long as they are constant for all i. The intuition is that each $t(X, Y)$ chooses the value of the target based on the CPT, and the final value of the target is chosen from the different t's using the multiplexer nodes. The multiplexer is activated such that it takes only one value given by the stochastic function (mean or weighted-mean).

Proposition:The given representation of MLNs exactly captures the distribution given by equation 8.

Proof Sketch: This can be proved mathematically. For simplicity, consider only 2 instantiations of the rule presented above and we are interested in $P(b|a_1, a_2)$ which is given by equation 8 for $i = 2$. There will correspondingly be 4 different cases: both t_1 and t_2 (hidden variables) are true and one of h_1 or h_2 (multiplexers) is true (2 cases) and 2 cases where only the multiplexer h_i and the corresponding t_i are true. i.e.,

$$P(b|a_1, a_2) =$$
$$P(b, t_1 = 1, t_2 = 1, h_1 = 1, h_2 = 0|a_1, a_2) + P(b, t_1 = 1, t_2 = 1, h_1 = 0, h_2 = 1|a_1, a_2) +$$
$$P(b, t_1 = 1, t_2 = 0, h_1 = 1, h_2 = 0|a_1, a_2) + P(b, t_1 = 0, t_2 = 1, h_1 = 0, h_2 = 1|a_1, a_2)$$

$$= \frac{1}{Z}\left(e^{\theta_1+\theta_2+log(w_1)} + e^{\theta_1+\theta_2+log(w_2)} + e^{\theta_1+log(w_1)} + e^{\theta_2+log(w_2)}\right), \text{where } \theta_i = log\frac{p_i}{1-p_i}$$
$$= \frac{1}{Z}\frac{w_1p_1+w_2p_2}{(1-p_1)(1-p_2)} = \frac{w_1p_1+w_2p_2}{w_1+w_2}$$

where Z can be shown to be $\frac{w_1+w_2}{(1-p_1)(1-p_2)}$ by summing over the two values of b i.e., over $(0, 1)$. We omit the calculation of Z for brevity. Thus we can show that the final distribution due to these MLNs is equal to the distribution presented in equation 8. The same proof can be extended for multi-level combining rules as well.

Worked Example: Consider the FOCI statements about student satisfaction presented earlier. We now present the case where $CR1$ is mean while $CR2$ is weighted-mean. We show the translation to MLNs below. First, consider the CPT clauses. Since the grade of the student can be any of say, $\langle A, B, C, D, F\rangle$, we use $+G$ in Alchemy that uses all possible groundings of G. The CPT clauses for each rule are as follows:

$$w_G : student(S), course(C), takes(T, S, C), grade(T, +G) \Rightarrow t1(S, T, C, +G)$$
$$w_Q : student(S), paper(P, S), quality(P, +Q) \Rightarrow t2(S, P, +Q)$$

where each w_G and w_Q are the log-odds for each grade (G) and quality (Q) respectively. We refer to the Alchemy manual for a detailed discussion on $+$. For the purposes of this paper, it suffices to say that for each *grade, student* and *course* combination, there will be a clause corresponding to the CPT entry. Next, we present the multiplexer clauses.

$$\infty : h1(S, T, C, G) \Rightarrow t1(S, T, C, G) \Leftrightarrow r(S, 1)$$
$$\infty : h2(S, P, Q) \Rightarrow t2(S, P, Q) \Leftrightarrow r(S, 2)$$
$$\infty : hr(S, R) \Rightarrow r(S, R) \Leftrightarrow satisfaction(S) \quad (9)$$

