Machine Learning manuscript No.
(will be inserted by the editor)

Learning Directed Relational Models With Recursive Dependencies

Oliver Schulte and Hassan Khosravi
Tong Man
oschulte@cs.sfu.ca, hkhosrav@cs.sfu.ca,
mantong01@gmail.com

Abstract Recently, there has been an increasing interest in generative models that represent probabilistic patterns over both links and attributes. A common characteristic of relational data is that the value of a predicate often depends on values of the same predicate for related entities. For directed graphical models, such recursive dependencies lead to cycles, which violates the acyclicity constraint of Bayes nets. In this paper we present a new approach to learning directed relational models which utilizes two key concepts: a pseudo likelihood measure that is well defined for recursive dependencies, and the notion of stratification from logic programming. An issue for modelling recursive dependencies with Bayes nets are redundant edges that increase the complexity of learning. We propose a new normal form format that removes the redundancy, and prove that assuming stratification, the normal form constraints involve no loss of modelling power. Emprirical evaluation compares our approach to learning recursive dependencies with undirected models (Markov Logic Networks). The Bayes net approach is orders of magnitude faster, and learns more recursive dependencies, which lead to more accurate predictions.

1 Introduction: Relational Data and Recursive Dependencies

Relational data are very common in real-world applications, ranging from social network analysis to enterprise databases. A phenomenon that distinguishes relational data from single-population data is that the value of an attribute for an entity can be predicted by the value of the same attribute for related entities; this phenomenon has been called a "nearly ubiquitious characteristic" of relational datasets [1, Sec.1]. For example, whether individual a smokes may be predicted

Supported by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada. We are indebted to reviewers of the ILP conference and the Machine Learning journal for helpful comments.

School of Computing Science
Simon Fraser University
Vancouver-Burnaby, B.C., Canada

by the smoking habits of a's friends. This pattern can be represented by clausal notation such as $Smokes(X) \leftarrow Smokes(Y), Friend(X, Y)$.

Different subfields concerned with relational data have introduced different terms for this phenomenon. From a logic programming perspective, it is natural to speak of a *recursive dependency*, where a predicate depends on itself. In statistical-relational learning, Jensen and Neville introduced the term *relational autocorrelation* in analogy with temporal autocorrelation [2,1]. In multi-relational data mining, such dependencies are found by considering *self-joins* where a table is joined to itself [3]. We will use both the terms recursive dependency and autocorrelation. The former emphasizes the format of the rules we consider, whereas the latter distinguishes the probabilistic dependencies we model from deterministic logical entailment.

In this paper we investigate a new approach to learning recursive dependencies with Bayes nets, specifically Poole's Parametrized Bayes Nets (PBNs) [4]; however, our results apply to other directed relational models as well, such as Probabilistic Relational Models (PRMs) [5] and Bayes Logic Programs (BLPs) [6]. Two key difficulties are well known for learning recursive dependencies using directed models.

(1) Recursive dependencies lead to cyclic dependencies among ground facts [7–9]. The cycles make it difficult to define a model likelihood function for observed ground facts in the data, which is an essential component of statistical model selection. To define a model likelihood function for Bayes net search, we utilize Schulte's recent relational Bayes net pseudo likelihood [10] that measures the fit of a PBN to a relational database and is well-defined even in the presence of recursive dependencies. The recent efficient learn-and-join algorithm [11] searches for models that maximize the pseudo likelihood. In this paper we evaluate the pseudo likelihood approach on datasets with strong autocorrelations.

(2) A related problem is that defining valid probabilistic inferences in cyclic models is difficult. To avoid cycles in the ground model while doing inference, Khosravi *et al.* proposed converting a learned Bayes net to an undirected model using the standard moralization procedure [11]. In graphical terms, moralization connects all co-parents of a node, then omits edge directions. Inference with recursive dependencies can then be carried out using Markov Logic Networks (MLNs), a prominent relational model class that combines the syntax of logical clauses with the semantics of Markov random fields [8]. The moralization approach combines the efficiency and scalability of Bayes net learning with the high-quality inference procedures of MLNs.

