Modelling and Searching of Combinatorial Spaces Based on Markov Logic Networks

Marenglen Biba†, Stefano Ferilli‡ and Floriana Esposito‡
†Department of Computer Science, University of New York Tirana,
Rr. Komuna e Parisit, Tirana, Albania
‡Department of Computer Science, University of Bari,
Via E. Orabona, 4, 70125, Bari, Italy
{marenglenbiba @unyt.edu.al}, {ferilli,esposito} @di.uniba.it

Received: 22/02/2010; Accepted: 07/04/2010

ABSTRACT

Markov Logic Networks (MLNs) combine Markov networks (MNs) and first-order logic by attaching weights to first-order formulas and using these as templates for features of MNs. Learning the structure of MLNs is performed by state-of-the-art methods by maximizing the likelihood of a relational database. This leads to suboptimal results for prediction tasks due to the mismatch between the objective function (likelihood) and the task of classification (maximizing conditional likelihood (CL)).

In this paper we propose two algorithms for learning the structure of MLNs. The first maximizes the CL of query predicates instead of the joint likelihood of all predicates while the other maximizes the area under the Precision-Recall curve (AUC). Both algorithms set the parameters by maximum likelihood and choose structures by maximizing CL or AUC. For each of these algorithms we develop two different searching strategies. The first is based on Iterated Local Search and the second on Greedy Randomized Adaptive Search Procedure. We compare the performances of these randomized search approaches on real-world datasets and show that on larger datasets, the ILS-based approaches perform better, both in terms of CLL and AUC, while on small datasets, ILS and RBS approaches are competitive and RBS can also lead to better results for AUC.

1 INTRODUCTION

Many applications of artificial intelligence require both probability and first-order logic (FOL) to deal with uncertainty and structural complexity. For a long time, research in this area has followed two distinct lines: one based on logic

representations, and one on statistical ones. Logic approaches like logic programming, description logics, classical planning, etc, tend to handle complexity. Statistical approaches like Bayesian networks, hidden Markov models, Markov decision processes, etc, tend to handle uncertainty. However, intelligent systems must be able to deal with both for applications in the real world. The first attempts to integrate logic and statistics in artificial intelligence were the works in [1, 2]. Later, several authors began using logic programs to specify Bayesian networks [3].

Recently, different approaches for combining logic and probability have been proposed such as Probabilistic Relational Models [4], First-order Probabilistic Models with Combining Rules [5], Relational Dependency Networks [6], and others. The advantage of these models is that they can represent probabilistic dependencies between attributes of related different objects. Most of these approaches combine probabilistic graphical models with subsets of FOL (e.g., Horn Clauses). In this paper we focus on Markov Logic [7], a representation language that has finite FOL and probabilistic graphical models as subcases. It extends FOL by attaching weights to formulas providing full expressiveness as graphical models and FOL in finite domains and remaining well defined in many infinite domains [7, 8].

Learning an MLN consists in structure learning (learning the logical clauses) and weight learning (setting the weight of each clause). In [7] structure learning was performed through the rule-induction system CLAUDIEN [9] followed by a weight learning phase in which maximum pseudo-likelihood [10] weights were learned for each clause. In [11] structure is learned in a single phase using weighted pseudo-likelihood as the evaluation measure in a beam search. The algorithm performs systematic greedy search being therefore very suscetible to local optima. A state-of-the-art algorithm for generative structure learning is that in [12] which follows a bottom-up approach trying to consider fewer candidates for evaluation. This algorithm uses a "propositional" Markov network learning method to construct "template" networks that guide the construction of candidate clauses. In this way, it generates fewer clauses for evaluation. Recently, another generative algorithm was proposed in [13] that exploits randomized search guided by pseudo-likelihood.

Generative approaches optimize the joint distribution of all the variables. This can lead to suboptimal results for predictive tasks because of the mismatch between the objective function used (likelihood or a similar function based on it) and the goal of classification (maximizing accuracy or conditional likelihood). In contrast discriminative approaches maximize the conditional likelihood of a

set of outputs given a set of inputs [14] and this has often produced better results for prediction problems. In [15] the voted perceptron based algorithm for discriminative weight learning of MLNs was shown to greatly outperform maximum-likelihood and pseudo-likelihood approaches for two real-world prediction problems. Recently, the algorithm in [16], outperforming the voted perceptron became the state-of-the-art method for discriminative weight learning of MLNs. However, both discriminative approaches to MLNs learn weights for a fixed structure, given by a domain expert or learned through another structure learning method (usually generative). Better results could be achieved if the structure could be learned in a discriminative fashion.

Unfortunately the computational cost of optimizing structure and parameters for conditional likelihood is prohibitive. We present two algorithms that set parameters by maximum likelihood and choose structures by maximum CLL and AUC of PR curve respectively. For both algorithms, we present two search strategies: the ILS-Discriminative Structure Learning (ILS-DSL) algorithm which is based on the Iterated Local Search (ILS) metaheuristic [17, 18] and the Randomized Beam Search DSL (RBS-DSL) algorithm which is inspired from the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic [19, 20]. ILS-DSL explores the space of structures through a biased sampling of the set of local optima. The algorithm focuses the search not on the full space of solutions but on a smaller subspace defined by the solutions that are locally optimal for the optimization engine. RBS-DSL, following GRASP approach, performs a randomized beam search in two phases: in the construction phase it randomly constructs a beam of best candidates in terms of likelihood while in the second phase it searches for a beam of candidates with highest CLL or AUC score.

Through empirical evaluation, we discover that under some conditions satisfied by the domain and the problem, one heuristic shows better behaviour than the other. This is an important result, since it opens the way to identify for Statistical Relational Models the proper search strategy according to the properties of the domain.

2 LOGIC AND LEARNING

Because of the computational complexity, KBs are generally constructed using a restricted subset of FOL where inference and learning is more tractable. The most widely-used restriction is to Horn clauses, which are clauses containing at most one positive literal. In other words, a Horn clause is an implication with all positive antecedents, and only one (positive) literal in the consequent. A program

in the Prolog language is a set of Horn clauses. Prolog programs can be learned from examples (often relational databases) by searching for Horn clauses that hold in the data. The field of inductive logic programming (ILP) [21] deals exactly with this problem. The main task in ILP is finding an hypothesis H (a logic program, i.e. a definite clause program) from a set of positive and negative examples P and N. In particular, it is required that the hypothesis H covers all positive examples in P and none of the negative examples in N. The representation language for representing the examples together with the *covers* relation determines the ILP setting [22].