The first two clauses serve to choose the intermediate values of r corresponding to rules 1 and 2. The third rule then chooses the final value of satisfaction from the two intermediate values. The stochastic function clauses are given by:

$$log(w_1) : hr(S, 1)$$
$$log(w_2) : hr(S, 2)$$

The above clauses specify the prior over the intermediate nodes as a function of their weights w_i[4]. At the first level, the value of the intermediate node is chosen according to an uniform distribution (mean combining rule) and hence the weights of the MLN clauses are 0 and are not presented here. Finally, we present the integrity constraints that restrict the multiplexer to choose only one value from among a set of possible values

$$\begin{aligned}
&\infty : h1(S, T1, C1, G1) \wedge h1(S, T2, C2, G2) \Rightarrow (T1 = T2 \wedge C1 = C2 \wedge G1 = G2)\\
&\infty : \text{ Exists T,C,G } h1(S, T, C, G)\\
&\infty : h2(S1, P1, Q1) \wedge h2(S2, P2, Q2) \Rightarrow (P1 = P2 \wedge Q1 = Q2)\\
&\infty : \text{ Exists P,Q } h2(S, P, Q)\\
&\infty : hr(S, R1) \wedge hr(S, R2) \Rightarrow R1 = R2\\
&\infty : \text{ Exists R } hr(S, R)
\end{aligned} \tag{10}$$

4.2 Noisy Functions:

For this case, let us assume a single rule and that the different instantiations of that rule are combined using a noisy function. For *Noisy-Or*, the marginal is computed as,

$$P(b = \mathrm{T}|a_1, \ldots, a_n) = 1 - \prod_{i=1}^{n} f_i^{a_i} \tag{11}$$

where f_i's represent the probability that a present (Boolean-valued) cause, a_i, fails to make the result b true. When converting these to MLNs, the transformation is mostly similar to the earlier case. Though the CPT clauses are constructed similarly, we present them for clarity. They are of the form:

$$\begin{aligned}
&\infty : \neg a(X, Y) \Rightarrow \neg t(X, Y, 1).\\
&w_i : \quad a(x_i, Y) \Rightarrow t(x_i, Y, 1).
\end{aligned}$$

where, $w_i = \log((1 - f_i)/f_i)$. As can be seen, if $a(X, Y)$ is false for a particular value of X, $t(X, Y, 1)$ will always be false while if a is true, t can be false due to some noise. The multiplexer and integrity clauses are similar to the average case. A careful reader will note that the multiplexer and integrity clauses are redundant for this case as they derive the r-values directly from t-values as shown below. The stochastic function (deterministic here) is given by,

$$\infty : r(Y, I) \Leftrightarrow \exists X.t(X, Y, I) \tag{12}$$

This asserts that $r(Y, i)$ is true if and only if some $t(X, Y, i)$ is true, which is effectively deterministic Or applied to noisy versions of the inputs. It can be shown that this set of clauses exactly capture the distribution given by equation 11. We omit the proof as it is a trivial mathematical exercise similar to the weighted-mean case.

Noisy existentials can be constructed similarly, except that we have tied weights. When constructing noisy-and, the noise adds a probability of success instead of a probability of failure:

$$\begin{aligned}
&w_i : \neg a(X, Y, 1) \Rightarrow \neg t(X, Y, 1)\\
&\infty : \quad a(x_i, Y, 1) \Rightarrow t(x_i, Y, 1).
\end{aligned}$$

[4] These weights are the weights of the combining function and must not be confused with the weight of the MLN clauses.

The multiplexer and the stochastic functions are also modified accordingly to reflect the *And* function.

Also, note that any MLN can be seen as a noisy-and in which the target $b(Y)$ is known to be true and each $a(x_i, Y)$ is a clause from the original MLN. Because of the infinite-weight conjunction, all t_i must be true. Since t_i is true, we can simplify each implication $\neg a(x_i, Y) \Rightarrow \neg t(x_i, Y)$ to $a(x_i, Y)$. The final, simplified MLN is therefore just the weighted clauses from the original MLN: $w_i : a(x_i, Y)$.

