Scalable Learning for Structure in Markov Logic Networks

Zhengya Sun, Zhuoyu Wei, Jue Wang, Hongwei Hao
Institute of Automation, Chinese Academy of Sciences
{zhengya.sun, zhuoyu.wei, jue.wang, hongwei.hao}@ia.ac.cn

Abstract

Markov Logic Networks (MLNs) provide a unifying framework that incorporates first-order logic and probability. However, learning the structure of MLNs is a computationally hard task due to the large search space and the intractable clause evaluation. In this paper, we propose a random walk-based approach to learn MLN structure in a scalable manner. It uses the interactions existing among the objects to constrain the search space of candidate clauses. Specifically, we obtain representative subset of simple paths by sampling from all sequences of distinct objects. We then transform each sampled path into possible ground atoms, and use them to form clauses. Based on the resulting ground network, we finally attach a set of weights to the clauses by optimizing ℓ_1-constrained conditional likelihood. The experimental results demonstrate that our approach performs favorably compared to previous approaches.

Introduction

Markov logic networks (MLNs) have gained wide attention in recent years due to their ability to unify the strengths of first-order logic and probabilistic graphical models (Richardson and Domingos 2006). An MLN typically attaches weights to first-order clauses and views these as templates for Markov network features. Learning MLN structure is the process of generating clauses together with their weights (Kok and Domingos 2005), which allows us to capture uncertain dependencies underlying the relational data. Recent advances have revealed its superiority in dealing with real world data, such as biochemistry(Huynh and Mooney 2008), social network and text (Domingos and Lowd 2009).

The structure learning problem, although important, can be computationally prohibitive with the continuous increase in data size. Not only is the search space super-exponentially large, the clause evaluation also produces an overwhelming number of groundings. That is, the exhaustive search and full grounding may fail to finish in a reasonable amount of time. Most efforts are channeled towards constraining the space of candidate structures in a top-down or bottom-up manner.

Top-down approaches (Kok and Domingos 2005; Biba, Ferilli, and Esposito 2008) follow a generate-and-test strategy in which each clause is generated recursively, requiring that the newly added atom shares at least one variable with the existing ones, and then tested for the empirical efficacy. Such approaches are inefficient as they test many potential revisions with no support in the data, and are susceptible to local optima. Bottom-up approaches (Mihalkova and Mooney 2007) overcome these limitations by using the training data to guide their search for clause candidates. The relational pathfinding serves this purpose, which finds paths of true ground atoms, and then generalizes them into first-order clauses. The number of possible paths is often decreased by inducing high level representation (Kok and Domingos 2009; 2010), which still do not scale well to the ever-growing datasets. Meanwhile, evaluating each clause requires producing a grounded Markov network with the variables fully instantiated. This may be infeasible in the presence of numerous objects, which prevents a scalable structure learner.

In this paper, we present *Learning via ground Network Sampling(LNS)*, a novel MLN structure learning approach based on Markov Chain Monte Carlo (MCMC) simulation. The basic idea is to obtain representative groundings of logical statements for clause generation and evaluation. LNS starts by converting the unary atoms (i.e., an atom that involves only one argument) into binary ones, so that the already-present ground atoms can all be treated as interactions between their arguments. The resulting argument pairs that share common objects are then concatenated in order of increasing length, hence forming a partial order graph (POG). Every node in the POG corresponds to a simple path, i.e., a sequence of distinct connected objects, which may describe whether or not the initial and terminal objects interact with each other. Subsequently, LNS performs a random walk on the partial order graph with a prescribed transition probability matrix, and returns every unique simple path it visits. Furthermore, LNS creates the ground clauses from each sampled path by transforming the meaningful interactions into ground atoms. Based on the sampled ground network, LNS finally performs discriminative weight learning with ℓ_1-norm constraint to encourage sparse solutions.

