# Generative Structure Learning for Markov Logic Networks Based on Graph of Predicates

**Quang-Thang Dinh*, Matthieu Exbrayat*, Christel Vrain ****
*LIFO, Université d'Orléans, Orléans, France
**LIP6, Université Pierre et Marie Curie, Paris, France
{thang.dinh,matthieu.exbrayat,christel.vrain}@univ-orleans.fr

## Abstract

In this paper we present a new algorithm for generatively learning the structure of Markov Logic Networks. This algorithm relies on a graph of predicates, which summarizes the links existing between predicates and on relational information between ground atoms in the training database. Candidate clauses are produced by means of a heuristical variabilization technique. According to our first experiments, this approach appears to be promising.

## 1 Introduction

Statistical Relational Learning (SRL) seeks to combine the power of both statistical learning and relational learning in order to deal with both uncertainty and complex relational structures, which are required by many real-world applications. A large and growing number of SRL approaches have been proposed, including PRISM [Sato and Kameya, 1997], Probabilistic Relational Models (PRM) [Friedman *et al.*, 1999], Bayesian Logic Programs (BLP) [Kersting and De Raedt, 2007], Relational Dependency Networks (RDN) [Neville and Jensen, 2004], Relational Markov Models (RMM) [Anderson *et al.*, 2002] and Markov Logic Networks (MLN) [Richardson and Domingos, 2006].

MLNs generalize both full First-Order Logic (FOL) and Markov networks (MN). A MLN consists of a set of weighted clauses, viewed as templates for the features of a MN. Learning a MLN can be decomposed into structure learning and weight learning. Structure learning leads to a set of first-order formulas while weights learning estimates the weight associated with each formula of a given structure. Early developed strategies focused on parameter learning, the underlying structure being provided. Such an approach is likely to lead to non optimal results when these clauses do not capture the essential dependencies in the domain. As a consequence, recent work on MLNs has focused on global learning methods that integrate the structure learning step. This integration however remain challenging, due to the very large search space. Consequently, only a few practical approaches have been proposed to date, among which the MSL [Kok and Domingos, 2005], BUSL [Mihalkova and Mooney, 2007], ILS [Biba *et al.*, 2008b], LHL [Kok and Domingos, 2009], LSM [Kok and Domingos, 2010] and HGSM [Dinh *et al.*, 2010b].

MLNs can be learned in generative (i.e. learning all predicates in the domain) and discriminative (i.e. learning a single target predicate) frameworks. In this paper we focus on generative learning, our contribution consisting in a new algorithm for the generative learning of the structure of a MLN. This algorithm first constructs a graph of predicates that synthesizes the paths existing in the training database. In other words, this graph highlights the binary associations of predicates that share constants. Building candidate clauses is thus made faster, as the algorithm focuses on paths that exist in the graph. Each path is transformed into a clause by means of a heuristic variabilization technique. The algorithm creates relevant clauses w.r.t. the graph, according to a progressive length (starting with clauses of length 2). Each clause is then evaluated for insertion in the final MLN.

The remainder of the paper is organized as follows. We first remind some background elements in Section 2. Then we describe our algorithm in Section 3. Experiments are presented in Section 4. Related work is briefly drawn in Section 5. Last, Section 6 concludes this paper.

## 2 Backgrounds

Let us recall here some basic notions of FOL, which will be used throughout this paper. We consider a function-free first order language composed of a set $\mathcal{P}$ of predicate symbols, a set $C$ of constants and a set of variables. An *atom* is an expression $p(t_1, \ldots, t_k)$, where $p$ is a predicate and $t_i$ are either variables or constants. A *literal* is either a positive or a negative atom; a *ground literal (resp. variable literal)* contains no variable (resp. only variables). A *clause* is a disjunction of literals. The *length* of a clause $c$, denoted by *len(c)*, is the number of literals in $c$. Two ground literals (resp. variable literals) are said to be connected if they share at least one ground argument (one variable). A clause (resp. a ground clause) is *connected* when there is an ordering of its literals $L_1 \vee \ldots \vee L_p$, such that for each $L_j$, $j = 2 \ldots p$, there exists a variable (a constant) occurring both in $L_j$ and in $L_i$, with $i < j$. A *world* is an assignment of truth value to all possible ground atoms. A *database* is a partial specification of a world; each atom in it is true, false or (implicitly) unknown. In this paper, we use the closed world assumption: if a ground atom is not in the database, it is assumed to be *false*.