Worked Example: We now present the rules for the satisfaction example where $CR2$ is *Or* while $CR1$ is *Noisy-Or* with q_i as inhibition probability. The following are the CPT clauses

$$log(\tfrac{1-q_1}{q_1}) : student(S), course(C), takes(T, S, C), grade(T, G) \Rightarrow t1(S, T, C, G)$$
$$log(\tfrac{1-q_2}{q_2}) : student(S), paper(P, S), quality(P, Q) \Rightarrow t2(S, P, Q)$$

Note that the CPT parameters are a function of the noise (inhibition) for the two rules. The multiplexer clauses can be constructed similar to the weighted mean case given in Equation 9. The stochastic function clauses are created according to the following clauses. Note that the stochastic function clauses state that there is an Or function at each level (the noise at the first-level is captured in the CPT clauses).

$$\infty : r(S, 1) \Leftrightarrow \text{Exists T,C,G } t1(S, T, C, G)$$
$$\infty : r(S, 2) \Leftrightarrow \text{Exists P,Q } t2(S, P, Q)$$
$$\infty : satisfaction(S) \Leftrightarrow \text{Exists R } tr(S, R)$$

The integrity constraints are similar to the earlier case (Equation 10). The example here combines *Noisy-Or* with the Or combining rule. We can similarly imagine combining different decomposable combining rules at the different levels. We are not presenting all the combinations in this work but note that the same templates can be used to construct the different sets of combining functions.

4.3 Algorithm for creating MLN clauses from Decomposable Combining Rules

This formulation allows for arbitrary nesting of combining rules. The combining rules used for combining different instantiations of different rules could be different. For instance, we can imagine a situation such as $NoisyAnd(w_1\ A, w_2\ B, w_3\ NoisyOr(w_4\ C, w_5\ Mean(w_6\ D, w_7\ E, w_8\ F)))$ where we have both Noisy-Or and Mean inside the Noisy-And function. w_i's are the weights while A through F are first-order logic Formulae. Such a representation is a significant generalization of MLNs.

Figure 3 describes the pseudocode for constructing MLNs from a set of FOCI statements combined using combining rule CR_2. Each statement s_i has its own 1^{st} level combining rule CR_1^i. Lines 3 through 9 present the methods for constructing the clauses corresponding to s_i and its combining rule. For each independent parameter in the CPT of s_i, a clause is created. Also for each s_i, one multiplexer clause, one stochastic function and two integrity constraints are created. Once all the 1^{st} level combining rules are considered, the clauses corresponding to CR_2 are constructed in lines 10 through 14. We note that it requires $O(1)$ to construct each clause.

Fig. 3. CreateMLNClauses (FOCI Statements S, CR_2)

```
1: MLNClauses clauseList = [ ];
2: // Each FOCI Statement s_i has CPT, Predicates, CR_1^i
3: For Each FOCI statement s_i ∈ S
4:    For Each Independent parameter θ_i^j in s_i.CPT
5:       Add one CPT clause to clauseList, e.g. as in Eqn 4
6:    Add to clauseList based on 1^st level combining rule CR_1^i:
7:       One multiplexer clause as in Eqn. 5
8:       One stochastic function clause, e.g. as in Eqn. 12
9:       Two integrity constraint clauses as in Eqn. 7
10: For Each FOCI statement s_i ∈ S
11:    Add to clauseList based on the 2^nd level combining rule CR_2^i:
12:       One multiplexer clause as in Eqn. 6
13:       One stochastic function clause, e.g. as in Eqn. 12
14: Two integrity constraint clauses as in Eqn. 7
```

4.4 Complexity of the resulting MLN

We now provide a bound on the number of clauses required by such an MLN. In particular, we consider the general SRL case of multi-level combining rules where the instantiations of a single rule are combined using CR_1 and different rules are combined using CR_2.

Theorem 1. *For any joint distribution which can be represented by* n *FOCI statements combined with nested decomposable combining rules, and* k *independent parameters, there exists an equivalent MLN of* O(nk) *rules which can be constructed in* O(nk) *time.*

Proof (sketch) The proof of equivalence is straightforward from the definition of the various clauses. Let n be the number of FOCI statements and k be the number of independent CPT parameters. From the algorithm in Figure 3, for each rule, there are k CPT clauses, one multiplexer clause, one stochastic function clause and two integrity constraints yielding $k + 4$ clauses. Hence the total number of clauses created in lines $(3 - 9)$ is $n(k + 4)$. For lines $10 - 14$ of the algorithm, the number of clauses is $n(1 + 1) + 2 = 2(n + 1)$. Hence the total number of clauses is $n(k + 6) + 2 = O(nk)$. Since each clause can be constructed in constant time given the FOCI statement and the combining rule, the resulting MLN can be constructed in $O(nk)$ time. Note that the minimal number of clauses required to model FOCI statements using MLN is $O(nk)$ as we need a clause for every parameter. Hence, our translation creates a model that is no more complex than the minimal MLN. ■

