Structure Learning of Markov Logic Networks through Iterated Local Search

Marenglen Biba and **Stefano Ferilli** and **Floriana Esposito**[1]

Abstract.
Many real-world applications of AI require both probability and first-order logic to deal with uncertainty and structural complexity. Logical AI has focused mainly on handling complexity, and statistical AI on handling uncertainty. Markov Logic Networks (MLNs) are a powerful representation that combine Markov Networks (MNs) and first-order logic by attaching weights to first-order formulas and viewing these as templates for features of MNs. State-of-the-art structure learning algorithms of MLNs maximize the likelihood of a relational database by performing a greedy search in the space of candidates. This can lead to suboptimal results because of the incapability of these approaches to escape local optima. Moreover, due to the combinatorially explosive space of potential candidates these methods are computationally prohibitive. We propose a novel algorithm for learning MLNs structure, based on the Iterated Local Search (ILS) metaheuristic that 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. We show through experiments in two real-world domains that the proposed approach improves accuracy and learning time over the existing state-of-the-art algorithms.

1 Introduction

Traditionally, AI research has fallen into two separate subfields: one that has focused on logical representations, and one on statistical ones. Logical AI approaches like logic programming, description logics, classical planning, symbolic parsing, rule induction, etc, tend to emphasize handling complexity. Statitistical AI approaches like Bayesian networks, hidden Markov models, Markov decision processes, statistical parsing, neural networks, etc, tend to emphasize handling uncertainty. However, intelligent agents must be able to handle both for real-world applications. The first attempts to integrate logic and probability in AI date back to the works in [1, 8, 19]. Later, several authors began using logic programs to compactly specify Bayesian networks, an approach known as knowledge-based model construction [26].

Recently, in the burgeoning field of statistical relational learning [7], several approaches for combining logic and probability have been proposed such as probabilistic relational models [17], bayesian logic programs [10], relational dependency networks [18], and others. All these approaches combine probabilistic graphical models with subsets of first-order logic (e.g., Horn Clauses). In this paper we focus on Markov logic [22], a powerful representation that has finite first-order logic and probabilistic graphical models as special cases. It extends first-order logic by attaching weights to formulas providing the full expressiveness of graphical models and first-order logic in finite domains and remaining well defined in many infinite domains [22, 25]. Weighted formulas are viewed as templates for constructing MNs and in the infinite-weight limit, Markov logic reduces to standard first-order logic. In Markov logic it is avoided the assumption of i.i.d. (independent and identically distributed) data made by most statistical learners by using the power of first-order logic to compactly represent dependencies among objects and relations.

Learning an MLN consists in structure learning (learning the logical clauses) and weight learning (setting the weight of each clause). In [22] structure learning was performed through ILP methods [13] followed by a weight learning phase in which maximum-pseudolikelihood [2] weights were learned for each learned clause. State-of-the-art algorithms for structure learning are those in [11, 16] where learning of MLNs is performed in a single step using weighted pseudo-likelihood as the evaluation measure during structure search. However, these algorithms follow systematic search strategies that can lead to local optima and prohibitive learning times. The algorithm in [11] performs a beam search in a greedy fashion which makes it very susceptible to local optima, while the algorithm in [16] works in a bottom-up fashion trying to consider fewer candidates for evaluation. Even though it considers fewer candidates, after initially scoring all candidates, this algorithm attempts to add them one by one to the MLN, thus changing the MLN at almost each step, which greatly slows down the computation of the optimal weights. Moreover, both these algorithms cannot benefit from parallel architectures. We propose an approach based on the Iterated Local Search (ILS) metaheuristics that samples the set of local optima and performs a search in the sampled space. We show that, through a simple parallelism model such as independent multiple walk, ILS achieves improvements towards the state-of-the-art algorithms.

The paper is organized as follows: Section 2 introduces MNs and MLNs, Section 3 describes learning approaches for MLNs, Section 4 introduces stochastic local search methods, Section 5 presents the ILS metaheuristic for MLNs structure learning. We present the experiments in Section 6 and conclude in Section 7.