Learning from entailment is probably the most popular rule-induction setting (e.g., FOIL [23]). In this setting examples are definite clauses and an example e is covered by an hypothesis H, w.r.t the background theory B if and only if $B \cup H \models e$. Most rule-induction systems in this setting require ground facts as examples. They typically proceed following a separate-and-conquer rule-learning approach [24]. This means that in the outer loop they repeatedly search for a rule covering many positive examples and none of the negatives (set-covering approach). In the inner loop such systems generally perform a general-to-specific heuristic search using refinement operators [25, 26] based on θ-subsumption [27]. These operators perform the steps in the search-space, by making small modifications to a hypothesis. Prom a logical perspective, these refinement operators typically realize elementary generalization and specialization steps (usually under θ-subsumption). More sophisticated systems employ a search bias to reduce the search space of hypothesis.

In the rule-induction setting of learning from interpretations, examples are Herbrand interpretations and an examle e is covered by an hypthesis H, w.r.t the background theory B, if and only if e is a model of $B \cup H$. A possible world is described through sets of true ground facts which are the Herbrand interpretations. Learning from interpretations is generally easier and computationally more tractable than learning from entailment [22]. This is due to the fact that interpretations carry much more information than the examples in learning from entailment. In learning from entailment, examples consist of a single fact, while in interepretations all the facts that hold in the example are known. The approach followed by ILP systems learning from interpretations is similar to those that learn from entailment. The most important difference stands in the generality relationship. In learning from entailment an hypothesis H_1 is more general than H_2 if and only if $H_1 \models H_2$, while in learning from interpretations when $H_2 \models H_1$. A hypothesis H_1 is more general than a hypothesis H_2 if all examples covered by H_2 are also covered by H_1. ILP

systems that learn from interpretations are also well suited for learning from positive examples only [9].

3 MARKOV LOGIC NETWORKS

A MN (or Markov random field) represents a model for the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n)$ [28]. The model has an undirected graph G and a set of potential functions. There is a node for each variable and a potential function ϕ_k for each clique in the graph. A potential function is a non-negative real-valued function of the state of the corresponding clique. The joint distribution defined by a MN is given by the following:

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}) \tag{1}$$

where $x_{\{k\}}$ is the state of the kth clique (i.e., the state of the variables that appear in that clique). Z is known as the *partition function* and is given by:

$$z = \sum_{x \in x} \prod_k \phi_k(x_{\{k\}}) \tag{2}$$

MNs are often represented as log-linear models, by replacing each clique potential with an exponentiated weighted sum of features of the clique state. This replacement gives the following:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x)\right) \tag{3}$$

A feature f may be any real-valued function of the state. The focus of this paper is on binary features, $f_j \in \{0,1\}$. Thus, if we translate from the potential-function form, the model will have one feature for each possible state x_k. of each clique and its weight will be $\log(\phi(x_{\{k\}}))$. This representation is exponential in the size of the cliques, but however, we can specify a much smaller number of features in a more compact representation than the potential-function form. This is the case when large cliques are present and MLNs try to take advantage of this.

A first-order knowledge base (KB) can be considered as a set of hard constraints on a set of possible worlds: if a world violates a single formula, it will have zero probability. The idea in Markov Logic is to soften these constraints: when a world violates a formula in the KB it will be less probable,

but not impossible. The fewer formulas a world will violate, the more probable it will be. Each formula has an attached weight that represents how hard a constraint it is. A higher weight of a formula means there is a greater difference in log probability between a world that satisfies that formula and one that does not, all other things being equal.

An MLN [7] T is a set of pairs $(F_i; w_i)$, where F_i is a formula in FOL and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2,..., c_p\}$ it defines a MN $M_{T;C}$ as follows:

1. There is a binary node in $M_{T;C}$ for each possible grounding of each predicate appearing in T and the value of the node is 1 if the ground predicate is true, and 0 otherwise.
2. There is one feature in $M_{T;C}$ for each possible grounding of each formula F_i in T and the value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight w_i of the formula F_i in T becomes the weight of this feature. There is an edge between two nodes of $M_{T;C}$ iff the corresponding ground predicates appear together in at least one grounding of a formula in T.

An MLN can be viewed as a template for constructing MNs. The probability distribution over possible worlds x defined by the ground MN $M_{T;C}$ is given by:

$$P(X = x) = \frac{1}{Z}\exp\left(\sum_{i=1}^{F} w_i n_i(x)\right) \tag{4}$$

where F is the number of formulas in T and $n_i(x)$ is the number of true groundings of F_i in x. When formula weights increase, an MLN will resemble a purely logical KB and in the limit of all infinite weights it becomes equivalent to it.

The focus of this paper is on MLNs with function-free clauses assuming domain closure in order to ensure that the MNs generated will be finite. In this case, the groundings of a formula are formed by replacing the variables with constants in all possible ways.

4 LEARNING MARKOV LOGIC NETWORKS

4.1 Generative Learning of MLNs

One of the approaches for learning MN weights is iterative scaling [28]. However, in [29], maximizing the likelihood using a quasi-Newton

optimization method was found to be faster. Regarding feature induction, a possible approach is that [28], where the authors start with a set of atomic features (the original variables), conjoin each current feature with each atomic feature, add to the network the conjunction that most increases likelihood, and repeat. The work in [30] extended this to the case of conditional random fields, which can be seen as MNs trained to maximize the conditional likelihood of a set of output variables given a set of input variables.

The first approach to induce features of MLNs was proposed in [7], where the authors used the CLAUDIEN system to learn the MLNs clauses and then learned weights by maximizing pseudo-likelihood. In [11] another method was proposed that combines ideas from ILP and feature induction of MNs. This algorithm, that performs a beam or shortest first search in the space of clauses guided by a weighted pseudo-likelihood (WPLL) measure [10], outperformed that of [7]. Recently, in [12] a bottom-up approach was proposed in order to reduce the search space. This algorithm uses a *propositional* MN learning method to construct *template* networks that guide the construction of the candidate clauses. In this way, it generates much fewer candidates for evaluation. Even though it evaluates fewer candidates, after initially scoring all them, the algorithm attempts to add the candidates one by one to the current MLN, thus changing the MLN at almost each step. This greatly slows down the computation of the optimization measure WPLL. For each candidate structure, in both [11, 12] the parameters that optimize the WPLL are set through L-BFGS [31] that approximates the second-derivative of the WPLL by keeping a running finite-sized window of previous first-derivatives.