(3) A third problem that we observed in research with autocorrelation datasets is that the repetition of predicates causes additional complexity in learning if each predicate instance is treated as a separate random variable. For example, suppose that the dependence of smoking on itself is represented in a Bayes net with a 3-node structure

$$Smokes(Y) \rightarrow Smokes(X) \leftarrow Friend(X, Y).$$

Now suppose that we also include a binary attribute *Cancer* that indicates whether a person has cancer or not. Then a Bayes net learner would potentially consider two edges, $Smokes(X) \rightarrow Cancer(X)$ and $Smokes(Y) \rightarrow Cancer(Y)$. If there is in fact a statistical dependence of cancer on smoking, then each of these edges correctly

represents this dependency, but one of them is redundant, as the logical variables X, Y are interchangeable placeholders for the same domain of entities. We propose a normal form for Parametrized Bayes nets that eliminates such redundancies: For each function/predicate symbol, designate one node as the *main node*. Then constrain the Bayes net such that only main nodes have edges pointing into them. In the example above, if $Cancer(X)$ is the main functor for *Cancer*, the edge $Smokes(Y) \rightarrow Cancer(Y)$ is forbidden. We prove that this constraint incurs no loss of expressive power in the following sense: if a Bayes net B is stratified, then there is a Bayes net $B^{'}$ in main functor format such that B and $B^{'}$ induce the same ground graph for every relational database instance. We show how the learn-and-join algorithm can be extended to incorporate this constraint.

We compared our learning algorithms with two state-of-the-art Markov Logic Network methods using public domain datasets The pseudo likelihood algorithm with main functor format is orders of magnitude faster, and learns more recursive dependencies, which lead to more accurate predictions.

Paper Organization. We review the relevant background and define our notation. We prove theoretical results regarding relational autocorrelation: the first gives a necessary and sufficient condition for a ground Parametrized Bayes net to be acyclic, the second is the normal form theorem mentioned. We describe the normal form extension of the learn-and-join algorithm. Our simulations evaluate the ability of the extended algorithm to learn recursive dependencies, compared to Markov Logic Network learner.

Contributions. The main contributions may be summarized as follows.

1. A new formal form theorem for Parametrized Bayes nets that addresses redundancies in modelling autocorrelations.
2. An extension of the learn-and-join algorithm for learning Bayes nets that include autocorrelations.
3. An evaluation of the pseudo-likelihood measure [10] for learning autocorrelations.

2 Related Work.

Parametrized Bayes nets (PBNs) are a basic statistical-relational model due to Poole [4]. PBNs utilize the functor concept from logic programming to connect logical structure with random variables.

Bayes Net Learning for Relational Data. Adaptations of Bayes net learning methods for relational data have been considered by several researchers [11, 12, 7, 13, 6]. Issues connected to learning Bayes nets with recursive dependencies are discussed in detail by Ramon *et al.* [7]. Early work on this topic required ground graphs to be acyclic [6, 13]. For example, Probabilistic Relational Models allow dependencies that are cyclic at the predicate level as long as the user guarantees acyclicity at the ground level [13]. A recursive dependency of an attribute on itself is shown as a self loop in the model graph. If there is a natural ordering of the ground atoms in the domain (e.g., temporal), there may not be cycles in the ground graph; but this assumption is restrictive in general. The generalized order-search

of Ramon *et al.* instead resolves cycles by learning an ordering of ground atoms. A basic difference between our work and generalized order search is that we focus on learning at the *predicate level*. Our algorithm can be combined with generalized order-search as follows: First use our algorithm to learn a Bayes net structure at the predicate/class level. Second carry out a search for a good ordering of the ground atoms. We leave integrating the two systems for future work.

Stratified Models. Stratification is a widely imposed condition on logic programs, because it increases the tractability of reasoning with a relatively small loss of expressive power. Our definition is very similar to the definition of local stratification in logic programming [14]. The difference is that levels are assigned to predicates/functions rather than ground literals, so the definition does not need to distinguish positive from negative literals. Related ordering constraints appear in the statistical-relational literature [15, 13].

3 Background and Notation

We define the target model class of Parametrized Bayes nets. Then we briefly discuss the problems arising from cyclic dependencies that have been addressed in our previous work. The next section discusses the redundancy problem that has not been previously addressed.