A MLN is a set of weighted first-order formulas. Together with a set of constants, it defines a MN [Pearl, 1988] with

one node per ground atom and one feature per ground formula. The weight of a feature is the weight of the corresponding first-order formula. The probability distribution over a possible world *x* specified by the ground MN is: $P(X = x) = \frac{1}{Z} \exp \sum_{i \in F} \sum_{j \in G_i} w_i g_j(x)$ where *Z* is a normalization constant, *F* is the set of formulas, $G_i$ and $w_i$ are respectively the set of groundings and weight of the *i*-th formula, and $g_j(x) = 1$ if the *j*-th ground formula is true.

The Pseudo-Log-Likelihood (PLL) of $x$ is $\log P_w(X = x) = \sum_{l=1}^{n} \log P_w(X_l = x_l | MB_x(X_l))$ where $MB_x(X_l)$ is the state of the Markov blanket of the ground atom $X_l$, $x_l$ is the truth value of $X_l$ in $x$. Generative approaches optimize the PLL using iterative scaling or a quasi-Newton optimization method such as the *L-BFGS* [Sha and Pereira, 2003].

The weighted PLL (WPLL) measure is $\log P_w^*(X = x) = \sum_{r \in R} c_r \sum_{k=1}^{g_r} \log P_w(X_{r,k} = x_{r,k} | MB_x(X_{r,k}))$ where *R* is the set of predicates, $g_r$ is the number of groundings of predicate $r$, $x_{r,k}$ is the truth value of the *k*-th grounding of *r*, and $c_r = 1/g_r$ [Kok and Domingos, 2005].

# 3 Generative Structure Learning Based on Graph of Predicates

In this section, we first introduce our definition of graph of predicates (GoP) and then describe how we use this graph to learn the structure of a MLN from a training dataset DB and an initial MLN (which can be empty).

## 3.1 Graph of Predicates

Let us consider here a set $\mathcal{P}$ of *m* predicates $\{p_1, ..., p_m\}$.

**Definition 1** *A template atom of a predicate p is an expression p(type*$_1$*, ... , type*$_n$*), where type*$_i$*,* $1 \leq i \leq n$ *indicates the type of the i-th argument.*

**Definition 2** *A link between two template atoms* $p_i(a_{i_1}, ..., a_{i_q})$ *and* $p_j(a_{j_1}, ..., a_{j_k})$ *is an ordered list of pairs of positions* $u$ *and* $v$ *such that the types of arguments at position* $u$ *in* $p_i$ *and* $v$ *in* $p_j$ *are identical. It is denoted by* $link(p_i(a_{i_1}, ..., a_{i_q}), p_j(a_{j_1}, ..., a_{j_k})) = \langle u\ v | ... \rangle$*, where* $a_{i_u} = a_{j_v}$, $1 \leq u \leq q, 1 \leq v \leq k$.

*The link between two predicates* $p_i$ *and* $p_j$*, denoted by* $link(p_i, p_j)$*, is the set of all possible links between their template atoms* $p_i(a_{i_1}, ..., a_{i_q})$ *and* $p_j(a_{j_1}, ..., a_{j_k})$.

*When two template atoms do not share any type, there exists no link between them.*

**Definition 3** *A formula satisfying a* $link(p_i(a_{i_1}, ..., a_{i_q}), p_j(a_{j_1}, ..., a_{j_k})) = \langle si_1\ sj_1 | \ ... | si_c\ sj_c | \rangle$ *is a disjunction of literals* $p_i(V_{i_1}, ..., V_{i_q}) \vee p_j(V_{j_1}, ..., V_{j_k}))$ *where* $V_t$*,* $t = i_1, ..., i_q, j_1, ..., j_k$*, are distinct variables except* $V_{i_{si_d}} = V_{j_{sj_d}}$, $1 \leq d \leq c$.

The definitions of link and formula satisfying a link are naturally extended to negation. For example, $link(p_i(a_{i_1}, ..., a_{i_q}), !p_j(a_{j_1}, \ldots, a_{j_k}))$ is the link between $p_i(a_{i_1}, ..., a_{i_q})$ and $p_j(a_{j_1}, ..., a_{j_k})$: $link(p_i, !p_j) \equiv link(p_i, p_j)$.

**Example 4** *We consider a domain consisting of two predicates* AdvisedBy *and* Professor *respectively with two*

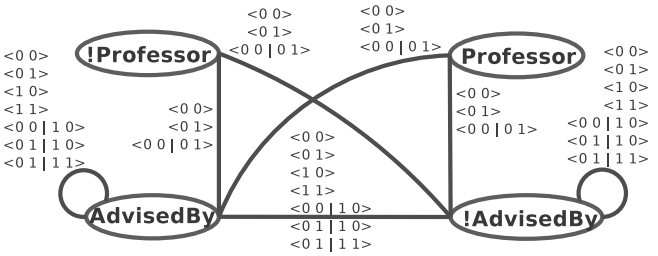


Figure 1: Example of graph of predicates

*template atoms* AdvisedBy(person, person) *and* Professor(person). *The argument (type)* person *appears at position* 0 *of* Professor(person) *and at positions* 0 *and* 1 *of* AdvisedBy(person, person). *Several possible links exist between them, as for instance* $\langle 0\ 0 \rangle$ *and* $\langle 0\ 1 \rangle$*, and a possible formula satisfying the latter one is* Professor(A) $\vee$ AdvisedBy(B, A).