4.2 Discriminative Learning of MLNs

Learning MLNs in a discriminative fashion has produced for predictive tasks much better results than generative approaches as the results in [15] show. In this work the voted-perceptron algorithm was generalized to arbitrary MLNs by replacing the Viterbi algorithm with a weighted satisfiability solver. The new algorithm is gradient descent and uses an MPE approximation for the expected sufficient statistics which in this case are the true clause counts. In practice, these can vary widely between clauses, causing the learning problem to be highly ill-conditioned which makes gradient descent very slow. In [16] a preconditioned scaled conjugate gradient approach is shown to outperform the algorithm in [15] in terms of learning time and prediction accuracy. This algorithm is based on the scaled conjugate gradient method and good results were obtained with a simple approach: per-weight learning weights, with the

weight's learning rate being the global one divided by the corresponding clause's empirical number of true groundings.

However, for both these algorithms the structure is supposed to be given by an expert or learned previously and they focus only on the parameter learning task. This can lead to suboptimal results if the clauses given by an expert do not capture the essential dependencies in the domain in order to improve classification accuracy. On the other side, since to the best of our knowledge, no attempt has been made to learn the structure of general MLNs discriminatively, the clauses learned by generative structure learning algorithms tend to optimize the joint distribution of all the variables and applying discriminative weight learning after the structure has been learned generatively may lead to suboptimal results since the initial goal of the learned structure was not to discriminate query predicates. The only approach to learning the structure of MLNs discriminatively is that of [32] which is different from the approach we propose. In fact, their approach is restricted to only MLNs structures built with non-recursive definite clauses which is a very restricted subset of FOL (only in this case exact inference used in [32] is possible). While here we use full FOL expressiveness and propose a number of heuristics to make approximate inference tractable in the general MLNs case.

5 HYBRID RANDOMIZED METAHEURISTICS

Many widely known and high-performance local search algorithms make use of randomized choice in generating or selecting candidate solutions for a given combinatorial problem instance. These algorithms are called stochastic local search (SLS) algorithms [18] and represent one of the most successful and widely used approaches for solving hard combinatorial problem. Many "simple" SLS methods come from other search methods by just randomizing the selection of the candidates during search, such as Randomized Iterative Improvement (RII), Uniformed Random Walk, etc. Many other SLS methods combine "simple" SLS methods to exploit the abilities of each of these during search. These are known as Hybrid SLS methods [18]. ILS and GRASP are among these metaheuristics because they can be easily combined with other SLS methods.

5.1 Iterated Local Search

One of the simplest and most intuitive ideas for addressing the fundamental issue of escaping local optima is to use two types of SLS steps: one for reaching local optima as efficiently as possible, and the other for effectively escaping

Algorithm 1 The Iterated Local Search metaheuristic

Procedure Iterated Local Search
  $s_0 = GenerateInitialSolution$
  $s = LocalSearch(s_0)$
repeat
  $s' = Perturb(s, history)$
  $s' = LocalSearch(s')$
  $s = Accept(s', s, history)$
until termination condition is true
return s
end

local optima. ILS methods [18, 17] exploit this key idea, and essentially use two types of search steps alternatingly to perform a walk in the space of local optima w.r.t the given evaluation function. Algorithm 1 works as follows: the search process starts from a randomly selected element of the search space. From this initial candidate solution, a locally optimal solution is obtained by applying a subsidiary local search procedure. Then each iteration step of the algorithm consists of three major steps: first a perturbation method is applied to the current candidate solution *S*;* this yields a modified candidate solution *S'* from which in the next step a local search is performed until a local optimum *S'** is obtained. In the last third step, an acceptance criterion is used to decide from which of the two local optima *S** or *S′** the search process is continued. The algorithm can terminate after some steps have not produced improvement or simply after a certain number of steps.

In general, it is not straightforward to decide whether to use a systematic or SLS algorithm in a certain task. Systematic and SLS algorithms can be considered complementary to each other. SLS algorithms are advantageous in many situations, particularly if reasonably good solutions are required within a short time, if parallel processing is used and if knowledge about the problem domain is rather limited. In other cases, when time constraints are less important and some knowledge about the problem domain can be exploited, systematic search may be a better choice. Structure learning of MLNs is a hard optimization problem due to the large space to be explored, thus SLS methods are suitable for finding solutions of high quality in short time.

5.2 Greedy Randomized Adaptive Search Procedure

GRASP [19, 20] is an approach for quickly finding high-quality solutions by applying a greedy construction search method (that starting from an empty candidate solution at each construction step adds the solution component ranked best, according to a heuristic selection function) and subsequently a perturbative

Algorithm 2 The GRASP metaheuristic.

```
Procedure GRASP
S =
repeat
   S_0 = GreedyRandomizedConstruction(S)
   S   = LocalSearch(S_0)
   S = UpdateSolution(S,S  )
until termination criteria is met
Return S
end
```

local search algorithm to improve the candidate solution thus obtained. This type of hybrid search method often yields much better solution quality than simple SLS methods initialized at candidate solutions by Uninformed Random Picking [18]. Moreover, when starting from a greedily constructed candidate solution, the subsequent perturbative local search process typically takes much fewer improvement steps to reach a local optimum. Since greedy construction methods can typically generate a very limited number of different candidate solutions, GRASP avoids this disadvantage by randomizing the construction method such that it can generate a large number of different good starting points for a perturbative local search method. In Figure 2, in each iteration, the randomized constructive local search algorithm *GreedyRandomizedConstruction* and the perturbative *LocalSearch* algorithm are applied until the termination criterion is met. The algorithm *GreedyRandomizedConstruction,* in contrast to greedy constructive algorithms, does not necessarily add the best solution component but rather selects it randomly from a list of highly ranked solution components (*Restricted Candidate List*) which can be defined by cardinality restriction or by value restriction. In this paper we present a novel algorithm inspired from GRASP that performs randomized beam search by scoring the structures through maximum likelihood in the first phase and then uses maximum CLL or AUC for PR curve in a second step to randomly generate a beam of the best clauses to add to the current MLN structure.