3.1 Bayes Nets for Relational Data

We follow the original presentation of Parametrized Bayes nets due to Poole [4]. A **functor** is a function symbol or a predicate symbol. In this paper we discuss only functors with a finite range of possible values. A functor whose range is $\{T, F\}$ is a **predicate**, usually written with uppercase letters like P, R. A **parametrized random variable** or **functor node** or simply **fnode** is of the form $f(X_1, \ldots, X_k) = f(\mathbf{X})$ where f is a functor and each first-order variable X_i is of the appropriate type for the functor. If a functor node $f(\boldsymbol{\tau})$ contains no variable, it is **ground node**. An assignment of the form $f(\boldsymbol{\tau}) = a$, where a is a constant in the range of f, is an **atom**; if $f(\boldsymbol{\tau})$ is ground, the assignment is a **ground atom**. A **population** is a set of individuals, corresponding to a domain or type in logic. Each first-order variable X is associated with a population. An **instantiation** or **grounding** for a set of variables $X_1, \ldots, X_k$ assigns to each variable X_i a constant from the population of X_i. Getoor and Grant discuss the applications of function concepts as a unifying language for statistical-relational modelling [16].

Figure 1 shows a simple database instance in the E-R format [8] and the ground atoms in functor notation. The results in this paper extend to functors built with nested functors, aggregate functions [17], and quantifiers; for the sake of notational simplicity we do not consider more complex functors explicitly. A **table join** of two or more tables contains the rows in the Cartesian products of the tables whose values match on common fields. In logical terms, a join corresponds to a conjunction [18].

A **Bayes net structure** is a directed acyclic graph (DAG) G, whose nodes comprise a set of random variables. A **family** is a child node together with its parents. A Bayes net (BN) is a directed acyclic graph with conditional probability

People		
Name	Smokes	Cancer
Anna	T	T
Bob	T	F

Friend	
Name1	Name2
Anna	Bob
Bob	Anna

Smokes(Anna) =T	Friend(Anna, Bob) = T
Smokes(Bob) =T	Friend(Bob, Anna) = T
Cancer(Anna) = T	Friend(Anna, Anna) = F
Cancer(Bob) = F	Friend(Bob, Bob) = F

Fig. 1 Left: A simple relational database instance. Right: The ground atoms for the database, and their values as specified by the database, using functor notation.

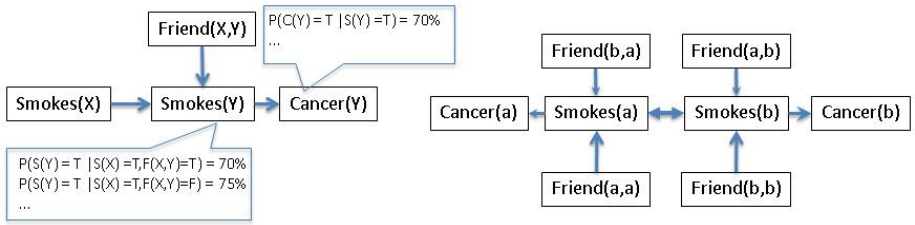

Fig. 2 A Parametrized Bayes Net and its grounding for two individuals a and b. The double arrow $\leftrightarrow$ is equivalent to two directed edges, hence a cycle between $Smokes(a)$ and $Smokes(b)$.

parameters. A **Parametrized Bayes Net** (PBN) is a Bayes net whose nodes are functor nodes. A **ground** PBN $\overline{B}$ is a directed graph derived from B by instantiating the variables in the functor nodes in B with all possible constants. Figure 2 illustrates a Parametrized Bayes Net for the dataset in Figure 1 and its grounding. In what follows, we often refer to PBNs simply as Bayes Nets.

3.2 Relational Pseudo-Likelihood Measure for Bayes Nets

Score-based learning algorithms for Bayes nets require the specification of a numeric model selection score that measures how well a given Bayes net model fits observed data. A common approach to defining a score for a relational database is known as *knowledge-based model construction* [19–22]. The basic idea is to consider the ground graph for a given database, illustrated in Figure 2. A given database like the one in Figure 1 specifies a value for each node in the ground graph. Thus the likelihood of the Parametrized Bayes net for the database can be defined as the likelihood assigned by the ground graph to the facts in the database following the usual Bayes net product formula.