2 Markov Networks and Markov Logic Networks

A MN (also known as Markov random field) is a model for the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n) \in \chi$ [5]. It is composed of an undirected graph G and a set of potential functions. The graph has a node for each variable, and the model has 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

[1] Department of Computer Science, University of Bari, Italy, email: {biba,ferilli,esposito}@di.uniba.it

clique. The joint distribution represented by a MN is given by:

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

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

$$Z = \sum_{x \in \mathcal{X}} \prod_{k} \phi_k(x_{\{k\}})$$

MNs are often conveniently represented as log-linear models, with each clique potential replaced by an exponentiated weighted sum of features of the state, leading to:

$$P(X = x) = \frac{1}{Z} \exp(\sum_{j} w_j f_j(x))$$

A feature may be any real-valued function of the state. We will focus on binary features, $f_j \in \{0, 1\}$. In the most direct translation from the potential-function form, there is one feature corresponding to each possible state x_k of each clique, with its weight being $\log(\phi(x_{\{k\}})$. This representation is exponential in the size of the cliques. However a much smaller number of features (logical functions of the state of the clique) can be specified, allowing for a more compact representation than the potential-function form, particularly when large cliques are present. MLNs take advantage of this.

A first-order knowledge base (KB) can be seen as a set of hard constraints on the set of possible worlds: if a world violates even one formula, it has zero probability. The basic idea in Markov logic is to soften these constraints: when a world violates one formula in the KB it is less probable, but not impossible. The fewer formulas a world violates, the more probable it is. Each formula has an associated weight that reflects how strong a constraint it 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.

A MLN [22] *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* = $\{c_1, c_2, \ldots, c_p\}$ it defines a MN $M_{L;C}$ as follows:

1. $M_{L;C}$ contains one binary node for each possible grounding of each predicate appearing in *L*. The value of the node is 1 if the ground predicate 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*. Thus there is an edge between two nodes of $M_{L;C}$ iff the corresponding ground predicates appear together in at least one grounding of one formula in *L*. An MLN can be viewed as a template for constructing MNs. The probability distribution over possible worlds *x* specified by the ground MN $M_{L;C}$ is given by

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

where *F* is the number of formulas in the MLN and $n_i(x)$ is the number of true groundings of F_i in *x*. As formula weights increase, an MLN increasingly resembles a purely logical KB, becoming equivalent to one in the limit of all infinite weights.

In this paper we focus on MLNs whose formulas are function-free clauses and assume domain closure (it has been proven that no expressiveness is lost), ensuring that the MNs generated are finite. In this case, the groundings of a formula are formed simply by replacing its variables with constants in all possible ways.

3 Structure and Parameter Learning of MLNs

A first-order knowledge base (KB) is a set of sentences or formulas in first-order logic [6]. Formulas are constructed using four types of symbols: constants, variables, functions, and predicates. Constant symbols represent objects in the domain of interest. Variable symbols range over the objects in the domain. Function symbols represent mappings from tuples of objects to objects. Predicate symbols represent relations among objects in the domain or attributes of objects. A term is any expression representing an object in the domain. It can be a constant, a variable, or a function applied to a tuple of terms. An atomic formula or atom is a predicate symbol applied to a tuple of terms. A ground term is a term containing no variables. A ground atom or ground predicate is an atomic formula all of whose arguments are ground terms. Formulas are recursively constructed from atomic formulas using logical connectives and quantifiers. A positive literal is an atomic formula; a negative literal is a negated atomic formula. A KB in clausal form is a conjunction of clauses, a clause being a disjunction of literals. A definite clause is a clause with exactly one positive literal (the head, with the negative literals constituting the body). A possible world or Herbrand interpretation assigns a truth value to each possible ground predicate.

Inductive Logic Programming (ILP) systems learn clausal KBs from relational databases, or refine existing KBs [13]. Hypotheses are constructed through refinement operators that add or remove literals from clauses. In the learning from interpretations setting of ILP, the examples are databases, and the system searches for clauses that are true in them. For example, CLAUDIEN [4], starting with a trivially false clause, repeatedly forms all possible refinements of the current clauses by adding literals, and adds to the KB those that satisfy a minimum accuracy and coverage criterion. In the learning from entailment setting, the system searches for clauses that entail all positive examples of some relation and no negative ones. For example, FOIL [21] learns each definite clause by starting with the target relation as the head and greedily adding literals to the body.