6 DISCRIMINATIVE STRUCTURE LEARNING OF MLNS

In this section we describe our approach of tailoring the ILS metaheuristic to the problem of learning the structure of MLNs. We also present a novel algorithm inspired from GRASP together with some simple heuristics for making tractable randomized beam searching. We describe how weights are set and how structures are scored. The approach we follow is similar to [33] where Bayesian Networks were learned by setting weights through maximum likelihood and choosing structures by maximizing conditional likelihood. In

Algorithm 3 The structure learning algorithm

```
Input: (P:set of predicates, MLN:Markov Logic Network, RDB:Relational Database, QP: Query predicate)
CLS = All clauses in MLN ∪ P;
LearnWeights(MLN,RDB);
BestScore = ComputeScore(MLN,RDB,QP);
repeat
   BestClause = SearchBestClause(P,MLN,BestScore,CLS,RDB,QP);
   if BestClause ≠ null then
      Add BestClause to MLN;
      BestScore = ComputeScore(MLN,RDB,QP);
   end if
until BestClause = null for    consecutive steps
Return MLN
For DSL_CLL ComputeScore computes the average CLL over all the groundings of the query predicate QP
For DSL_AUC ComputeScore computes the AUC of PR curve
If Iterated Local Search: SearchBestClause = ILS-SearchBestClause (Algorithm 4)
If Randomized Beam Search: SearchBestClause = RBS-SearchBestClause (Algorithm 6)
```

addition, we propose a number of heuristics to make the computation of CLL tractable for the very large number of potential candidates.

6.1 Space Exploration Strategy

Algorithm 3 iteratively adds the best clause to the current MLN until S consecutive steps have not produced improvement (however other stopping criteria could be applied). It can start from an empty network or from an existing KB. Like in [7, 11] we add all unit clauses (single predicates) to the MLN. The initial weights are learned in *LearnWeights* through L-BFGS and the initial structure is scored in *ComputeScore* through MC-SAT. MC-SAT takes in input a MLN, a query predicate and evidence ground facts and computes for each grounding of the query predicate, its probability of being true. From these values in *ComputeScore,* the CLL is computed as the average of CLL over all these groundings. *DSLQLL* scores directly the candidate clauses by CLL, while *DSLAUC* computes the AUC of the PR curve by using the package of [34].

The search for the best clause is performed in the *SearchBestClause* procedure (Algorithm 4). It performs an iterated local search to find the best clause to add to the current MLN. It starts by randomly choosing a unit clause CL_C in the search space. Then it performs a greedy local search to efficiently reach a local optimum CL_S. At this point, a perturbation method is applied leading to the neighbor CL'_C of *CLs* and then a greedy local search is applied to CL'_C to reach another local optimum CL'_S. The *accept* function decides whether the search must continue from the previous local optimum CL_C or from the last found local optimum CL'_S (*accept* can perform random walk or iterative improvement in the space of local optima).

Algorithm 4 The SearchBestClause component of ILS

Input:(P: set of predicates, MLN: Markov Logic Network, BestScore: CLL of AUC score, BestWPLL: WPLL score, CLS: List of clauses, RDB: Relational Database, QP: Query predicate)
CL_C = Random Pick a clause in CLS ∪ P;
CL_S = $LocalSearch_{II}$(CL_S,BestScore,BestWPLL);
BestClause = CL_S;
repeat
  CL'_C = $Perturb$(CL_S);
  CL'_S = $LocalSearch_{II}$(CL'_C,MLN,BestScore,BestWPLL);
  if ComputeScore(BestClause,MLN,RDB,QP) ComputeScore(CL'_S,MLN,RDB,QP) **then**
    BestClause = CL'_S;
    Add BestClause to MLN;
    BestScore = ComputeScore(CL'_S,MLN,RDB,QP)
  end if
  CL_S = accept(CL_S,CL'_S);
until k consecutive steps have not produced improvement
Return BestClause
For ILS_{CLL} ComputeScore computes the average CLL over all the groundings of the query predicate
For ILS_{AUC} ComputeScore computes the AUC of PR curve

Careful choice of the various components of *SearchBestClause* is important to achieve high performance. The clause perturbation operator (flipping the sign of literals, removing literals or adding literals) has the goal to jump in a different region of the search space where search should start with the next iteration. There can be strong or weak perturbations which means that if the jump in the search space is near to the current local optimum the subsidiary local search procedure *LocalSearch*$_{II}$ (Algorithm 5) may fall again in the same local optimum and enter regions with the same value of the objective function called *plateau,*

Algorithm 5 The subsidiary procedure LocalSearch and the Step function of ILS

LocalSearch$_{II}$(CL_S,BestScore,BestWPLL)
wp: walk probability, the probability of performing an improvement step or a random step
repeat
  NBHD = Neighborhood of CL_C constructed using the clause construction operators;
  CL_S = $Step_{RII}$(CL_C,NBHD,wp,BestScore,BestWPLL);
  CL_C = CL_S;
until two consecutive steps do not produce improvement
Return CL_S;
Step$_{RII}$(CL_C,NBHD,wp,BestScore,BestWPLL)
U = random(]0,1]); random number using a Uniform Probability Distribution
if U wp) **then**
  CL_S = step$_{URW}$(CL_C,NBHD)
  Uninformed Random Walk: randomly choose a neighbor from NBHD
else
  CL_S = step$_{II}$(CL_C,NBHD)
  Iterative Improvement: among the improving neighbors in NBHD that improve BestWPLL, choose the one that maximally improves BestScore in terms of CLL or AUC.
  If there is no improving neighbor choose the minimally worsening one
end if
Return CL_S

Algorithm 6 The SearchBestClause procedure of the RBS-DSL algorithm

```
SearchBestClause(P: set of predicates, MLN: Markov Logic Network, BestScore: CLL or AUC
score, BestWPLL: WPLL score,CLS: List of clauses, RDB: Relational Database, QP: Query
predicate)
Beam = CLS;
repeat
   Candidates = GenerateCandidates(Beam,P);
   for Each Clause C in Candidates do
      Add C to the current MLN; LearnWeights(MLN,RDB);
      CWPLL = Score of C by WPLL; WPLLGain of C = CWPLL - BestWPLL;
   end for
   BestWPLLClauses = RandomizedConstruction(Candidates,BestWPLL);
   scoredList: list of candidates scored in terms of CLL (or AUC);
   for Each Clause C in BestWPLLClauses do
      Add C to the current MLN;
      ComputeScore(MLN,RDB);
      Add C to scoredList;
   end for
   NewBeam = RandomizedBeam(scoredList,BestScore);
   BestClause = Best Clause in NewBeam;
   Beam = NewBeam;
until two consecutive iterations have not produced improvement
Return BestClause
For RBS_CLL ComputeScore computes the average CLL over all the groundings of the query
predicate
For RBS_AUC ComputeScore computes the AUC of PR curve
```

but if the jump is too far, $LocalSearch_{II}$ may take too many steps to reach another good solution. In our algorithm we use only strong perturbations, i.e., we always re-start from unit clauses. Regarding the procedure $LocalSearch_{II}$, we decided to use an iterative improvement approach (the walk probability is set to zero and the best clause is always chosen in $Step_{II}$) in order to balance intensification (greedily increase quality) and diversification (randomness induced). Finally, the *accept* function always accepts the best solution found so far.