4.5 MLN Macros

While theoretically MLNs can represent most of the distributions that we considered, it seems impractical to expect a domain expert to come up with these rules. Firstly, the domain expert has to understand the underlying distribution and the combining rules as operating on values as against distributions (i.e, multiplexers). Secondly, the domain expert needs to be an MLN expert as well and has to understand the translation. In this

section, we present a macro that can be used to construct MLNs given the domain expert's statements. The key idea is to remove the burden of specifying the MLNs from the user and allow our "translator" to create the MLNs corresponding to the true distribution. We now present the structure of the macro:
CR{ CR_1 : $X_1^1 \wedge ... \wedge X_{n_1}^1 \Rightarrow Y$
CR_2 : $X_1^2 \wedge ... \wedge X_{n_2}^2 \Rightarrow Y$ . . . }
The above macro can be interpreted as: CR, CR_1 and CR_2 are the combination functions – *And, Or, Noisy-Or, Noisy-And etc*. While CR_i combines the multiple instantiations of clause i, CR combines the multiple clauses. The clauses that are to be combined are specified next in the macro. X_i^j and Y are predicates. The first clauses specifies n_1 causes for the target predicate Y. X_1^1 is the first cause of Y in rule 1 and so on. Instead of writing 2^n different clauses, the user specifies a single clause that is then unrolled into the different clauses by the translator. The user can specify several clauses that can be combined.

The translator then converts these macros to the MLN clausal representation. A natural question now is: where do the weights come from? A simple solution would be to construct the clauses and allow the underlying MLN package (in our case *Alchemy* [11]), to learn the weights. We could hold the weights of the hard clauses (integrity constraints and some multiplexer clausers) and instruct Alchemy to learn the weights of only the "soft clauses" leading to a more efficient learning. In cases where the training data is not available, we allow the conditional probabilities to be specified as 2^n array corresponding to the different configurations of the predicates in the body of the clause. Hence the clauses are now of the form, $\langle p_1, p_2, ..., p_{2^n} \rangle X_1 \wedge ... \wedge X_n \Rightarrow Y$, where $p_i = P(Y = T | Config(X_1, ..., X_n) = i)$ is the conditional probability of the target being T given that the truth value of the predicates in the body form the i^{th} configuration. Similarly, the parameters of the combining rules (ex. weights of weighted mean) can also be specified as $CR\langle w_1, ..., w_m \rangle$ for m clauses. Based on the combining rule used, the translator then computes the weights of the different clauses based on the probabilities and assigns the weights to the corresponding clauses.

5 Experiments

In the following experiments, we used the Alchemy system[5] to learn the weights and/or perform inference. The same settings were used for both MLNs with combining rules (denoted by MLN^+) and the default MLNs(MLN^*). The clauses of the MLN^* are the parent configurations of the CPT of each rule. Hence, for each independent parameter of the CPT, there exists a clause in MLN^*. MLN^* was chosen so that it had the same number of parameters as that of a directed model to make a fair comparison. The clauses of MLN^+ consist of the CPT clauses and the multiplexer, stochastic function and integrity clauses. For both MLN^+ and MLN^*, we used the same settings for the learning and inference algorithms(i.e., used the same number of iterations, discriminative learning, same number of MCMC steps, MC-SAT for inference etc.).