MN weights have traditionally been learned using iterative scaling [5]. However, maximizing the likelihood (or posterior) using a quasi-Newton optimization method like L-BFGS has recently been found to be much faster [23]. Regarding structure learning, the authors in [5] induce conjunctive features by starting with a set of atomic features (the original variables), conjoining each current feature with each atomic feature, adding to the network the conjunction that most increases likelihood, and repeating. The work in [15] extends this to the case of conditional random fields, which are MNs trained to maximize the conditional likelihood of a set of outputs given a set of inputs.

The first attempt to learn MLNs was that in [22], where the authors used the CLAUDIEN system to learn the clauses of MLNs and then learned the 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 [2], outperformed that of [22].

Recently, in [16] 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 candidate clauses. In this way, it generates fewer candidates for evaluation. Even though it evaluates fewer candidates, after initially scoring all candidates, the algorithm attempts to add them one by one to the MLN, thus changing the MLN at almost each step, which greatly slows down the computation of the WPLL. For every candidate structure, in both [11, 16] the parameters that optimize the WPLL are set through L-BFGS that approximates the second-derivative of the WPLL by keeping a running finite-sized window of previous first-derivatives.

Regarding weight-learning, as pointed out in [11] a potentially serious problem that arises when evaluating candidate clauses using WPLL is that the optimal (maximum WPLL) weights need to be computed for each candidate. Since this involves numerical optimization, and needs to be done millions of times, it could easily make the algorithm too slow. In [15, 5] the problem is addressed by assuming that the weights of previous features do not change when testing a new one. Surprisingly, the authors in [11] found this to be unnecessary if it is used the very simple approach of initializing L-BFGS with the current weights (and zero weight for a new clause). Although in principle all weights could change as the result of introducing or modifying a clause, in practice this is very rare. Second-order, quadratic-convergence methods like L-BFGS are known to be very fast if started near the optimum [23]. This is what happened in [11]: L-BFGS typically converges in just a few iterations, sometimes one. We use the same approach for setting the parameters that optimize the WPLL.

4 Iterated Local Search

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 [9] and represent one of the most successful and widely used approaches for solving hard combinatorial problems. 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 [9]. ILS is one of these metaheuristics because it can be easily combined with other SLS methods.

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 local optima. ILS methods [9, 14] 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. The algorithm 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 subsidiary local search is performed until a local optimum *s"* is obtained. In the last step, an acceptance criterion is used to decide from which of the two local optima *s* or *s'* the search process

Algorithm 1 Structure Learning

```
Input: P:set of predicates, MLN:Markov Logic Network,
RDB:Relational Database
CLS = All clauses in MLN $\cup$ P;
LearnWeights(MLN,DB); Score = WPLL(MLN,RDB);
repeat
  BestClause = SearchBestClause(P,MLN,Score,CLS,RDB);
  if BestClause $\neq$ null then
      Add BestClause to MLN;
      Score = WPLL(MLN,RDB);
    if BestScore <= Score then
        Gain = Score - BestScore; BestScore = Score;
    end if
  end if
until BestClause = null || Gain <= minGain for two consecutive
steps
return MLN
```

is continued. The algorithm can terminate after some steps have not produced improvement or simply after a certain number of steps. The choice of the components of the ILS has a great impact on the performance of the algorithm.

As pointed out in [9] there are three good reasons to consider applying SLS algorithms instead of systematic algorithms. The first is that many problems are of a constructive nature and their instance is known to be solvable. In these situations, the goal of any search algorithm is to find a solution rather than just to decide whether a solution exists. This holds in particular for optimization problems, where the actual problem is to find a solution of sufficiently high quality. Therefore, the main advantage of a complete systematic algorithm (the ability to detect that a given problem instance has no solution) is not relevant for finding solutions of solvable instances. Secondly, in most application scenarios, the time to find a solution is limited. In these situations, systematic algorithms often have to be aborted after the given time has been exhausted, which renders them incomplete. This is problematic for many systematic optimization algorithms that search through spaces of partial solutions without computing complete solutions early in the search, and if such a systematic algorithm is aborted prematurely, usually a non solution candidate is available, while in the same situation SLS algorithms typically return the best solution found so far. Thirdly, algorithms for real-time problems should be able to deliver reasonably good solutions at any point during their execution. For optimization problems this typically means that run-time and solution quality should be positively correlated; for decision problems one could guess a solution when a time-out occurs, where the accuracy of the guess should increase with the run-time of the algorithm. This so-called any-time property of algorithms is usually very difficult to achieve, but in many situations the SLS paradigm is naturally suited for devising any time algorithms.