For Randomized Beam Search, Algorithm 6 starts with a beam of the initial clauses (in case there are other clauses previously learned) and unit clauses. Prom this beam, by using the clause construction operators, the algorithm constructs in *GenerateCandidates* all the potential candidate clauses to be scored for adding to the current structure. Then for each of these candidates the gain in WPLL is computed. In the next step, the algorithm performs a randomized construction of candidate clauses in the *RandomizedConstruction* procedure (Algorithm 7). In this procedure, similar to a GRASP approach, it is first defined the Restricted Candidate List (RCL) in a random fashion by cardinality value of WPLL. All candidates with a gain in WPLL greater than $minGain + \alpha * (maxGain - minGain)$ (where α is a random number from a uniform probability distribution), are considered to be included in the RCL. To induce randomness in the algorithm the parameter α has an important function and is called the RCL parameter. It determines the level of randomness or

Algorithm 7 Randomized Construction of the best WPLL candidate list

```
RandomizedConstruction(CandidateClauses,BestWPLL)
BestWPLLClauses: Randomized List of best WPLL candidates;
maxNumClauses = maximum number of clauses to choose from RCL;
  = random([0,1]); random number using a Uniform Probability Distribution
threshold: value to use as limit;
minGain = minimumWPLLGain(CandidateClauses);
maxGain = maximumWPLLGain(CandidateClauses);
if minGain > 0 then
   threshold = minGain +    * (maxGain - minGain) ;
else
   threshold = 0;
end if
for Each Clause C in CandidateClauses do
   if WPLLGain(C) > threshold then
      rand = random([0,1]); random number using a Uniform Probability Distribution
      if rand >    then
         Add C to BestWPLLClauses;
      end if
      if size of BestWPLLClauses = maxNumClauses then
         break;
      end if
   end if
end for
Return BestWPLLClauses
```

greediness in the construction. In some GRASP implementations the parameter is fixed while in others it is adapted dynamically. The case $\alpha = 0$ corresponds to a pure greedy algorithm, while $\alpha = 1$ is equivalent to a random construction.

The similarity of our algorithm with GRASP is that randomization is applied not only to the choice of the candidates from the RCL but also to the construction of the RCL. On the other side, the difference with GRASP is that in GRASP only one candidate is randomly chosen from the RCL in order to continue the search, while in our algorithm a list of clauses is randomly constructed by choosing them from the RCL. Another difference is that we introduce the following heuristic: only candidates with a positive gain in WPLL are to be considered for scoring of CLL. Thus, in case there are candidates with no gain ($minGain \leq 0$), we set the value *threshold* to zero. In order not to loose randomness in case of *threshold* = 0, a random choice among the RCL candidates follows. Once the potential candidates for the RCL are randomly constructed, the algorithm randomly chooses among these according to the random number *rand* and the paramater λ. In our experiments we found empirically that the value $\lambda = 0.5 * beamSize/100$ induces enough randomness in the choice from RCL candidates. This value depends on the size of the beam which is a parameter of the main algorithm. In most cases, the number of candidates in the RCL and those that are chosen from this list can be very high. This can cause intractable computation times because most of these candidates

have to be scored again in terms of CLL (or AUC). For this reason, it is reasonable to pose a limit in the size of the clauses to be evaluated in the next step. This is achieved by setting the parameter *maxNumClauses* which determines the number of potential candidates to be scored by CLL (or AUC).

After the procedure *RandomizedConstruction* returns the list *BestWPLL-Clauses,* all the candidates in this list are scored for CLL (or AUC) and given in input to the *RandomizedB earn* procedure. In this procedure it is performed the same randomized process on the candidates but this time based on their CLL (or AUC) values. Differently from the *RandomizedConstruction* procedure, the randomized construction of the beam does not exclude negative values for the gain of the candidates. The value of the parameter λ is the same as for the *RandomizedConstruction* procedure.

7 EXPERIMENTS

Through experimental evaluation we want to compare the behaviour of the two randomized search strategies.

7.1 Datasets

All the experiments were carried out on two publicly available benchmark databases: the UW-CSE database used by [11, 7, 12] and the CORA dataset used by [11, 35, 16]. Both databases represent standard relational domains that have been used for two important relational problems: CORA on Entity Resolution and UW-CSE on Social Network analysis. For CORA we used the cleaned version from [35]. The published version of the UW-CSE database has 15 predicates of 10 types. Types include: publication, person, course, quarter etc, while predicates include: Student (person), Professor(person), AdvisedBy(personl, per-son2), TaughtBy(course, person, quarter), Publication (paper, person) etc. The dataset has in total 2673 tuples which represent the true ground atoms, assuming the remainder as false (under closed-word assumption). The CORA dataset has 1295 citations of 132 different computer science papers, drawn from the CORA Computer Science Research Paper Engine. The original task was to predict which citations refer to the same paper, given words in their author, title, and venue fields. Moreover, the labeled data specify which pairs of author, title, and venue fields refer to the same entities, thus we performed experiments for each field in order to evaluate the ability of the model to deduplicate fields as well as citations. The number of equivalences for CORA may become very large, and to render this problem tractable, we used the canopies found in [35].

7.2 Systems and Methodology

We implemented all the algorithms in the Alchemy package [36]. We used the implementation of L-BFGS and Lazy-MC-SAT in Alchemy to learn maximum WPLL weights and compute CLL during clause search.