In the presence of recursive dependencies, the grounding approach runs into the *cyclicity problem*: As illustrated in Figure 2, the ground graph may contain a cycle. It is well-known that such cycles arise in the presence of *self-relationships* that relate entities of the same type [23] (e.g., *Friend* is a self-relationship that relates one person to another). Schulte [10] proposed a way to measure the fit of a Bayes net model to relational data that does not require acylicity: the idea is to consider a *random* grounding of the 1st-order variables in the Parametrized Bayes

Γ	X	Y	F(X,Y)	S(X)	S(Y)	C(Y)	P_B^γ	$ln(P_B^\gamma)$
γ_1	Anna	Bob	T	T	T	F	0.105	-2.254
γ_2	Bob	Anna	T	T	T	T	0.245	-1.406
γ_3	Anna	Anna	F	T	T	T	0.263	-1.338
γ_4	Bob	Bob	F	T	T	F	0.113	-2.185

Table 1 The computation of the random grounding pseudo likelihood for the Bayes net of Figure 2 and the database of Figure 1. Each row is a simultaneous grounding of *all* 1st-order variables in the Bayes net. The values of functors for each grounding defines an assignment of values to the Bayes net nodes. The Bayes net assigns a likelihood for each grounding using the standard product formula. The rounded numbers shown were obtained using the CP parameters of Figure 2 together with $P_B(Smokes(X) = T) = 1$ and $P_B(Friend(X, Y) = T) = 1/2$, chosen for easy computation. The pseudo log-likelihood is the average of the log-likelihoods for each grounding, given by $-(2.254 + 1.406 + 1.338 + 2.185)/4 \approx -1.8$.

net, rather than a complete grounding. The pseudo log-likelihood is defined as follows.

1. Randomly select a grounding for *all* 1st-order variables that occur in the Bayes Net. The result is a ground graph with as many nodes as the original Bayes net.
2. Look up the value assigned to each ground node in the database. Compute the log-likelihood of this joint assignment using the usual product formula; this defines a log-likelihood for the random instantiation.
3. The expected value of this log-likelihood is the *pseudo log-likelihood* of the database given the Bayes net.

Table 1 shows the computation of the pseudo likelihood assigned to our toy database by the Bayes net of Figure 2. A naive computation of the pseudo log-likelihood involves enumerating all possible groundings of the 1st-order Bayes net, which is infeasible for realistic population sizes. However, there is an equivalent tractable closed-form expression [10, Prop. 2]. The closed form is almost exactly the same as the standard log-likelihood for a Bayes net given a single data table, except that row counts in the data table are replaced by event frequencies in the database. Schulte shows that the learn-and-join algorithm [11] (implicitly) maximizes the pseudo-likelihood [10].

3.3 Inference and Moralization

In the presence of cycles, the ground graph does not provide a valid basis for probabilistic inference. Several researchers advocate the use of undirected rather than directed models because cycles do not arise with the former [8, 9, 1]. Undirected Markov random fields are therefore important models for inference with relational data. The recently introduced moralization approach [11] is essentially a hybrid method that uses directed models for learning and undirected models for inference.

Bayes net graphs can be converted to undirected Markov net graphs through the standard **moralization** method [8, 12.5.3]: Connect ("marry") all co-parents, then omit edge directions. We refer to the result of this conversion as a **Moralized Bayes Net**. Figure 3 shows the Moralized Bayes Net of Figure 2. Valid probabilistic inferences can then be defined in terms of the ground Markov network, also shown in Figure 3.

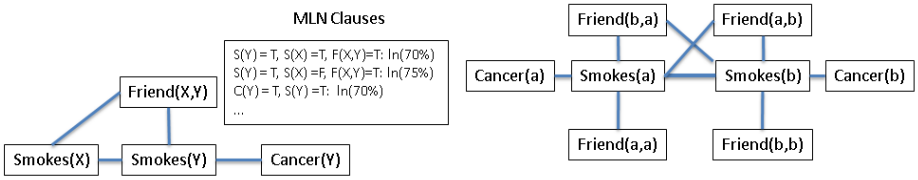

Fig. 3 The moralized Bayes net of Figure 2 and its ground Markov network for the database of Figure 1.