*We also have* link(AdvisedBy, AdvisedBy)= $\{\langle 0\ 0 \rangle, \langle 0\ 1 \rangle, \langle 1\ 0 \rangle, \langle 1\ 1 \rangle, \langle 0\ 0 \mid 1\ 0 \rangle, \langle 0\ 0 \mid 1\ 1 \rangle, \langle 0\ 1 \mid 1\ 0 \rangle, \langle 0\ 1 \mid 1\ 1 \rangle\}$.

*It must be noted that several links are not considered here. For example, the link* $\langle 0\ 0 \mid 1\ 1 \rangle$ *leads to a formula composed of two similar literals. The link* $\langle 1\ 0 \mid 0\ 1 \rangle$ *is similar to the link* $\langle 0\ 1 \mid 1\ 0 \rangle$ *because they are satisfied by similar formulas. The link* $\langle 1\ 1 \mid 0\ 0 \rangle$ *is also similar to the link* $\langle 0\ 0 \mid 1\ 1 \rangle$*. The same remark can be done for* $link(Professor, Professor) = \langle 0\ 0 \rangle$, $link(Professor, !Professor) = \langle 0\ 0 \rangle$, ...

**Definition 5** *Let DB a database,* $\mathcal{P}$ *a set of* $m$ *predicates* $\{p_1, ..., p_m\}$ *occurring in* $DB$*, and D the domain (set of constants). An undirected graph of predicates (GoP) is a pair* G=(V, E) *composed of a set* V *of nodes (or vertices) together with a set* E *of edges, where:*

*i. A node* $v_i \in V$ *corresponds to a predicate* $p_i$ *or its negation,* $|V| = 2 \times |\mathcal{P}|$

*ii. If there exists a link between two template atoms of predicates* $p_i$ *(or* $!p_i$*) and* $p_j$ *(or* $!p_j$*), then there exists an edge between the corresponding nodes* $v_i$ *and* $v_j$*, and this edge is labelled by the link.*

Figure 1 illustrates a GoP for the domain in Example 4.

## 3.2 Structure of Algorithm

Given as inputs a training dataset DB, a background MLN (possibly empty) and a positive integer *maxLength*, specifying the maximum length of clauses, we present an algorithm, called Generative Structure Learning based on a graph of Predicates (*GSLP*), to learn generatively a MLN structure. Algorithm 1 sketches main steps of GSLP.

The approaches developed in [Kok and Domingos, 2005], [Biba *et al.*, 2008b] and [Huynh and Mooney, 2008] explore in an intensive way the space of clauses, generating a lot of useless candidate clauses, while the ones developed in [Mihalkova and Mooney, 2007], [Kok and Domingos, 2009] and [Dinh *et al.*, 2010b]) search all paths of ground atoms in the database in order to reduce computation time. However, searching in this space is also computationally expensive. We aim at generating a set of interesting potential clauses based on three observations. First, associations amongst predicates constitute a smaller model space than clauses and can thus

**Algorithm 1:** GSLP(DB, MLN, maxLength)

```
1  Initialization: set of clauses CC = ∅, set of paths SP = ∅;
2  Add all unit clauses to the final MLN;
   // Creating a graph of predicates
3  Create graph G of predicates;
4  Reduce Edges of Graph (G, CC, SP) ;
   // Building a set of candidate clauses
5  for length=2 to maxLength do
6      CC ← CreateCandidateClauses(DB, G, SP, length);
   // Learning the final MLN
7  Eliminate several Candidate Clauses (CC) ;
8  Add Candidate Clauses into the MLN (CC, MLN) ;
9  Prune Clauses out of the MLN ;
10 Return(MLN);
```

be searched more efficiently [Kersting and De Raedt, 2007]. Second, it seems useless to consider connected clauses that do not correspond to any instance in the database. Third, we believe that the more useful a connected clause $A_1 \vee ... \vee A_n$ is, the larger number of true ground atoms in the database it covers. In other words, relevant clauses are mostly the ones that are frequent enough in terms of true instances. It is obvious that if a connected formula is frequent, then its connected sub-formulas are also at least as frequent as it (similar remarks serve as the basis of many well known strategies for frequent pattern mining). We thus propose to generate candidate clauses, starting from binary ones (i.e. consisting of two connected atoms) then gradually extend them to get longer and better ones. The problem now is transformed into finding such a set of binary connected clauses $A_i \vee A_j$ from which the set of candidate clauses is gradually lengthened.