LNS combines the benefits of random walk and subgraph pattern mining in formulating the problem of structure learn-

ing for MLNs. In the clause generation stage, the approach constrains the search space according to the connections existing among objects. In this way, LNS only constructs clauses that have support in the data, while retaining both true and false instantiated clause samples. In the clause evaluation stage, the approach directly exploits these samples to learn a set of weights, avoiding enumeration of every grounding of the candidate clauses.

The remainder of this paper is organized as follows. In the next section, we begin by providing some background on MLNs and briefly defining some terms we use. Then we present our structure learning approach (Section 3) and report the experimental results (Section 4). Finally, we summarize the conclusion of this work (Section 5).

Markov Logic Networks

This section provides some background on representation, structure and discriminative weight learning for Markov logic networks. Let us briefly review some basic terminology which will be used throughout this paper. Predicates represent relations among objects or attributes of objects. An atom is a predicate symbol applied to a tuple of arguments. A positive literal is an atom, and a negative literal is a negated atom. A ground atom is an atom all of whose arguments are constants. A formula in clausal form, also called a clause, is a disjunction of positive/negative literals. A world is an assignment of truth values to all possible ground atoms.

Representation

Markov logic can be viewed as a probabilistic extension of first-order logic by attaching weights to formulas (clauses). The magnitude of each weight reflects the relative strength or importance of the corresponding clause. The higher weight indicates the greater reward to a world that satisfies the clause, with other things being equal. In the infinite-weight limit, Markov logic becomes equivalent to standard first-order logic.

Definition 1. *(Richardson and Domingos 2006) A Markov logic network (MLN) L is a set of pairs (f_i, w_i), where f_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ as follows:*

1. *$M_{L,C}$ contains one binary node for each possible grounding of each atom appearing in L. The value of the node is 1 if the ground atom is true, and 0 otherwise.*
2. *$M_{L,C}$ contains one feature for each possible grounding of each formula f_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with f_i in L.*

More formally, let X be the set of all ground atoms, $\mathcal{F}$ be the set of all first-order clauses in the MLN, w_i be the weight associated with clause $f_i \in \mathcal{F}$, $\mathcal{G}_{f_i}$ be the set of all possible groundings of clause f_i with the constants in the domain. Then, the probability of a possible world x is defined as:

$$P_w(X = x) = \frac{1}{Z} exp\Big(\sum_{f_i \in \mathcal{F}} w_i \sum_{g \in \mathcal{G}_{f_i}} g(x)\Big) = \frac{1}{Z} exp\Big(\sum_{f_i \in \mathcal{F}} w_i n_i(x)\Big) \quad (1)$$

where the subscript w represents a set of clause weights, $g(x)$ equals 1 if g is satisfied and 0 otherwise, $n_i(x)$ denotes the number of true groundings of f_i in x, and $Z = \sum_{x' \in \mathcal{X}} exp\Big(\sum_{f_i \in \mathcal{F}} w_i n_i(x)\Big)$ is the normalizing constant.

Structure Learning

The main task in MLN structure learning is to find a set of potentially good clauses. This allows us to represent complex dependencies among atoms, avoiding the assumption of i.i.d. (independent and identically distributed) data made by most statistical learners. Current solutions primarily focus on connected atoms that share common arguments (e.g., variables or constants) and produce all possible clauses consistent with them. In most cases, there is an exponential number of paths that connect the arguments of the given atoms. Thus, approximate search techniques are further required, including scaling down the size of input atoms by clustering (Kok and Domingos 2009; 2010) or restricting the candidate paths to a particular mode (Huynh and Mooney 2011). All candidate clauses are then fully instantiated and evaluated using weighted pseudo-log-likelihood (WPLL). Candidates that do not improve the overall WPLL of the learned structure are discarded.