For both datasets, for all our algorithms we used the following parameters: the mean and variance of the Gaussian prior were set to 0 and 100, respectively; maximum variables per clause = 4; maximum predicates per clause = 4; penalization of weighted pseudo-likelihood = 0.01 for UW-CSE and 0.001 for CORA. beamSize = 5 for UW-CSE and 10 for CORA. For L-BFGS we used the following parameters: maximum iterations = 10,000 (tight) and 10 (loose); convergence threshold = 10^{-5} (tight) and 10^{-4} (loose). For Lazy-MC-SAT during learning we used the following parameters: for iterated local search algorithms memory limit = 300MB for both datasets, for randomized beam search algorithms, memory limit = 600MB for UW-CSE and 1GB for CORA, maximum number of steps for Gibbs sampling = 100, simulated annealing temperature = 0.5. For ILS based algorithms, the parameter k (number of iterations without improvement) was set to three, while the parameter S was set to 2 for ILS and 1 for RBS based algorithms. All these parameters were set in an *ad hoc* manner and per-fold optimization may lead to better results. In particular the memory requirements of Lazy-MC-SAT were set higher for RBS based algorithms because we empirically observed that a larger number of

Algorithm 8 Randomized choice of the best CLL (or AUC) candidate list to form the new beam.

```
RandomizedBeam(ListClauses,BestScore)
ListClauses: list of clauses scored for CLL (or AUC)
newBeam: new list of clauses to randomly generate from ListClauses;
beamSize = Size of beam for the algorithm RBS-DSL;
  = random([0,1]); random number using a Uniform Probability Distribution
threshold: value to use as limit;
minGain = minimumGain(ListClauses);
maxGain = maximumGain(ListClauses);
threshold = minGain +    * (maxGain - minGain) ;
for Each Clause C in ListClauses do
   if Gain(C) > threshold then
      rand = random([0,1]); random number using a Uniform Probability Distribution
      if rand >    then
         Add C to newBeam;
      end if
      if size of newBeam = beamSize then
         break;
      end if
   end if
end for
Return newBeam;
For RBS_CLL minimumGain returns the minimum gain in CLL among all candidates
For RBS_AUC minimumGain returns the minimum gain in AUC among all candidates
```

potential clauses compared to ILS, required larger memory requirements. This is due to the nature of RBS which evaluates more clauses than ILS during search. However, as can be noted from the results of the experiments a larger memory spent by Lazy-MC-SAT to score the structures in the RBS based algorithms did not produce much higher results than the ILS based versions.

For both datasets we performed cross-validation (in the UW-CSE each area is considered as a fold). For each system on each test set, we measured the CLL and the area under the AUC for all the predicates. With CLL, we directly measure the quality of the probability estimates produced, while on the other side, the advantage of AUC is that it is insensitive to the large number of true negatives. The CLL of a query predicate is computed as the average over all its groundings of the ground atom's log-probability The precision-recall curve for a predicate is computed by varying the CLL threshold above which a ground atom is predicted to be true; i.e. the predicates whose probability of being true is greater than the threshold are positive and the rest are negative.

7.3 Results

After learning the structure discriminatively, we performed inference on the test fold for both datasets by using MC-SAT with number of steps = 10000 and simulated annealing temperature = 0.5. For each experiment, on the test fold all the groundings of the query predicates were commented: advisedBy for the UW-CSE dataset (professor and student are also commented) and sameBib, sameTitle, sameAuthor and same Venue for CORA. MC-SAT produces probability outputs for every grounding of the query predicate on the test fold. We used these values to compute the average CLL over all the groundings and to compute the PR curve.

We denote the two versions of each algorithm as ILS_{CLL}, ILS_{AUC} and RBS_{CLL}, RBS_{AUC}. For the algorithms that optimize AUC of PR curve during search, we scored each structure by using the package of [34]. The results for all algorithms on the UW-CSE dataset are reported in Table 1 for CLL and Table 2 for AUC. In Table 1, CLL is averaged over all the groundings of the predicate advisedBy in the test fold.

As the results of Table 1 show, the ILS_{CLL} algorithm performs better than RBS_{CLL} in terms of CLL. The better solution found by ILS_{CLL}, leads to the conclusion that in this domain for this optimizing measure, this algorithm performs a better exploration of the search space compared to RBS_{CLL}. The other two versions, ILS_{AUC} and RBS_{AUC} achieve the same result, which means that for this measure both these algorithms are equivalent to use.

Table 1: CLL Results for the query predicate advisedBy in the UW-CSE domain

Alg.	language	graphics	systems
ILS_{CLL}	−0.048±0.016	−0.016±0.003	−0.020±0.003
RBS_{CLL}	−0.043±0.015	−0.026±0.004	−0.058±0.002
ILS_{AUC}	−0.028±0.008	−0.015±0.003	−0.017±0.002
RBS_{AUC}	−0.025±0.007	−0.015±0.003	−0.017±0.003
Alg.	**theory**	**ai**	**Overall**
ILS_{CLL}	−0.020±0.005	−0.022±0.003	−0.025±0.006
RBS_{CLL}	−0.019±0.004	−0.032±0.005	−0.036±0.006
ILS_{AUC}	−0.018±0.004	−0.019±0.003	−0.019±0.004
RBS_{AUC}	−0.018±0.004	−0.020±0.003	−0.019±0.004

However, from Table 2, we can see that RBS_{CLL}, produces much better results than ILS_{CLL} in terms of AUC. And overall, the RBS-based algorithms perform better than the ILS-based approaches. The inverse situation is verified in Table 3, where ILS-based approaches provide much better solutions in terms of CLL. The same happens also in Table 4 where the ILS-based algorithms achieve higher results in terms of AUC.