Before describing the algorithm in details, it is important to explain how a clause is evaluated. Traditional ILP methods use the number of true instances which is not suitable for uncertainty in MLN, while most methods for MLN use the WPLL, which sometimes miss clauses with good weights. We propose to evaluate a clause $c$ based on its weight (*c.weight*) and gain (*c.gain*). These two values are computed according to a temporary MLN composed of the unit MLN plus $c$. The weights of this MLN are then learned to maximize the measure (i.e. WPLL). The weight of $c$ is its weight in this temporary MLN. The gain of $c$ is defined as: $newMea - unitMea - penalty * len(c)$, where $newMea$, $unitMea$ are respectively the performance measure (WPLL) of the temporary MLN and of the unit MLN; *penalty* is a coefficient. It essentially corresponds to the improvement of the measure when adding $c$ to the unit MLN. To estimate the interest of $c$, instead of using the number of true ground instances or only the WPLL, we use the product $c.mf = |c.weight| * c.gain$. As a clause weight can be negative, we use absolute value (denoted by $|\ |$) to consider both positively and negatively weighted clauses. This is reasonable because in a MLN, $c.weight$ implies how strong $c$ is and $c.gain$ describes how the measure is increased when adding $c$ into the unit MLN. Nevertheless, this measure is no longer monotonous, contrary to the classical frequency measure.

In a MLN, a weight associated to a formula reflects how strong a constraint is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal. Concerning unit clauses (clauses of a single atom), the weights of these capture the marginal distribution of predicates, allowing longer clauses to focus on modeling predicate dependencies. For this reason, adding all unit clauses into the MLN is usually useful. This is also the first step of our algorithm and we call this first MLN the *unit MLN* (*line 2*).

Our clause-generation algorithm relies on a GoP in order to search clauses through the space of associations between predicates. For each predicate, we create two nodes (corresponding to the predicate itself and its negation), hence the graph has only $2 \times |\mathcal{P}|$ nodes, where $|\mathcal{P}|$ is the number of predicates in the domain. The number of edges incident to a node depends on the number of arguments of the corresponding predicate S and their type relations in the domain, but it is usually not too large. The search space of paths in GoP is then much smaller than the whole search space of paths of ground atoms, especially when considering a domain with a large number of predicates and when there exists a lot of shared-arguments between ground atoms.

Every edge *e* of GoP is created corresponding to a possible link between two nodes (*line 3*). We can see that *e* always corresponds to a binary clause *bc* satisfying its link (label). We do not consider clauses with no or a few true instances in the database, hence if the number of true instances of *bc* is less than or equal to a given threshold *nti* then *e* is discarded. Otherwise, *bc* is evaluated to compute its measure of frequency, *bc.mf*. In our method, only edges with a big "enough" measure of frequency will be considered, therefore if *bc.mf* is smaller than or equal to a given threshold *mft*, then *e* is also eliminated. The number of edges of GoP is thus remarkably reduced (*line 4*), leading to a more efficient search.

Next, the algorithm generates a set of candidate clauses based on this reduced graph (*line 5-6*). The idea is to lengthen successively every binary clause to generate longer ones such that the extended clauses improves the measure *c.mf* of frequency. A clause is considered as a candidate clause when it reaches a given *maxLength*, or when it does not lead to any longer and better clause (i.e. associated with an improved WPLL). We explain this in more details in Subsection 3.3.

Having got a set *CC* of candidate clauses, the algorithm next eliminates some clauses (*line 7*) before adding them into the MLN in turn (*line 8*). Finally, it prunes some clauses of the MLN (*line 9*). We present these steps in Subsection 3.4.