Discriminative Weight Learning

In discriminative learning, we know a priori which atoms will be evidence and which ones will be queried, and the goal is to correctly predict the latter given the former. Given a set of clauses, weight learners try to maximize the conditional likelihood of query atoms Y given evidence ones X,

$$P_w(Y = y|X = x) = \frac{1}{Z_x} exp(\sum_{i \in \mathcal{F}_Y} w_i n_i(x, y)) \quad (2)$$

where $\mathcal{F}_Y$ denotes the set of all MLN clauses with at least one grounding involving a query atom, $n_i(x, y)$ is the number of true groundings of the ith clause involving query atoms. When non-recursive clauses are considered, the query atoms become independent given the evidence. The conditional log-likelihood (CLL) in this case becomes

$$\log P_w(Y = y|X = x) = \log \prod_{j=1}^{n} P_w(Y_j = y_j|X = x) = \sum_{j=1}^{n} \log P_w(Y_j = y_j|X = x) \quad (3)$$

and,

$$P_w(Y_j = y_j|X = x) = \frac{exp(\sum_{i\in\mathcal{F}_{Y_j}} w_i n_i(x, y_{[Y_j=y_j]}))}{exp(\sum_{i\in\mathcal{F}_{Y_j}} w_i n_i(x, y_{[Y_j=0]})) + exp(\sum_{i\in\mathcal{F}_{Y_j}} w_i n_i(x, y_{[Y_j=1]}))} \quad (4)$$

where $n_i(x, y_{[Y_j=y_j]})$ is the number of true groundings of the ith clause when all the evidence atoms in X and the query atom Y_j are set to their truth values, and similarly for $n_i(x, y_{[Y_j=0]})$ and $n_i(x, y_{[Y_j=1]})$ when Y_j is set to 0 and 1 respectively. Then the derivative of the CLL with respect to the ith clause is

$$\frac{\partial}{\partial w_i} \log P_w(Y = y|X = x) = \sum_{j=1}^{n} [n_i(x, y_{[Y_j=y_j]}) - P_w(Y_j = 0|X = x) n_i(x, y_{[Y_j=0]}) - P_w(Y_j = 1|X = x) n_i(x, y_{[y_j=1]})]$$

Notice that the ith component of the gradient is simply the difference between the total count of true groundings of the ith clause and the expected count according to the current model. When the closed world assumption is made, all the counts can be computed exactly without approximate inference, since there is no evidence atom with unknown value.

Learning via Network Sampling

We now describe LNS (Learning via ground Network Sampling), a scalable approach for MLN structure learning. LNS performs a random walk search by starting with each observed ground atom as interactions between its arguments. It then creates a clause to match the sequence of connected objects by labeling predicates. In this paper, we only consider definite clauses containing exactly one positive atom, but the extension to more general case is straightforward.

The four crucial components of LNS, introduced in the next subsections, are: (i) how to construct the state space for random walks, (ii) how to obtain simple path samples, (iii) how to generalize the clauses, and (iv) how the overall approach operates.

State Space of the Random Walk

The partial order graph (POG) of simple paths works as the state space of the Markov Chain on which LNS runs its simulation. We do this by first converting unary atoms (e.g., Student(Bob)) into binary ones augmented by new predicates (e.g., IsA(Bob, Student)). These transformations allow already-present ground atoms to be formulated as interactions between their arguments. Let $\mathcal{I}$ denote the set of all the possible pairwise interactions from the training data. Intuitively, the pairwise interactions that share common objects could be concatenated into paths describing whether or not the initial and terminal objects interact with each other. In our case, we are only interested in simple paths, i.e., a sequence of distinct connected objects, which are also used to restrict every node in the POG. This restriction is actually reasonable to avoid the propagation of redundant information. The length of a simple path is the number of edges in it. We then construct the partial order graphs locally around the current node. If the current node represents the simple path $p(s, t)$ joining the vertex s to t, its neighbors consist of nodes corresponding to all simple super-paths that expand $p(s, t)$ with one more vertex t' such that (t', s) or $(t, t') \in \mathcal{I}$, and sub-paths that removes the initial vertex s or the terminal one t from $p(s, t)$. The random walker chooses one neighbor (super- or sub- path) according to its transition probability (see below for details). If the current node is a new simple path that was not visited before, LNS returns the corresponding simple path for ground clause generation. The local construction of POG is essential as it circumvents the construction of the entire POG, which would require finding all the simple paths.