In general, it is not straightforward to provide sound analysis of the behaviour of randomized search approaches. It becomes even more difficult to arrive at conclusions when functions to be optimized are complex such as those in Statistical Relational Models. However, we try here to characterize the searching problem based on the properties of the dataset. The UW-CSE dataset with a total

Table 2: AUC results for the query predicate advisedBy in the UW-CSE domain

Alg.	language	graphics	systems	theory	ai	Overall
ILS_{CLL}	1.011	0.006	0.007	0.010	0.006	0.008
RBS_{CLL}	0.034	0.009	0.010	0.012	0.008	0.015
ILS_{AUC}	0.016	0.005	0.007	0.005	0.008	0.008
RBS_{AUC}	0.073	0.005	0.005	0.005	0.007	0.019

Table 3: CLL results for all query predicates in the CORA domain

Alg.	sameBib	sameTitle	sameAuthor	sameVenue	Overall
ILSCLL	−0.087±0.001	−0.077±0.006	−0.148±0.009	−0.121±0.004	−0.108±0.005
RBSCLL	−0.222±0.003	−0.120±0.008	−0.126±0.008	−0.129±0.005	−0.149±0.006
ILSAUC	−0.168±0.002	−0.117±0.010	−0.158±0.011	−0.101±0.004	−0.136±0.007
RBSAUC	−0.254±0.002	−0.077±0.007	−0.133±0.011	−0.172±0.005	−0.159±0.006

of 2673 tuples can be considered of much smaller size compared to CORA that has 70367 tuples. The results of Table 2 show that on the UW-CSE dataset, the RBS-based algorithms perform better in terms of AUC, and Table 1 shows they are competitive in terms of CLL since they underperform the other ILS-based algorithms only because of the low results in the *systems* fold of the dataset.

On the other side, on the larger dataset CORA, the ILS-based approaches perform better, both in terms of CLL and AUC. The superiority of ILS towards RBS is evident and leads us to the conclusion that on small datasets, ILS and RBS approaches are competitive and RBS can also lead to better results (for AUC), while for larger datasets ILS approaches produce much better results.

The performance comparison between different search strategies and our findings regarding the dimension of the domain, provide additional development in the field of Statistical Relational Learning where exploration of combinatorial spaces is necessary. Further analysis is needed to understand how ILS can be further improved in small domains where RBS achieves sometimes better performance.

Moreover, due to the general language that MLNs represent, it is possible to express a very large class of problems which need the expressiveness power of

Table 4: AUC results for all query predicates in the CORA domain

Alg.	sameBib	sameTitle	sameAuthor	sameVenue	Overall
ILS_{CLL}	0.603	0.428	0.371	0.315	0.429
RBS_{CLL}	0.265	0.546	0.600	0.233	0.411
ILS_{AUC}	0.334	0.470	0.688	0.252	0.436
RBS_{AUC}	0.322	0.423	0.534	0.175	0.364

logic. Combinatorial spaces characterize a large variety of problems which could be expressed in MLNs and the algorithms proposed here could be exploited without restrictions on the applicability due to the power of logic to represent most of the existing real-world problems. However, it is not always necessary to use MLNs to express problems since in many cases relations are not needed (in this case MNs would be sufficient) or there is no uncertainty in the domain (in this case the pure logic approach is sufficient), and MLNs can deal with both these as special cases.

Overall, regarding scalability, since the problems we treat here represent benchmark tasks in learning in relational uncertain domains, we can confirm that the proposed algorithms are powerful exploring methods for large spaces. In particular, the CORA domain is very large and a deterministic approach would require much more computational effort. The results on this dataset show that very large datasets can be processed in reasonable time with randomized strategies by preserving solutions quality.

8 RELATED WORK

Recently many works related to hybrid languages have addressed the problem of discriminative learning. Our discriminative method falls among those approaches that tightly integrate rule-induction and statistical learning in a single step for structure learning. The earlier works in this direction are those in [37, 38] that employ statistical models such as maximum entropy modeling in [37] and logistic regression in [38]. These approaches can be computationally very expensive. A simpler approach that integrates FOIL and Naive Bayes is nFOIL proposed in [39]. This approach interleaves the steps of generating rules and scoring them through CLL. In another work [40] these steps are coupled by scoring the clauses through the improvement in classification accuracy. This algorithm incrementally builds a Bayes net during rule learning and each candidate rule is introduced in the network and scored by whether it improves the performance of the classifier. In a recent approach [41], the kFOIL system integrates rule-induction and support vector learning. kFOIL constructs the feature space by leveraging FOIL search for a set of relevant clauses. The search is driven by the performance obtained by a support vector machine based on the resulting kernel. The authors showed that kFOIL improves over nFOIL. Recently, in TFOIL [42], Tree Augmented Naive Bayes, a generalization of Naive Bayes was integrated with FOIL and it was shown that TFOIL outperforms nFOIL.

Regarding the two steps integration, the most closely related approach to the proposed algorithms is nFOIL (and TFOIL as an extension) which is the first

system in literature to tightly integrate feature induction and Naive Bayes. Such a dynamic propositionalization was shown to be superior compared to static propositionalization approaches that use Naive Bayes only to post-process the rule set. The approach is different from ours in that nFOIL selects features and parameters that jointly optimize a probabilistic score on the training set, while our algorithms maximize the likelihood on the training data but select the clauses based on the tuning set. This approach is similar to SAYU [40] that uses the tuning set to compute the score in terms of classification accuracy or AUC, with the difference that DSL_{CLL} uses CLL as score instead of AUC. SAYU is similar only to DSL_{AUC}. From the point of view of steps integration, MACCENT [37] follows a similar approach by inducing clausal constraints (one at a time) that are used as features for maximum-entropy classification.

Finally, from the point of view of search strategies, our algorithms are also similar to approches in ILP that exploit SLS [43]. The algorithms that we propose here are different in that they use likelihood as evaluation measure instead of ILP coverage criteria. Moreover, our algorithms differ from those in [43] in that we use Hybrid SLS approaches which can combine other simple SLS methods to produce high performance algorithms.

9 CONCLUSIONS AND FUTURE WORK

Markov Logic Networks are a powerful representation that combines first-order logic and probability by attaching weights to first-order formulas and using these as templates for features of Markov networks. In this paper we have introduced two algorithms that learn discriminatively first-order clauses and their weights. The algorithms score the candidate structures by maximizing conditional likelihood or area under the Precision-Recall curve while setting the parameters by maximum pseudo-likelihood. For each algorithm, we have proposed two versions based respectively on the Iterated Local Search and Greedy Randomized Adaptive Search Procedure metaheuristics. Empirical evaluation with real-world data in two domains show that on larger datasets, the ILS-based approaches perform better, both in terms of CLL and AUC, while on small datasets, ILS and RBS approaches are competitive and RBS can also lead to better results for AUC.

Directions for future work include adapting dynamically the nature of perturbation in the ILS procedure; adapting dynamically the randomization parameter of GRASP; finding heuristics to select, among clauses that do not improve WPLL, potential candidates that can improve CLL. Finally, we would like to apply our algorithms to other complex structural uncertain domains.