We present our learning results in two real-world domains: *Cora* and *UW-CSE*. The goal of the experiment is: given minimal domain knowledge (typically 2 rules to predict the target), will the structure imposed by combining rules be useful in learning a good

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

model? We compared MLN^* against MLN^+ for Noisy-Or combining rule. For the UW-dataset, the goal was to predict the *advisedBy* relationship between a student and a professor. The rules that we used were:

```
student(S) ∧ professor(P) ∧ course(C) ∧ taughtBy(P,C,Q) ∧ ta(S,C,Q)
⇒ advisedBy(S,P).
student(S) ∧ professor(P) ∧ publication(P,W) ∧ publication(S,W)
⇒ advisedBy(S,P).
```

MLN^* used all the combinations of the predicates in the head of the clauses and learned weights for each of them. For MLN^+, we used Noisy-Or as the combining rule at both levels. We learned the weights using Alchemy and used MC-SAT for performing inference. We trained the algorithms on the AI group data that consisted of 35 positive instances of the *advisedBy* relation. We present the average likelihood of the test set in the last column of Table 1. Note that since we are in the relational setting, the test set will mostly consist of negatives. Hence, an algorithm that always predicts *false* will have a reasonably high likelihood. To avoid this situation, we forced the test set to contain 50% negative examples by sampling the negative examples randomly. This way a likelihood of 0.5 would mean that everything is either predicted *true* or as *false*. There were a total of 80 test examples with 40 of them being positive instances of the *advisedBy* relation.

Domain	Algorithm	AUC-ROC	AUC-PR	Likelihood
Cora	MLN^+	1.0	1.0	0.987
	MLN^*	1.0	1.0	0.963
UW	MLN^+	0.560	0.672	0.611
	MLN^*	0.4722	0.523	0.5

Table 1. Results on real world domains.

We also compare the area under curve for the ROC and PR curves. MLN^* was not able to learn reasonable weights with a small number of rules and hence predicted everything as 0. In a test-set with 50% positive examples, this yielded a likelihood of 0.5. On the other hand, with MLN^+, we were able to learn a more reasonable model that had a higher likelihood. More importantly, MLN^+ did not predict every query predicate as 0 or 1 and instead had a reasonable distribution over the target. When we added more rules to MLN^* (7 more rules from Alchemy that were earlier used in other MLN experiments to predict *advisedBy*) the average likelihood increased to 0.63. The values of AUC for ROC and PR for MLN^+ are significantly higher than MLN^*. This demonstrates that in this domain the use of more complex combining functions seem to improve the performance of MLN learning.

The results were far more impressive for *Cora* dataset where the goal is to predict whether two citations refer to the same one. The training set consisted of about 7500 examples (about 70% of them were negative examples) and the test set consisted of 100 examples (out of which 50% were negative examples). The two rules that we used were:

```
Author(bc1,a1) ∧ Author(bc2,a2) ∧ SameAuthor(a1,a2) ⇒ SameBib(bc1,bc2).
Title(bc1,t1) ∧ Title(bc2,t2) ∧ SameTitle(t1,t2) ⇒ SameBib(bc1,bc2).
```

As can be seen from the table, MLN^+ learned nearly the perfect model for the domain and had a very high likelihood and AUC values. This clearly showed that with just two rules, given some more knowledge (as hard constraints of the combining rules), MLN^+ was able to learn a highly predictive model. While MLN^* with exactly the same setting as MLN^+ (i.e., discriminative, rules for all the combinations of the predicates etc.) predicted all the test examples as 0 and thus had a lower likelihood and AUC values (very similar to the UW data set). To improve the performance of MLN^*, we changed the settings (to generative learning, dropped some seemingly irrelevant clauses that had a large number of groundings). With these changes, we were able to get MLN^* to perform comparably with MLN^+.

Specialized inference algorithms Admittedly, the presence of hidden predicates increased the running time of Alchemy[6], but this motivates the need for learning algorithms that exploit the special structure efficiently (as we used the default EM learning algorithm of Alchemy to learn weights for MLN^+).