Simple Path Sampling

To achieve uniform distribution for all the simple paths, LNS exploits ideas from Metropolis-Hastings (MH) algorithm to perform random walks on the graph (Al Hasan and Zaki 2009). Consider a partial order graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, with a set of nodes $\mathcal{N}$ corresponding to distinct simple paths and a set of edges $\mathcal{E}$ indicating possible extensions of simple paths to larger simple paths. Let $N(u) = \{v \in \mathcal{N}|(u, v) \in \mathcal{E}\}$ denote the set of neighbors adjacent to the node $u \in \mathcal{N}$, and $d(u) = |N(u)|$ denote the degree of the node u. For any arbitrarily given target stationary distribution $\boldsymbol{\pi} = (\pi(u), u \in \mathcal{N})$, the Metropolis-Hastings random walk (MHRW) $\{X_t \in \mathcal{N}, t = 0, 1, \dots\}$ on the graph $\mathcal{G}$ operates as follows. At the current state u of X_t, the next state X_{t+1} is proposed with a proposal probability $Q(u, v)(u \neq v)$. Then, the proposed transition to v is accepted with an acceptance probability $A(u, v)$, that is,

$$A(u, v) = \min\left\{1, \frac{\pi(v)Q(v, u)}{\pi(u)Q(u, v)}\right\}, \quad (5)$$

and rejected with probability $1 - A(u, v)$ in which case the state X_{t+1} remains unchanged. Thus, the transition probability $P(u, v)$ becomes, for $u \neq v$,

$$P(u, v) = Q(u, v)A(u, v) = \min\left\{Q(u, v), Q(v, u)\frac{\pi(v)}{\pi(u)}\right\}, \quad (6)$$

with $P(u, u) = 1 - \sum_{u\neq v} P(u, v)$.

For unbiased graph sampling, the target stationary distribution should be set to the uniform distribution $\boldsymbol{\pi} = (1/n, 1/n, \dots, 1/n)$, where $n = |\mathcal{N}|$ represents the number of possible states. Assuming that $N_+(u)$ and $N_-(u)$ denote the set of super- and sub- path neighbors of the node u, with $d_+(u) = |N_+(u)|$ and $d_-(u) = |N_-(u)|$ being the set size correspondingly, LNS then chooses the proposal probabilities for all $(u, v) \in \mathcal{E}$ as follows:

$$Q(u, v) = \begin{cases} \frac{1}{2d_+(u)}, & \text{if } v \in N_+(u) \\ \frac{1}{2d_-(u)}, & \text{if } v \in N_-(u) \end{cases} \quad (7)$$

Observe that any node in the POG as defined earlier has no more than two sub-path neighbors, while with a larger

number of super-path neighbors in most cases. For this, the proposal probability given by (7) can reduce the bias towards the nodes with longer simple paths, and increase the convergence rate of the sampling procedure. The resulting transition probability is thus given as follows::

$$P(u,v) = \begin{cases} \min\{\frac{1}{2d_+(u)}, \frac{1}{2d_-(v)}\}, \text{if } (u,v) \in \mathcal{E}, v \in N_+(u), \\ \min\{\frac{1}{2d_-(u)}, \frac{1}{2d_+(v)}\}, \text{if } (u,v) \in \mathcal{E}, v \in N_-(u), \\ 0, \qquad\qquad\qquad \text{if } (u,v) \notin \mathcal{E}, u \neq v. \end{cases}$$

This ensures that $\boldsymbol{P} = \{P(u,v)\}_{(u,v)\in\mathcal{E}}$ is reversible with respect to $\boldsymbol{\pi}$, i.e., $\pi(u)P(u,v) = \pi(v)P(v,u)$ for any u, v. Since there exists a positive probability to reach from one simple path to another via the $\emptyset$ path, $\boldsymbol{P}$ is irreducible. In this way, the uniqueness of $\boldsymbol{\pi}$ is granted due to the finite state space.