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 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. Moreover, one of the key advantages of SLS methods is that they can greatly speed up learning through parallel processing, where speedups proportional to the number of CPUs can be achieved [9]. We also exploit this feature of our ILS algorithm, by parallelizing multiple independent walks of ILS in separate CPUs.

5 Generative Structure Learning of MLNs through ILS

In this section we describe the ILS metaheuristic tailored to the problem of learning the structure of MLNs. Algorithm 1 iteratively adds the best clause to the current MLN until two consecutive steps have not produced improvement (however other stopping criteria could be applied). Algorithm 2 performs an iterated local search to find the best clause to add to the 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 CL_S 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).

Careful choice of the various components of Algorithm 2 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*$_{II}$ may fall again in the same local optimum and enter regions with the same value of the objective function called *plateau*, 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 (in future work we intend to adapt dynamically the nature of the perturbation). Regarding the procedure *LocalSearch*$_{II}$ we decided to use an iterative improvement approach in order to balance intensification (greedily increase solution quality by exploiting the evaluation function) and diversification (randomness induced by strong perturbation to avoid search stagnation). The *accept* function always accepts the best solution found so far.

Algorithm 2 SearchBestClause

```
Input: P:set of predicates, MLN:Markov Logic Network, BestScore: current best score, CLS: List of clauses, RDB:Relational Database)
CL$_C$ = Random Pick a clause in CLS $\cup$ P;
CL$_S$ = LocalSearch$_{II}$(CL$_S$);
BestClause = CL$_S$;
repeat
  CL'$_C$ = Perturb(CL$_S$);
  CL'$_S$ = LocalSearch$_{II}$(CL'$_C$,MLN,BestScore);
  if WPLL(BestClause,MLN,RDB) $\leq$ WPLL(CL'$_S$,MLN,RDB) then
    BestClause = CL'$_S$;
    Add BestClause to MLN;
    BestScore = WPLL(CL'$_S$,MLN,RDB)
  end if
  CL$_S$ = accept(CL$_S$,CL'$_S$);
until two consecutive steps have not produced improvement
Return BestClause
```

6 Experiments

6.1 Datasets

We carried out experiments on two publicly-available databases: the UW-CSE database used by [11, 22, 16] (available at http://alchemy.cs.washington.edu/data/uw-cse) and the Cora dataset originally labeled by Andrew McCallum. Both represent standard relational datasets and are used for two important relational tasks: Cora for entity resolution and UW-CSE for social network analysis. For Cora we used a cleaned version from [24], with five splits for cross-validation.

The published UW-CSE dataset consists of 15 predicates divided into 10 types. Types include: publication, person, course, etc. Predicates include: Student(person), Professor(person), AdvisedBy(person1, person2), TaughtBy(course, person, quarter), Publication (paper, person) etc. The dataset contains a total of 2673 tuples (true ground atoms, with the remainder assumed false). The Cora dataset consists of 1295 citations of 132 different computer science papers, drawn from the Cora Computer Science Research Paper Engine. The task is to predict which citations refer to the same paper, given the words in their author, title, and venue fields. The labeled data also specify which pairs of author, title, and venue fields refer to the same entities. We performed experiments for each field in order to evaluate the ability of the model to deduplicate fields as well as citations. Since the number of possible equivalences is very large, we used the canopies found in [24] to make this problem tractable.

6.2 Systems and Methodology

We implemented Algorithm 1 (ILS) in the Alchemy package [12]. We used the implementation of L-BFGS in Alchemy to learn maximum WPLL weights. We compared our algorithm performance with the state-of-the-art algorithms for generative structure learning of MLNs: BS (Beam Search) of [11] and BUSL (Bottom-Up Structure Learning) of [16].