Pedro Domingos has connected Markov random fields to logical clauses by showing that 1st-order formulas can be viewed as templates for Markov random fields whose nodes comprise ground atoms that instantiate the formulas. **Markov Logic Networks** (MLNs) are presented in detail by Domingos and Richardson [8]. The qualitative component or structure of an MLN is a finite set of formulas or clauses $\{\phi_i\}$, and its quantitative component is a set of weights $\{w_i\}$, one for each clause. The Markov Logic Network corresponding to a Moralized Bayes net simply contains one conjunctive clause for each possible state of each family. Thus the Markov Logic Network for a moralized PBN contains a conjunction for each conditional probability specified in the Bayes net. For converting the Bayes net conditional probabilities to MLN clause weights, Domingos and Richardson suggest using the log of the conditional probabilities as the clause weight [8, 12.5.3]. This is the standard conversion for propositional Bayes nets. Figure 3 illustrates the MLN clauses obtained by moralization using log-probabilities as weights.

4 Stratification and Recursive Dependencies

In this section we first consider analytically the relationship between cycles in a ground Bayes net and orderings of the functors that appear in the nonground Bayes net. It is common to characterize a Logic Program by the orderings of the functors that the logic program admits [24]; we adapt the ordering concepts for Bayes nets. The key ordering concept is the notion of a *level mapping*. We apply it to Bayes nets as follows.

Definition 1 Let B be an Parametrized Bayes net. A **level mapping** assigns to each functor f in B a nonnegative integer $level(f)$.

- A Bayes net is **strictly stratified** if there is a level mapping such that for every edge $f(\tau) \rightarrow g(\tau)$, we have $level(f) < level(g)$.
- A Bayes net is **stratified** if there is a level mapping such that for every edge $f(\tau) \rightarrow g(\tau)$, we have $level(f) \leq level(g)$.

Strict stratification corresponds to the concept of a hierarchical rule set [24]. Since it implies that one fnode cannot be an ancestor of another fnode with the same functor, strict stratification rules out recursive clauses. Stratification with a weak inequality, by contrast, does allow the representation of autocorrelations. Stratification is a widely imposed condition on logic programs, because it increases

the tractability of reasoning with a relatively small loss of expressive power [24, Sec.3.5],[14]. We next show that strict stratification characterizes the absence of cycles in a ground Bayes net. The proof is in Section 8.

Proposition 1 *Let B be a Parametrized Bayes net, and let $\mathcal{D}$ be a database instance such that every population (entity type) has at least two members. Then the ground graph $\overline{B}$ for $\mathcal{D}$ is acyclic if and only if the Bayes net B is strictly stratified.*

This result shows that cyclic dependencies arise precisely when a node associated with one functor is an ancestor of another node associated with the same functor.[1] This in turn is exactly the graphical condition associated with recursive dependencies, which means that recursive dependencies and cyclic dependencies are closely connected phenomena.

While stratified Bayes nets have the expressive power to represent autocorrelations, there is potential for additional complexity in learning if each functor is treated as a separate random variables. We discuss this issue in the next subsection and propose a normal form constraint for resolving it.

4.1 Stratification and the Main Functor Node Format

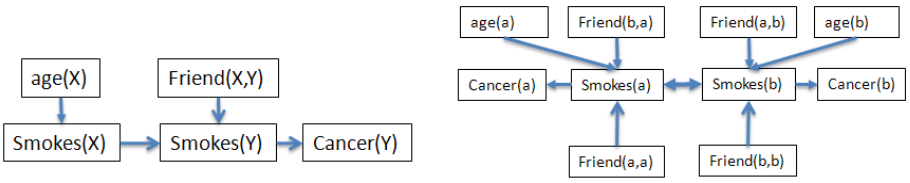

Fig. 4 A stratified Bayes net with different parent predictors for $Smokes(X)$ and $Smokes(Y)$, and its grounding for two individuals a and b.

Consider the left Bayes net in Figure 4. If we treat $Smokes(X)$ and $Smokes(Y)$ as entirely separate variables, learning needs to consider additional edges similar to those already in the Bayes net, like

$$Smokes(X) \rightarrow Cancer(X)$$

and

$$age(Y) \rightarrow Smokes(Y).$$

However, such edges are redundant because the 1st-order variables X and Y are interchangeable as they refer to the same entity set. In terms of ground instances, the two edges connect exactly the same ground instances.