### 3.3 Building Candidate Clauses

As we have mentioned above, the algorithm extends step by step every binary clause. Let *SP* a set of similar-length paths and *CC* a set of candidate clauses. Initially, *SP*, *CC* respectively contain all *1-length* paths and binary clauses: each pair of $sp_i \in SP$ and $cc_i \in CC$ corresponds to an edge $e_i$ of the reduced GoP, where $sp_i$ is the label of $e_i$ and $cc_i$ is the binary clause satisfying $sp_i$. For longer paths, we extend every *k-length* path $pk = p_1, ..., p_k \in SP$ to get a longer *(k+1)-length* path $pk1 = p_1, ..., p_k, p_{k+1}$ by finding an edge $p_{k+1}$ connecting to at least a node in *pk*. It must be noted that by indexing the nodes of the graph we can find quickly the set of *(k+1)-length* paths, since for a node $p$ in *pk* we only

have to look for every edge $p_{k+1}$ connecting $p$ to its higher indexed nodes. In our algorithm, each path is variabilized to form only one clause (as explained in next paragraph). Assume that $c_k$, $c_{k+1}$ are respectively two variabilized clauses of *pk* and *pk1*. The clause $c_{k+1}$ is evaluated to check whether $c_{k+1}.mf$ greater than $c_k.mf$. If it is true, the path $pk1$ is stored into *SP* for the next iteration, otherwise it is no longer considered. If there exists no clause $c_{k+1}$ extended from the path of $c_k$ such that $c_{k+1}.mf > c_k.mf$, the clause $c_k$ becomes a candidate clause stored into *CC*. At last iteration, all clauses corresponding to *maxLength*-length paths in *SP* are added into *CC*.

Let us consider now the process of variabilization. Most approaches variabilize a path of ground atoms by simply replacing constants by variables. We emphasize that, here, we produce clauses from paths in the reduced GoP, i.e. a list of edges containing only information of predicates and positions of shared arguments. There exists a lot of clauses satisfying this path, thus building all possible clauses is not reasonable: the more clauses generated, the more time spent to evaluate them and to learn the final MLN. We variabilize heuristically a path to generate only one connected clause. First, in order to reduce the number of distinct variables in the clause, we handle the predicates in the order of descending frequency in the path. For each predicate, we perform variabilization for shared variables first, and then for unshared ones. An argument of a predicate is variabilized by a new variable if its position does not appear in the link of any edge in the path.

**Example 6** *For example, variabilizing the path composed of two links* !advisedBy $\langle 0\ 1 \rangle$ !advisedBy *and* !advisedBy $\langle 0\ 0 \rangle$ student *generates the clause* !advisedBy(A,B) $\vee$ !advisedBy(C,A) $\vee$ student(A). *Conventionally, the link between* student *and* advisedBy *is applied to the first* advisedBy.

### 3.4 Learning the final MLN

Weights are learned using the standard L-BFGS algorithm to maximize PLL. For inference, as in previous studies, we use MC-SAT [Richardson and Domingos, 2006]. However, the PLL parameters may lead to poor results when long chains of inference are required [Richardson and Domingos, 2006]. To overcome this issue, we sort candidate clauses in *CC* in the decreasing order of measure of frequency and then we eliminate some of them (*line 7*) using two criteria as follows:

• **Sub-clause:** remove every clause $CC[j]$ when there exists a clause $CC[i], i < j$ such that $CC[i]$ and $CC[j]$ have a sub-clause relation[1]. For example, in case $len(CC[i]) < len(CC[j])$ and there exists a sub-clause *c* of *CC[j]* and a variable replacement $\theta$ such that $c\theta \equiv CC[i]$, we have a sub-clause relation between *CC[i]* and *CC[j]*.

• **Clauses of same predicates:** keep only one (with the maximum measure) of clauses composed of the same set of predicates. For example, we choose only one of the 2 clauses (*student(A)* $\vee$ *advisedBy(A,B)*, *student(A)* $\vee$ *advisedBy(B,A)*) consisting of the two predicates *student* and *advisedBy*.

Clauses in *CC* are added into the MLN in turn. A candidate clause is put into the MLN as long as it leads to an improvement of the performance measure (i.e. WPLL) of the MLN learned before. As adding clauses into the MLN can influence the weight of formerly added clauses, once all candidate clauses have been considered, we try to prune some of the clauses from the MLN. We use a zero-mean on each weight in order to remove a set of clauses, whose weights are less than a given threshold *mw*, as long as their elimination causes an increase of the performance measure. This step is iterated until no clause can be removed any more.

[1]See online appendix for more details http://www.univ-orleans.fr/lifo/Members/Dinh.Thang/

| DATASET | IMDB | UW-CSE | CORA | ML1 | ML2 | MT1 | MT2 |
|---|---|---|---|---|---|---|---|
| TYPE | 4 | 9 | 5 | 9 | 9 | 11 | 11 |
| CONSTANTS | 316 | 1323 | 3079 | 234 | 634 | 696 | 1122 |
| PREDICATES | 10 | 15 | 10 | 10 | 10 | 13 | 13 |
| TRUE ATOMS | 1540 | 2673 | 70367 | 3485 | 27134 | 5868 | 9058 |

Table 2: Details of datasets

## 4 Experiments

GSLP is implemented over the Alchemy[1] package. We perform experiments to estimate the performance of GSLP following two questions:

1. How does GSLP compare to the state of the art generative systems for MLN structure learning?
2. Can we compare GSLP to a state of the art discriminative structure learning system for specific predicates?