In the UW-CSE domain, we used the same leave-one-area-out methodology as in [22]. In the Cora domain, we performed cross-validation. For each system on each test set, we measured the conditional log-likelihood (CLL) and the area under the precision-recall curve (AUC) for all the predicates. The advantage of the CLL is that it directly measures the quality of the probability estimates produced. The advantage of the AUC is that it is insensitive to the large number of true negatives (i.e., ground atoms that are false and predicted to be false). The CLL of a query predicate is the average over all its groundings of the ground atoms log-probability given evidence. 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.

For all algorithms, we used the default parameters of Alchemy changing only the following ones: maximum variables per clause = 5 for UW-CSE and 6 for Cora; penalization of WPLL: 0.01 for UW-CSE and 0.001 for Cora. For L-BFGS: convergence threshold = 10^{-5} (tight) and 10^{-4} (loose); minWeight = 0.5 for UW-CSE for BUSL as in [16], 1 for BS as in [11] and 1 for ILS; minGain = 0.05 for ILS. For ILS we used a multiple independent walk parallelism, assigning an instance of the algorithm to a separate CPU on a cluster of Intel Core2 Duo 2.13 GHz CPUs.

6.3 Results

After learning the structure, we performed inference on the test fold for both datasets by using MC-SAT [20] with number of steps = 10000 and simulated annealing temperature = 0.5. For each experiment, all the groundings of the query predicates on the test fold were commented. 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 the relative AUC (for AUC we used the method proposed in [3]). For ILS we report the best performance in terms of CLL among ten parallel independent walks. Both CLL and AUC results (Table 1) are averaged over all predicates of the domain. Learning times are reported in Table 2. For BS in the Cora domain we were not able to report results, since structure learning with this algorithm did not finish in 45 days. BS is heavily slowed by its systematic top-down nature that tends to evaluate a very large number of candidates. In the UW-CSE domain, BS gets easily stuck in local optima due to its greedy strategy.

Table 1. Accuracy results for all algorithms

	UW-CSE		CORA	
Algorithm	CLL	AUC	CLL	AUC
BS	-0.312±0.046	0.320	-	-
BUSL	-0.074±0.014	0.431	-0.196±0.003	0.201
ILS	-0.069±0.016	0.432	-0.102±0.003	0.225

In both domains, ILS gives the best overall results in terms of CLL and AUC. BUSL is competitive with ILS in terms of accuracy but is much slower. Even though BUSL evaluates fewer candidates than ILS, it changes the MLN completely at each step, thus calculating the WPLL becomes very expensive. In ILS this does not happen because, like in [11], at each step L-BFGS is initialized with the current weights (and zero weight for a new clause) and it converges in a few iterations. We empirically observed that ILS is very effective in escaping local optima and further improvements can be achieved by dynamically adapting the strength of the perturbation operator.

Table 2. Average learning times for all algorithms (in minutes)

Algorithm	UW-CSE	CORA
BS	335	-
BUSL	618	9350
ILS	148	1597

7 Conclusion and Future Work

Markov logic networks are a powerful representation that combine first-order logic and probability. We have introduced an iterated local search algorithm for learning the structure of Markov Logic Networks. The approach is based on a biased sampling of the set of local optima focusing 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. We have shown through experiments in two real-world domains that the proposed algorithm performs better than state-of-the-art structure learning algorithms for MLNs. Future work includes implementing more sophisticated parallel models suich as MPI (Message Passing Interface) or PVM (Parallel Virtual Machine), dynamically adapting the nature of perturbations in ILS, using a Metropolis criterion in the acceptance function of ILS.

ACKNOWLEDGEMENTS

We thank Pedro Domingos and Stanley Kok for helpful discussions, Marc Sumner for help on using Alchemy and Lilyana Mihalkova for help on BUSL.

REFERENCES

[1] F. Bacchus, *Representing and Reasoning with Probabilistic Knowledge*, Cambridge, MA: MIT Press, 1990.

[2] J. Besag, 'Statistical analysis of non-lattice data', *Statistician*, **24**, 179–195, (1975).