Redundant edges can be avoided if we restrict the model class to the main functor format, where for each function symbol f, there is a *main functor node*

[1] In some statistical-relational models such as PRMs and LBNs, the ground graph is constructed somewhat using the known relational context to add fewer edges [13,15]. In that case strict stratification remains sufficient for acyclicity but may no longer be necessary; see Section 2.

$f(\boldsymbol{\tau})$ such that all other functor nodes $f(\boldsymbol{\tau}')$ associated with the same functor are sources in the graph, that is, they have no parents. The intuition for this restriction is that statistically, two functors with the same function symbol are equivalent, so it suffices to model the distribution of these functors conditional on a set of parents just once. This leads to the following formal definition.

Definition 2 A Bayes net B is in **main functor node form** if for every functor f of B, there is a distinguished functor node $f(\boldsymbol{\tau})$, called the **main functor node** for f, such that every other functor node $f(\boldsymbol{\tau}')$, where $\boldsymbol{\tau}' \neq \boldsymbol{\tau}$, has no parents in B.

Example. The Bayes net of Figure 4 is not in main functor form because we have two functor nodes for *Smokes* with nonzero indegree. The Bayes net in Figure 5 is in main variable format where Smokes(Y) is the main functor for Smokes(X). In terms of ground instances, the two Bayes nets have exactly the same ground graph.

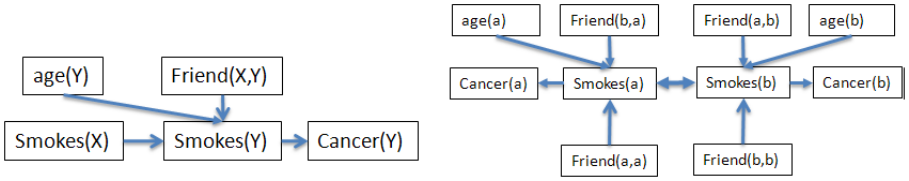

Fig. 5 An Bayes net in main functor format where $Smokes(Y)$ is the main functor for $Smokes(X)$. The ground Bayes net is the same as the ground Bayes net for the graph of Figure 4.

The next proposition shows that this equivalence holds in general: For any Bayes net B there is an equivalent Bayes net B' in main functor node form. This claim is established constructively by showing how the orginal B can be transformed into B'. The transformation procedure is a conceptual aid, rather than an algorithm to be used in practice; to build a practical learning algorithm, we simply restrict the Bayes net candidates to be in main functor form (see Section 5 below). It is easy to see that we can make *local* changes to the 1st-order variables such that all child nodes for a given functor are the same. For instance, in the Bayes net of Figure 4 we can first substitute Y for X to change the edge $age(X) \rightarrow Smokes(X)$ into the edge $age(Y) \rightarrow Smokes(Y)$. Then we delete the former edge and add the latter, that is, we make $age(Y)$ a parent of $Smokes(Y)$. Figures 4 and 5 illustrate that the original and transformed Bayes nets have the same ground graph. However, in general the change of variables may introduce cycles in the Bayes net. The basis for the next proposition is that if the original Bayes net is stratified, the transformed functor node graph is guaranteed not to contain cycles. The proof is in Section 8.

Theorem 1 *Let B be a stratified Bayes net. Then there is a Bayes net B' in main functor form such that for every database $\mathcal{D}$, the ground graph $\overline{B}$ is the same as the ground graph $\overline{B'}$.*

4.2 Discussion.

Even if Bayes nets with or without the main functor constraints have the same groundings, at the variable or class level the two models may not be equivalent. For instance, the model of Figure 4 implies that $age(X)$ is independent of $Friend(X, Y)$ given $Smokes(X)$. But in the model of Figure 5, the node $age(Y)$ is dependent on (d-connected with) $Friend(X, Y)$ given $Smokes(Y)$. The transformed model represents more of the dependencies in the ground graph. For instance, the ground nodes $age(a)$ and $Friend(b, a)$ are both parents of the ground node $Smokes(a)$, and hence d-connected given $Smokes(a)$.