### 4.1 Systems, Datasets and Methodology

To answer question 1, we choose three state-of-the-art generative systems:

• LSM [Kok and Domingos, 2010]: This algorithm uses *random walks* to identify densely connected objects in data, groups them with their associated relations into a motif, and then constrains the search for clauses to occur within motifs.

• HGSM [Dinh *et al.*, 2010b]: This algorithm heuristically builds a set of variable literals then finds dependent relations amongst them. Candidate clauses are created from sets of dependent variable literals.

• MBN [Khosravi *et al.*, 2010]: This algorithm first learns a (parametrized) Bayes net from which to apply a standard moralization technique in order to produce a MLN.

For question 2, we choose the DMSP [Dinh *et al.*, 2010a], a recent discriminative MLN structure learning system.

We used the three datasets IMDB, UW-CSE, CORA[2] that have been used to evaluate LSM, HGSM as well as DMSP. For each domain, we also performed a *5-fold cross-validation* as in previous studies. Unfortunately, this approach is not feasible for the current setting of MBN [Khosravi *et al.*, 2010]. To be able to compare to MBN, we used 2 subsample datasets (ML1, ML2) of the MovieLens database and 2 subsample datasets (MT1, MT2) of the Mutagenesis database[3]. For each

[2]Publicly-available at http://alchemy.cs.washington.edu. IMDB describes a movie domain, UW-CSE describes an academic department, and CORA is a collection of citations of CS papers.

[3]MovieLens is available at the UC Irvine machine learning repository and Mutagenesis is widely used in ILP research.

| DATASET | GSLP | | | LSM | | | HGSM | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | CLL | AUC | TIME | CLL | AUC | TIME | CLL | AUC | TIME | | |
| IMDB | **-0.099±0.03** | **0.790± 0.06** | **0.43** | -0.191±0.02 | 0.714±0.06 | 1.23 | -0.183±0.03 | 0.692±0.07 | 3.06 | | |
| UW-CSE | **-0.052±0.06** | **0.459± 0.07** | 3.05 | -0.068±0.07 | 0.426±0.07 | **2.88** | -0.103±0.04 | 0.311±0.02 | 8.68 | MBN | |
| CORA | **-0.054±0.05** | **0.887± 0.07** | 6.72 | -0.065±0.02 | 0.803±0.08 | **6.05** | -0.087±0.05 | 0.762±0.06 | 64.15 | CLL | AUC |
| MOVILENS1 | **-0.764** | **0.701** | **0.44** | -0.933 | 0.683 | 0.46 | -1.010 | 0.611 | 4.02 | -0.99 | 0.64 |
| MOVILENS2 | **-0.915** | **0.677** | 3.18 | -0.985 | 0.658 | **2.97** | NA | NA | NA | -1.15 | 0.62 |
| MUTAGENESIS1 | **-1.689** | **0.766** | **1.13** | -1.855 | 0.713 | 1.25 | -2.031 | 0.692 | 4.37 | -2.44 | 0.69 |
| MUTAGENESIS2 | **-1.754** | **0.743** | 8.28 | NA | NA | NA | NA | NA | NA | -2.36 | 0.73 |

Table 3: CLL, AUC measures and runtimes (hours) (generative)

of them, we learned the MLN on 2/3 random selected atoms and tested it on the remaining 1/3, as presented in [Khosravi *et al.*, 2010]. Details of these datasets are reported in Table 2.

Performance was measured in terms of CLL and AUC (area under precision recall curve). The CLL of a ground atom is its log-probability given the MLN and the database, and directly measures the quality of the estimated probabilities. AUC curves are computed by varying the CLL threshold above which a ground atom is predicted true. AUC is insensitive to a large number of true negatives. For these two measures, reported values are averaged over all predicates. CLL and AUC have been used in most of recent studies of MLN learning and as in them, we used the standard MC-SAT inference algorithm to compute a probability for every grounding of the query predicate on the test fold. We used the package provided in [Davis and Goadrich, 2006] to compute AUC.

Parameters for LSM, HGSM and DMSP were respectively set as in the papers of Kok and Domingos [2010] and Dinh et al. [2010b; 2010a]. We learned clauses composed of at most 5 literals (*maxLength = 5*) for all systems and limit the number of similar predicates per clause to 3 in order to learn weights in a reasonable time. We set *nti = 0*, *mft = 0*, *mw = 0.001*, *penalty = 0.001* for all domains. These parameters are chosen just to estimate the performance of GSLP; we did not optimize them for any dataset yet. We ran these experiments on a Dual-core AMD 2.4 GHz CPU - 4GB RAM machine.