[3] J. Davis and M. Goadrich, 'The relationship between precision-recall and roc curves', in *Proc. 23rd ICML*, pp. 233–240, (2006).

[4] L. De Raedt and L. Dehaspe, 'Clausal discovery', *Machine Learning*, **26**, 99–146, (1997).

[5] S. Della Pietra, V. Della Pietra, and J. Laferty, 'Inducing features of random fields', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **19**, 380–392, (1997).

[6] M. R. Genesereth and N. J. Nilsson, *Logical foundations of artificial intelligence*, San Mateo, CA: Morgan Kaufmann., 1987.

[7] L. Getoor and B. Taskar, *Introduction to Statistical Relational Learning*, MIT, 2007.

[8] J. Halpern, 'An analysis of first-order logics of probability', *Artificial Intelligence*, **46**, 311–350, (1990).

[9] H. H. Hoos and T. Stutzle, *Stochastic Local Search: Foundations and Applications*, Morgan Kaufmann, San Francisco, 2005.

[10] K. Kersting and L. De Raedt, 'Towards combining inductive logic programming with bayesian networks', in *Proc. 11th Int'l Conf. on Inductive Logic Programming*, pp. 118–131. Springer, (2001).

[11] S. Kok and P. Domingos, 'Learning the structure of markov logic networks', in *Proc. 22nd Int'l Conf. on Machine Learning*, pp. 441–448, (2005).

[12] S. Kok, P. Singla, M. Richardson, and P. Domingos, 'The alchemy system for statistical relational ai', Technical report, Department of CSE-UW, Seattle, WA, http://alchemy.cs.washington.edu/, (2005).

[13] N. Lavrac and S. Dzeroski, *Inductive Logic Programming: Techniques and applications*, UK: Ellis Horwood, Chichester, 1994.

[14] H.R. Loureno, O. Martin, and T. Stutzle, 'Iterated local search', in *Handbook of Metaheuristics*, 321–353, F. Glover and G. Kochenberger, Kluwer Academic Publishers, Norwell, MA, USA, (2002).

[15] A. McCallum, 'Efficiently inducing features of conditional random fields', in *Proc. UAI-03*, pp. 403–410, (2003).

[16] L. Mihalkova and R. J. Mooney, 'Bottom-up learning of markov logic network structure', in *Proc. 24th Int'l Conf. on Machine Learning*, pp. 625–632, (2007).

[17] D. Koller N. Friedman, L. Getoor and A. Pfeffer, 'Learning probabilistic relational models', in *Proc. 16th Int'l Joint Conf. on AI (IJCAI)*, pp. 1300–1307. Morgan Kaufmann, (1999).

[18] J. Neville and D. Jensen, 'Dependency networks for relational data', in *Proc. 4th IEEE Int'l Conf. on Data Mining*, pp. 170–177. IEEE Computer Society Press., (2004).

[19] N. Nilsson, 'Probabilistic logic', *Artificial Intelligence*, **28**, 71–87, (1986).

[20] H. Poon and P. Domingos, 'Sound and efficient inference with probabilistic and deterministic dependencies', in *Proc. 21st Nat'l Conf. on AI, (AAAI)*, pp. 458–463. AAAI Press, (2006).

[21] J. R. Quinlan, 'Learning logical definitions from relations', *Machine Learning*, **5**, 239–266, (1990).

[22] M. Richardson and P. Domingos, 'Markov logic networks', *Machine Learning*, **62**, 107–236, (2006).

[23] F. Sha and F. Pereira, 'Shallow parsing with conditional random fields', in *Proc. HLT-NAACL-03*, pp. 134–141, (2003).

[24] P. Singla and P. Domingos, 'Entity resolution with markov logic', in *Proc. ICDM-2006*, pp. 572–582. IEEE Computer Society Press, (2006).

[25] P. Singla and P. Domingos, 'Markov logic in infinite domains', in *Proc. 23rd UAI*, pp. 368–375. AUAI Press, (2007).

[26] J. S. Wellman, M. Breese and R. P. Goldman, 'From knowledge bases to decision models', *Knowledge Engineering Review*, (1992).