We implemented an inference algorithm that exploits the special structure of these clauses. We do not present the algorithm in detail in this work as the goal of this work is to show that the combining rules can be captured in MLNs and motivate the need for specialized algorithms that can exploit the local models similar to the ones presented in [13]. Initial results indicate that the time taken for MLN^+ in the UW-data set is 4 seconds while that of MLN^* is 30 seconds to obtain the same results presented in Table 1. As can be seen, there is a drastic improvement when the inference algorithm exploits the knowledge of the structure of the clauses. The modified inference algorithm assumes that the structure of the MLN is the one presented in Figure 2 and performs sampling on this network. Hence, it exploits the knowledge about the structure of the network and the multiplexers in the network. We expect that learning localized models in MLNs will enable efficient learning and inference. We are currently working on formalizing the details of the learning algorithm that uses this inference method. Our hypothesis is that since learning requires inference in its inner loop, the specialized inference algorithm will yield faster learning of the parameters.

6 Conclusions

Combining rules capture the notion of causal independence for SRL models. We have presented an algorithm for representing a class of combining rules (decomposable combining rules) in an undirected model (MLN). We derived the equivalent clauses and provided a bound on the number of clauses required for the representation. Our experiments demonstrated that for a small number of clauses, combining functions are useful in learning more accurate models. The structure imposed by these functions help in guiding the learning algorithms towards reasonable weights. Jaeger in [9] showed that RBNs can capture MLNs and pointed out to the reverse as an open problem. We take an important step in that direction by showing how MLNs can capture combination functions of the directed models and in turn, most of the features of directed models.

However, this translation from combining rules to MLNs is not without its cost. We found that the inference in the resulting MLN is 4-5 times slower than the one that

[6] The increase in running times was around 5 times on average.

does not use the combining rules. The problem is that while the declarative knowledge embedded in the combining rules can be encoded into clauses and given to MLNs, they have no effective means to exploit the causal independence for controlling inference. To be effective, the inference engine has to essentially rediscover the hidden structure that is naturally exploited by the directed models. One possible future direction is to develop specialized inference algorithms that can detect structure in MLNs and exploit it for efficiency. We have taken the first step in this direction, but are still working on the details of the learning algorithm that will exploit the structure. A more general and important direction is to develop hybrid models that allow us to specify different parts of the model differently and combine them using a decomposable structure. This should allow the application of specialized learning algorithms inside each module, and combine the results in an efficient manner.

7 Acknowledgements

Sriraam Natarajan, Tushar Khot and Jude Shavlik gratefully acknowledges the support of Air Force Research Laboratory (AFRL) under prime contract no. FA8750-09-C-0181. Prasad Tadepalli greatfully acknowledges the support of DARPA grant FA8750-09-C-0179. Kristian Kersting was supported by the Fraunhofer ATTRACT fellowship STREAM and by the Europen Commission under contract number FP7-248258-First-MM. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of the Air Force Research Laboratory (AFRL),US government or DARPA.

References

1. P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for AI*. Morgan & Claypool, San Rafael, CA, 2009.
2. D. Fierens, H. Blockeel, M. Bruynooghe, and J. Ramon. Logical bayesian networks and their relation to other probabilistic logical models. In *ILP*, 2005.
3. L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. *Relational Data Mining, S. Dzeroski and N. Lavrac, Eds.*, 2001.
4. L. Getoor and J. Grant. PRL: A probabilistic relational language. *Mach. Learn.*, 62(1-2):7–31, 2006.
5. L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning*. MIT Press, 2007.
6. D. Heckerman and J. Breese. A new look at causal independence. In *UAI*, 1994.
7. M. Jaeger. Relational Bayesian networks. In *Proceedings of UAI*, 1997.
8. M. Jaeger. Parameter learning for relational bayesian networks. In *ICML*, 2007.
9. M. Jaeger. Model-theoretic expressivity analysis. In *Probabilistic Inductive Logic Programming*, 2008.
10. K. Kersting and L. De Raedt. Bayesian logic programming: Theory and tool. In *An Introduction to Statistical Relational Learning*, 2007.
11. S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2007.
12. D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *IJCAI*, 1997.
13. S. Natarajan, P. Tadepalli, T. G Dietterich, and A. Fern. Learning first-order probabilistic models with combining rules. *Special Issue on Probabilistic Relational Learning, AMAI*, 2009.
14. N. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *JAIR*, 5:301–328, 1996.