Note that when the POG of sampling simple path generators is large, the random walker may get stuck in local regions of the graph. Following the strategy validated by (Li and Zaki 2012), we set a non-zero probability that makes the walker jump back to the root node (empty path) at each step.

Clause Creation and Pruning

LNS constructs the ground clauses from each sampled path by transforming the meaningful interactions into ground atoms. Note the fact that a definite clause is logically equivalent to an implication where the antecedents are a conjunction of positive literals, and the conclusion is a single positive literal (e.g., the clause $\neg R \vee \neg S \vee T$ can be rewritten as the implication $R \wedge S \Rightarrow T$). By analogy, a path here can be interpreted as a statement about whether a conjunction of interactions between two adjacent objects logically implies the interaction between the initial and terminal objects. In addition, as already pointed out, the pairwise interaction corresponds to a set of ground atoms which share the two interacting objects in their arguments. Here we assume that the ground atom contains at least two arguments. It is thus intuitive to convert a path into a set of ground clauses. More specifically, the antecedents are formed as a conjunction of ground atoms that sequentially connect adjacent objects, the ground atom connecting the initial and terminal objects becomes the conclusion.

To control the size of the ground network, LNS imposes restrictions on the search space. On one side, the algorithm starts with the already-present ground atoms, and creates the true antecedents for each clause. Note that any false antecedents alone can make a clause trivially satisfied, and need not to be considered here. On the other side, the algorithm draws conclusions from all the possible atoms with both true and false groundings provided. Also note that the majority of conclusions are false due to the sparseness of relational data (i.e., the great majority of atoms are false), leading to the severe imbalance between the true and false clauses. The algorithm then tackles the imbalance ratio by randomly sampling a predefined number of false clauses. When discriminative learning is concerned, the algorithm variabilizes the resulting ground clauses that have specific query atoms, and discards those without any support in their groundings (i.e., there is at least one true grounding in the network). In our case, the query atom may appear in the antecedents or the conclusion of a clause. Following Kok and Domingos (2009), the clauses containing variables that appear only once are ignored, since they might not be useful.

Given a set of clauses and a sampled ground network, one can easily find a set of weights that maximizes the CLL of query atoms. More precisely, the query atoms here are the sampled groundings of the target predicate. However, to prevent overfitting and enforce sparsity, LNS performs the optimization subject to an ℓ_1 constraint on the norm of the solution. The final objective function can be written as

$$\min_w \sum_{i=1}^{n} -\log P_w(Y_i = y_i | X = x), \qquad (8)$$
$$\text{subject to} \quad \|w\|_1 \leq \beta.$$

where $\beta > 0$ is a tunable parameter and need to be fixed in advance. In order to solve (8), LNS applies a state-of-the-art ℓ_1-ball projection algorithm (Duchi et al. 2008) which implements the first-order projected gradient descent update. The clauses with zero weights are finally pruned away.

LNS Overview

The LNS approach proceeds in two phases. The first step, summarized in Algorithm 1, samples a set of simple paths. The relational database $\mathcal{D}$ is converted to the state space for random walks. Then a simple path limited to l is randomly selected. The procedure terminates after KT iterations.