REFERENCES

[1] Halpern, J.: An analysis of first-order logics of probability. Artificial Intelligence 46 (1990) 311-350
[2] Nilsson, N.: Probabilistic logic. Artificial Intelligence 28 (1986) 71-87
[3] Wellman, M. Breese, J.S., Goldman, R.P.: From knowledge bases to decision models. Knowledge Engineering Review 7 (1992)
[4] Getoor, L., Friedman, N., Koller, D., Taskar, B.: Learning probabilistic models of link structure. Journal of Machine Learning Research 3 (2002) 679-707
[5] Natarajan, S., Tadepalli, P., Altendorf, E., Dietterich, T.G., Fern, A., Res-tificar, A.C.: Learning first-order probabilistic models with combining rules. In: ICML. (2005) 609-616
[6] Neville, J., Jensen, D.: Dependency networks for relational data. In: Proc. 4th IEEE Int'l Conf. on Data Mining, IEEE Computer Society Press. (2004) 170-177
[7] Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62 (2006)107-236
[8] Singla, P., Domingos, P.: Markov logic in infinite domains. In: Proc. 23rd UAI, AUAI Press (2007) 368-375
[9] De Raedt, L., Dehaspe, L.: Clausal discovery. Machine Learning 26 (1997) 99-146
[10] Besag, J.: Statistical analysis of non-lattice data. Statistician 24 (1975) 179-195
[11] Kok, S., Domingos, P.: Learning the structure of markov logic networks. In: Proc. 22nd Int'l Conf. on Machine Learning. (2005) 441-448
[12] Mihalkova, L., Mooney, R.J.: Bottom-up learning of markov logic network structure. In: Proc. 24th Int'l Conf. on Machine Learning. (2007) 625-632
[13] Biba, M., Ferilli, S., Esposito, F.: Structure learning of markov logic networks through iterated local search. In: Proceedings of 18th European Conf. on AI (ECAI-08). (2008) 361-365
[14] Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proc. 18th ICML. (2001) 282-289
[15] Singla, P., Domingos, P.: Discriminative training of markov logic networks. In: Proc. 20th Nat'l Conf. on AI, (AAAI), AAAI Press (2005) 868-873
[16] Lowd, D., Domingos, P.: Efficient weight learning for markov logic networks. In: Proc. 11th PKDD, Springer (2007) 200-211
[17] Loureno, H., Martin, O., Stutzle, T.: Iterated local search. In: Handbook of Metaheuristics. F. Glover and G. Kochenberger, Kluwer, Norwell, MA, USA (2002) 321-353
[18] Hoos, H.H., Stutzle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, San Francisco (2005)
[19] Feo, T.A., Resende, M.: A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8(2) (1989) 67-71

[20] Feo, T.A., Resende, M.: Greedy randomized adaptive search procedures. Journal of Global Optimization 6 (1995) 109-133
[21] Lavrac, N., Dzeroski, S.: Inductive Logic Programming: Techniques and applications. UK: Ellis Horwood, Chichester (1994)
[22] De Raedt, L.: Logical settings for concept-learning. Artificial Intelligence 95(1) (1997) 197-201
[23] Quinlan, J.R.: Learning logical definitions from relations. Machine Learning 5 (1990) 239-266
[24] Furnkranz, J.: Separate-and-conquer rule learning. Artificial Intelligence Review 13(1) (1999) 3-54
[25] Nienhuys-Cheng, S.H., de Wolf, R.: Foundations of Inductive Logic Programming. Springer-Verlag (1997)
[26] Shapiro., E.: Algorithmic Program Debugging. MIT Press (1983)
[27] Plotkin., G.D.: A note on inductive generalization. In Machine Intelligence, Edinburgh University Press 5 (1970) 153-163
[28] Delia Pietra, S., Pietra, V.D., Laferty, J.: Inducing features of random fields. IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (1997) 380-392
[29] Sha, F., Pereira, F.: Shallow parsing with conditional random fields. In: Proc. HLT-NAACL-03. (2003) 134-141
[30] McCallum, A.: Efficiently inducing features of conditional random fields. In: Proc. UAI-03. (2003) 403-410
[31] Liu, D.C., Nocedal, J.: On the limited memory bfgs method for large scale optimization. Mathematical Programming 45 (1989) 503-528
[32] Huynh, T.N., Mooney, R.J.: Discriminative structure and parameter learning for markov logic networks. In: In Proc. of the 25th International Conference on Machine Learning (ICML). (2008)
[33] Grossman, D., Domingos, P.: Learning bayesian network classifiers by maximizing conditional likelihood. In: Proc. 21st ICML, ACM Press (2004) 361-368
[34] Davis, J., Goadrich, M.: The relationship between precision-recall and roc curves. In: Proc. 23rd ICML. (2006) 233-240
[35] Singla, P., Domingos, P.: Entity resolution with markov logic. In: Proc. ICDM-06. (2006) 572-582
[36] Kok, S., Singla, P., Richardson, M., Domingos, P.: The alchemy system for statistical relational ai. Technical report, Department of CSE-UW, Seattle, WA, http://alchemy.cs.washington.edu/ (2005)
[37] Dehaspe, L.: Maximum entropy modeling with clausal constraints. In: Proc. of ILP-97, Springer (1997) 109-124

[38] Popescul, A., Ungar, L.H.: Structural logistic regression for link analysis. In: Proc. of MRDM-03, Washington, DC: ACM Press (2003) 92-106
[39] Landwehr, N., Kersting, K., De Raedt, L.: nfoil: Integrating naive bayes and foil. In: Proc. 20th Nat'l Conf. on Artificial Intelligence, AAAI Press (2005) 795-800
[40] Davis, J., E. Burnside, I.d.C.D., Page, D., Costa, V.S.: An integrated approach to learning bayesian networks of rules. In: Proc. ECML-2005. Volume 3720 LNCS. (2005) 84-95
[41] Landwehr, N., Passerini, A., L., D.R., Prasconi, P.: kfoil: Learning simple relational kernels. In: Proc. 21st Nat'l Conf. on Artificial Intelligence, AAAI Press (2006)
[42] Landwehr, N., Kersting, K., De Raedt, L.: Integrating naive bayes and foil. J. of Machine Learning Research (2007) 481-507
[43] Zelezny, F., Srinivasan., A., Page., D.: Randomised restarted search in ilp. Machine Learning 64(1-3) (2006) 183-208