## 4.2 Results

Table 3 reports the average CLLs, AUCs and runtimes on each dataset. Higher numbers indicate better performance and NA indicates that the system was not able to return a MLN, because of either crashing or timing out after 3 days running. For MBN, we used the values published in [Khosravi *et al.*, 2010]. It must be noted that, our results do slightly differ from the ones in [Kok and Domingos, 2010], as i) we used the standard versions of IMDB and UW-CSE instead of the ones reduced by Kok for LHL and LSM, and ii) for CORA, we learned clauses with at most 5 predicates instead of 4. Besides, since we aim at comparing two generative learning systems, after learning the MLN structure using LSM, we only learned the weights once instead of relearning them for each predicate before performing inference.

For the largest dataset CORA, GSLP increases 16.9% on CLL, 10.5% on AUC compared to LSM and 37.9% on CLL, 16.4% on AUC compared to HGSM. It dominates MBN on CLL in the range of 20-30%. Improvement not only occurs on average values over test folds but also on most of average values over predicates in each test fold. Since CLL determines the quality of the probability output by the algorithms, GSLP improves the ability to predict correctly the query predicates given evidence. As AUC is insensitive to a large number of true negatives, GSLP enhances the ability to predict a few positives in data. GSLP runs much faster than HGSM, especially for CORA, but performs a bit slower than LSM. We can answer question 1 that GSLP performs better, on these datasets, than the state-of-the art systems in terms of both CLL and AUC. However at the predicate level, LSM, MBN sometimes achieve better values than GSLP. We provide the average CLLs, AUCs for every predicate and examples of clauses learned by GSLP in the online appendix.

| ALGORITHMS → | | DMSP | | GSLP | |
|---|---|---|---|---|---|
| DB | PREDICATE | CLL | AUC | CLL | AUC |
| IMDB | WORKEDUNDER | **-0.022** | 0.382 | -0.023 | **0.394** |
| UW-CSE | ADVISEDBY | **-0.016** | 0.264 | **-0.016** | **0.356** |
| CORA | SAMEBIB | **-0.136** | 0.540 | -0.139 | **0.669** |
| | SAMETITLE | **-0.085** | 0.624 | -0.090 | **0.764** |
| | SAMEAUTHOR | **-0.132** | 0.619 | -0.132 | **0.995** |
| | SAMEVENUE | -0.109 | 0.475 | **-0.107** | **0.572** |

Table 4: CLL, AUC measures (discriminative)

Comparing GSLP to DMSP, we used DMSP to learn discriminatively MLN structure for predicates WorkedUnder (IMDB), Advisor (UW-CSE) and SameBib, SameTitle, SameAuthor, SameVenue (CORA). These predicates are often used in previous studies for MLN discriminative learning [Dinh *et al.*, 2010a; Biba *et al.*, 2008a]. Table 4 gives the average CLLs, AUCs on those predicates over all test folds for each dataset. The results show that GSLP produces much better AUCs than DMSP did while it loses a litte CLLs. Indeed, it is worst 5% on CLL on predicate SameTitle while the AUC is increased 18.32%. The improvement of AUC reaches at most 37.78% on predicate SameAuthor while the CLL is equal. GSLP gets better results than DMSP in terms of both CLL (1.87%) and AUC (16.96%) on the predicate SameVenue. GSLP is thus competitive to the discriminative system in terms of CLL and dominates it in terms of AUC.

# 5 Related Works

Structure graphs have been widely used in SRL, including both *directed* graph models [Neville and Jensen, 2004; Friedman *et al.*, 1999; Kersting and De Raedt, 2007] and *undirected* graph models [Anderson *et al.*, 2002; Kok and Domingos, 2009]. Our graph of predicates (GoP) although different can be related to PRM [Friedman *et al.*, 1999],

which relies on a class dependency directed graph, where each node corresponds to an attribute of some class (or table in a relational database).

Concerning MLN structure learning, the idea to encode database information into a graph has been applied in LHL and LSM by Kok and Domingos [2009; 2010], where relational path-finding [Richards and Mooney, 1992] is used to find paths of ground atoms, then lift them into an undirected "lifted-graph". This latter, however, is very different from GoP; a node corresponds to a set of constants instead of a predicate (or its negation) and an edge corresponds to a predicate instead of a link between predicates. Based on this GoP, our method shows some advantages compared to LHL:

• Building GoP is only based on the types of predicate arguments, which is much faster than tracing all paths of ground atoms. The size (number of nodes and edges) of GoP is hence "fixed" while the one of lifted-graphs varies and depends mostly on the database.