Algorithm 1 Simple Path Sampling$(\mathcal{D}, K, l, \gamma)$

Input: $\mathcal{D}$, a relational database
K, the number of initial paths
l, maximal length of each path
γ, restart probability to the empty path

Repeat for $k = 1, 2, \ldots, K$
  Start with any simple path p_0^k as the current state
  Repeat for iteration $t = 1, 2, \ldots, T$
    · Construct the neighbor paths of p_{t-1}^k, with each path no longer than l
    · Propose the next path $p_{cand}^k \sim Q(p_{t-1}^k, p_t^k)$
    · Generate $q \sim U(0,1)$
    · Compute the acceptance probability
$$\alpha = \min\Big\{1, \frac{Q(p_{cand}^k, p_{t-1}^k)}{Q(p_{t-1}^k, p_{cand}^k)}\Big\}$$
    · Set the next path p_t^k as
      (a) if $q \leq \alpha$, then $p_t^k = p_{cand}^k$
      (b) else if $\alpha < q \leq 1 - \gamma$, then $p_t^k = p_{t-1}^k$
      (c) else $p_t^k = \emptyset$

Output: Set of simple paths PS

The second step, summarized in Algorithm 2, creates a set of weighted clauses. Under certain constraints, the PS is translated into a set of clauses together with their groundings. The time-complexity in the worst case is $O(mKT)$. LNS finally attaches a set of sparse weights to the clauses based on ℓ_1-constrained optimization.

Algorithm 2 Clause Creation$(\mathcal{D}, PS, m)$

Input: $\mathcal{D}$, a relational database
PS, a set of simple paths
m, maximal number of sampled false clauses for a certain path

Initialize a set of ground clauses $GS = \emptyset$
Repeat for all simple paths p in PS
· $GS' \leftarrow CreateTrueClauses(\mathcal{D}, p)$
· if $GS' = \emptyset$, then
$GS' \leftarrow SampleFalseClauses(\mathcal{D}, p, m)$
· Remove clauses without query atoms GS'
· Add GS' to GS
$VS \leftarrow Variabilize(GS)$
$CS \leftarrow LearnSparseWeights(VS, GS)$
Output: Set of weighted clauses CS

Experimental Evaluation

Datasets

We use three publicly available datasets to evaluate the proposed approach. Table 1 shows the statistics of each dataset. The UW-CSE dataset, used by Richardson and Domingos (2006), describes facts about people in an academic department (e.g., Student, Professor) and their relationships (e.g., AdvisedBy, hasPosition). It is divided into five folds based on the areas of computer science. The WebKB dataset, processed by Mihalkova and Mooney (2007), contains information about the webpage categories and links between these pages. It includes four folds, each of which corresponds to one university. The Wordnet dataset, filtered by Richard Socher et al. (2013), provides ontological and lexical knowledge in common domains. It is given in one predefined split into training and test triplets.

Table 1: Statistics on datasets.

Dataset	Predicates	True literals	Total literals
UW-CSE	12	2112	260,254
WebKB	6	2065	688,193
Wordnet	11	133,669	$1,488,146,352$

Methodology

We performed two sets of experiments. The first experiment compared our approach with two baseline approaches, i.e., LSM-S and LSM-L, which provide the state-of-the-art solutions on relatively large datasets. The suffix '-S' and '-L' indicate the approaches with the short and long limit on clause length. We set the short limit to 5, and the long to 10. In the second experiment, we examined the influence of several sampling parameters on the overall performance of LNS.

The LNS parameter values were: $K = 20000, T = 50, \gamma = 0.1, m = 5, \beta = 10$. For LSM, we used the parameter settings provided by Kok and Domingos (2010). We run all the experiments on the identically configured servers (2.0GHz, 1T RAM, 18432KB CPU cache) for a maximum of 24-hour time limit during training.

To evaluate the performance of different structure learners, we implemented Alchemy's MC-SAT inference in MLNs over the groundings of each query predicate in the test set, using the groundings of all other predicates as evidence. Note that one MLN is tailored to each prediction task. We then chose the AUC-PR (Area under the Precision-Recall curve), CLL and runtime as the evaluation criteria. Since the CLL can be misleading when the number of negative examples overwhelms the number of positives (which happens in most cases), we kept twice the number of negative examples as positives in each test fold. For the calculation of AUC we used the package of (Davis and Goadrich 2006).

Results

Table 2 and 3 report the AUCs, CLLs and runtimes for different query predicates: advisedBy and tempAdvisedBy for the UW-CSE dataset, courseProf and courseTA for WebKB. The AUC and CLL results are averaged over all atoms in the test sets, while the runtimes are averaged over all the folds.

The tables below show that LNS consistently outperforms LSM in terms of the runtime when learning both short and long clauses. Specifically, the improved efficiency of LNS over LSM is by at least two orders of magnitude, and is independent of the clause length. As can be seen, LNS-S (with short clause limit) and LNS-L (with long clause limit) have similar runtimes on both datasets, with the latter even marginally faster than the former.

In addition to the runtime benefits, LNS also obtains competitive (often the best) AUC and CLL values. Comparing LNS-S to LNS-L, we see that long clauses are effective in lifting both evaluation criteria for most query predicates (with the exception of slightly lower CLL for advisedBy). This is not surprising because long clauses can capture more complex dependencies than short ones. By comparison with the LNS, LSM shows no better performance on all the predicates. For advisedBy, LSM-S has comparable AUC values to our structure learning approaches, while it is significantly worse on all the other predicates.

We examine the influence of the sampling parameters K and m for LNS on Wordnet, a much larger scale dataset. We do not report LSM on this testbed as its runtime exceeds the maximal time limit. Figure 1 plots the curves on AUC and runtime versus logarithmic number of initial paths K. As can be seen, the AUC values increase quickly within a certain interval, and then seem to attain a local optimum. Figure 2 plots the curves on CLL and runtime versus logarithmic number of false clauses m. This shows that the CLL values improve as the maximal number of sampled false clauses increases. The two figures both validate the fact that the time-complexity of LNS is linear related to either K or m.

Conclusion

In this paper, we developed a scalable MLN structure learning approach by combining the random walk with subgraph pattern mining. LNS restricts its search for clauses to within the sampled ground network. LNS then learns a set of sparse weights discriminatively. Our empirical studies on three datasets justified the effectiveness of LNS. Directions for future work include: sampling simple paths discriminatively, integrating more general clause forms, extending the

Table 2: Results on the UW-CSE datasets.

	AUC-PR		CLL		Runtime	
Approach	advisedBy	tempAdvisedBy	advisedBy	tempAdvisedBy	advisedBy	tempAdvisedBy
LNS-S	0.745±0.159	0.746±0.164	-0.602±0.015	-0.665±0.075	37.960±0.838 sec	38.960±0.783 sec
LNS-L	0.752±0.152	0.749±0.166	-0.606±0.015	-0.639±0.040	35.380±0.217 sec	33.680±0.572 sec
LSM-S	0.685±0.070	0.398±0.112	-1.590±0.017	-2.438±0.509	1.532±0.461 hr	1.532±0.461 hr
LSM-L	0.585±0.196	0.398±0.112	-1.676±0.181	-2.438±0.509	8.540±0.900 hr	8.540±0.900 hr

Table 3: Results on the WebKB datasets.

	AUC-PR		CLL		Runtime	
Approach	courseProf	courseTA	courseProf	courseTA	courseProf	courseTA
LNS-S	0.447±0.070	0.467±0.219	-0.919±0.168	-1.675±1.047	25.525±0.126 sec	26.450±0.507 sec
LNS-L	0.506±0.125	0.497±0.291	-0.837±0.133	-0.700±0.069	22.950±0.404 sec	23.500±1.065 sec
LSM-S	0.379±0.039	0.338±0.042	-2.315±0.007	-2.320±0.020	1.008±0.145 hr	1.008±0.145 hr
LSM-L	0.379±0.039	0.338±0.042	-2.315±0.007	-2.320±0.020	1.008±0.145 hr	1.008±0.145 hr