• Negative and positive literals are handled in the same way and only good links are used to extend good candidates seeking for better ones. A path in GoP, holding information on predicates and positions of shared-arguments, is a general description of paths of ground atoms. LHL has to search for all paths of ground atoms, each one being variabilized to create all possible clauses by flipping sign of its literals. This, more or less, still leads to a whole graph-search. The search space in our method is thus greatly reduced compared to LHL.

## 6 Conclusion

We propose a new algorithm for generative MLN structure learning. Our first experiments show that it outperforms the state-of-the art approaches on some classical benchmarks. Nevertheless, beside these satisfying results, we have identified some points, which must be investigated further:

• Our algorithm works in a generate-and-test manner, runtime is thus mostly spent for evaluating clauses. A strategy to generate less clauses would improve performance.

• If we keep clauses built with the same predicates (Subsection 3.3), then we get better performance (in terms of WPLL). The balance between performance, tractability and runtime will be an interesting point to study.

• Based on the graph of predicates, GSLP should be able to handle more complex domains with more predicates and many associations between them. More experiments need to be conducted to verify this remark.

## Acknowledgments

We would like to thank our reviewers for their valuable comments. This work was partly funded by Région Centre, France.

## References

[Anderson *et al.*, 2002] Corin R. Anderson, Pedro Domingos, and Daniel S. Weld. RMM and their application to adaptive web navigation. In *KDD '02*. ACM, 2002.

[Biba *et al.*, 2008a] Marenglen Biba, Stefano Ferilli, and Floriana Esposito. Discriminative structure learning of markov logic networks. In *ILP '08*. Springer-Verlag, 2008.

[Biba *et al.*, 2008b] Marenglen Biba, Stefano Ferilli, and Floriana Esposito. Structure learning of mln through iterated local search. In *ECAI '08*. IOS Press, 2008.

[Davis and Goadrich, 2006] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *ICML '06*, pages 233–240. ACM, 2006.

[Dinh *et al.*, 2010a] Quang-Thang Dinh, Matthieu Exbrayat, and Christel Vrain. Discriminative markov logic network structure learning based on propositionalization and $\chi^2$-test. In *ADMA*, 2010.

[Dinh *et al.*, 2010b] Quang-Thang Dinh, Matthieu Exbrayat, and Christel Vrain. Generative structure learning for markov logic networks. In *STAIRS '10*. IOS Press, 2010.

[Friedman *et al.*, 1999] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI '99*, pages 1300–1309. Springer, 1999.

[Huynh and Mooney, 2008] Tuyen N. Huynh and Raymond J. Mooney. Discriminative structure and parameter learning for MLNs. In *ICML '08*. ACM, 2008.

[Kersting and De Raedt, 2007] Kristian Kersting and Luc De Raedt. *Bayesian Logic Programming: Theory and Tool*. In Intro. to Statistical Relational Learning, 2007.

[Khosravi *et al.*, 2010] Hassan Khosravi, Oliver Schulte, Tong Man, Xiaoyuan Xu, and Bahareh Bina. Structure learning for MLNs with many descriptive attributes. In *ICML-10*, 2010.

[Kok and Domingos, 2005] Stanley Kok and Pedro Domingos. Learning the structure of MLNs. In *ICML '05*, 2005.

[Kok and Domingos, 2009] Stanley Kok and Pedro Domingos. Learning markov logic network structure via hypergraph lifting. In *ICML*, 2009.

[Kok and Domingos, 2010] Stanley Kok and Pedro Domingos. Learning MLNs Using Structural Motifs. In *ICML*, 2010.

[Mihalkova and Mooney, 2007] Lilyana Mihalkova and Raymond J. Mooney. Bottom-up learning of markov logic network structure. In *ICML '07*. ACM, 2007.

[Neville and Jensen, 2004] Jennifer Neville and David Jensen. Dependency networks for relational data. In *ICDM '04*. IEEE Computer Society, 2004.

[Pearl, 1988] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., 1988.

[Richards and Mooney, 1992] B. L. Richards and R. J. Mooney. Learning relations by pathfinding. In *Proc. of AAAI-92*, pages 50–55, San Jose, CA, 1992.

[Richardson and Domingos, 2006] Matthew Richardson and Pedro Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, 2006.

[Sato and Kameya, 1997] Taisuke Sato and Yoshitaka Kameya. Prism: a language for symbolic-statistical modeling. In *IJCAI'97*, pages 1330–1335, 1997.

[Sha and Pereira, 2003] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *NAACL '03*, pages 134–141, 2003.