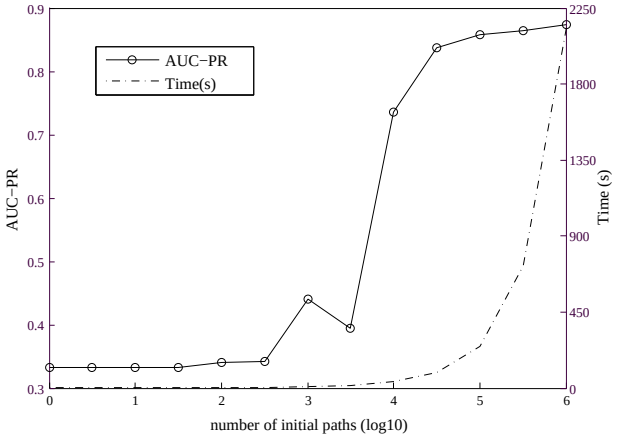

Figure 1: The curves of AUC-PR and runtime versus number of initial paths K on the Wordnet dataset

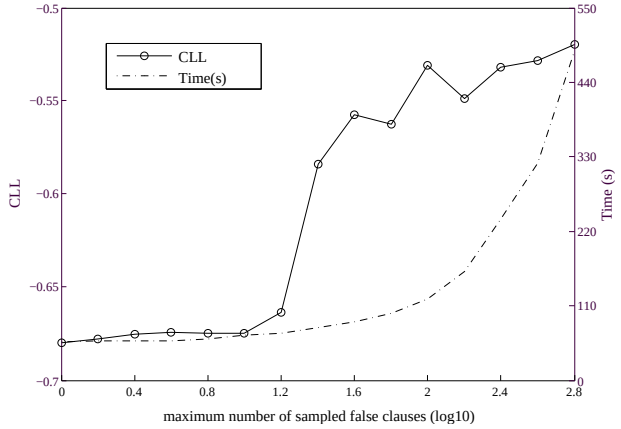

Figure 2: The curves of CLL and runtime versus maximal number of sampled false clauses m on the Wordnet dataset

sampling framework to scalable inference, and further applying LNS to larger domains.

Acknowledgments

The authors thank the anonymous reviewers for their valuable comments. This research work was funded by the National Natural Science Foundation of China under Grant No. 61303179 and No. U1135005.

References

Al Hasan, M., and Zaki, M. J. 2009. Output space sampling for graph patterns. 2(1):730–741.

Biba, M.; Ferilli, S.; and Esposito, F. 2008. Discriminative structure learning of markov logic networks. In *Inductive Logic Programming*. Springer. 59–76.

Davis, J., and Goadrich, M. 2006. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, 233–240. ACM.

Domingos, P., and Lowd, D. 2009. Markov logic: an interface layer for artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 3(1):1–155.

Duchi, J.; Shalev-Shwartz, S.; Singer, Y.; and Chandra, T. 2008. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, 272–279. ACM.

Huynh, T. N., and Mooney, R. J. 2008. Discriminative structure and parameter learning for markov logic networks. In *Proceedings of the 25th International Conference on Machine Learning*, 416–423. ACM.

Huynh, T. N., and Mooney, R. J. 2011. Online structure learning for markov logic networks. In *Machine Learning and Knowledge Discovery in Databases*. Springer. 81–96.

Kok, S., and Domingos, P. 2005. Learning the structure of markov logic networks. In *Proceedings of the 22 International Conference on Machine Learning*, 441–448. ACM.

Kok, S., and Domingos, P. 2009. Learning markov logic network structure via hypergraph lifting. In *Proceedings of*

the 26 International Conference on Machine Learning, 505–512. ACM.

Kok, S., and Domingos, P. 2010. Learning markov logic networks using structural motifs. In *Proceedings of the 27 International Conference on Machine Learning*, 551–558.

Li, G., and Zaki, M. J. 2012. Sampling minimal frequent boolean (dnf) patterns. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 87–95. ACM.

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. ACM.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62(1-2):107–136.