In general, the Bayes net that satisfy the main functor constraint feature more dependencies and nodes with more parents than Bayes nets without. If the dependencies do not exist in the data, the independencies are not captured in the Bayes net graph, but can be represented in the conditional probability table, or using a more flexible representation. For instance, in a Bayes Logic Program [6], we may have two Bayesian clauses[2]

$$Smokes(Y) \leftarrow age(Y)$$

and

$$Smokes(Y) \leftarrow Smokes(X), Friend(X, Y).$$

In a Parametrized Bayes Net, the two clauses are effectively merged into a single clause

$$Smokes(Y) \leftarrow age(Y), Smokes(X), Friend(X, Y).$$

Fundamentally, the merging occurs because the graphical format does not distinguish different sets of parents, not because of the main functor node form.

5 The Learn-and-Join Structure Algorithm With Recursive Dependencies

Khosravi *et al.* present the learn-and-join structure learning algorithm. Schulte shows that the learn-and-join algorithm maximizes the relational pseudo likelihood score (Section 3.2). The algorithm upgrades a single-table Bayes net learner for relational learning. It learns dependencies among descriptive attributes conditional on the existence of a relationship, or a chain of relationships, between them. We describe the fundamental ideas of the algorithm; for details and pseudocode please see [11]. The key idea is to build a Bayes net for the entire database by level-wise search through the *table join lattice*. The user chooses a single-table Bayes net learner. The learner is applied to table joins of size 1, that is, regular data tables. Then the learner is applied to table joins of size $s, s + 1, \ldots$, with the constraint that the absence or presence of learned edges from smaller join tables is propagated to larger join tables. These constraints are implemented by keeping a global cache of forbidden and required edges. *Implementing the main functor format simply requires adding all edges to the forbidden edge cache that do not point to main functor nodes.* Thus the main functor format provides constraints that reduce the complexity of learning. Algorithm 1 provides pseudocode for the case of a single

[2] BLP notation uses | instead of $\leftarrow$ for Bayesian clauses.

self-relationship R. The presentation for the single-relation case is simpler than for the multi-relational case and highlights the differences with the previous version of the learn-and-join algorithm [11]. Extending the algorithm to the multi-relational case can be done using the lattice search framework; the details were provided in previous work [11].

Algorithm 1: Pseudocode for structure learning (Single Self-Relationship)

Input: Database $\mathcal{D}$ with self-relationship R on entity table E.
Output: PBN graph G for $\mathcal{D}$
Calls: PBN: Any propositional Bayes net learner that accepts edge constraints and a single table of cases as input.
Notation: PBN(T, Econstraints) denotes the output DAG of PBN. Get-Constraints(G) specifies a new set of edge constraints, namely that all edges in G are required, and edges missing between variables in G are forbidden.

1: Add descriptive attributes of E and R to G. {These are the main functor nodes for the attributes.}
2: Add a duplicate node, for each descriptive attribute of E to G. {The duplicates are auxilliary nodes, not main functor nodes.}
3: Add a boolean indicator B_R for relationship table R to G.
4: Econstraints = $\emptyset$ {Required and Forbidden edges}
5: Econstraints += Get-Constraints(PBN(E , $\emptyset$)).
6: J := join of R, E, E.
7: MainFunctorConstraints := forbid edges into attribute nodes of E that are not main functor nodes.
8: Econstraints += MainFunctorConstraints.
9: Econstraints += Get-Constraints(PBN(J, Econstraints).
10: G := Set of all required edges from Econstraints.
11: If there is an edge $u \rightarrow v$ from an auxiliary node to a main functor node, add an edge $B_R \rightarrow v$ to G.
12: Return G.

5.1 Example of Algorithm.

We consider a the self-relationship *Friend* defined on the *People* entity set. Figure 6 illustrates the construction visually.

1. Applying the single-table Bayes net learner to the *People* table may produce a single-edge graph $Smokes(Y) \rightarrow Cancer(Y)$. (Line 5)
2. Then form the join data table

$$J = Friend \bowtie People \bowtie People$$

(Line 6). The Bayes net learner is applied to J, with the following constraints.
   (a) From the *People* Bayes net, there must be an edge $Smokes(Y) \rightarrow Cancer(Y)$, since $Cancer(Y)$.
   (b) No edges may point into $Smokes(X)$ or $Cancer(X)$, since these are not the main functor nodes for the functors *Smokes* and *Cancer* (Line 8).

The Bayes net learner applied to the join table J then may find an edge $Smokes(X) \rightarrow Smokes(Y)$ (Line 9). Since the dependency represented by this edge is valid only

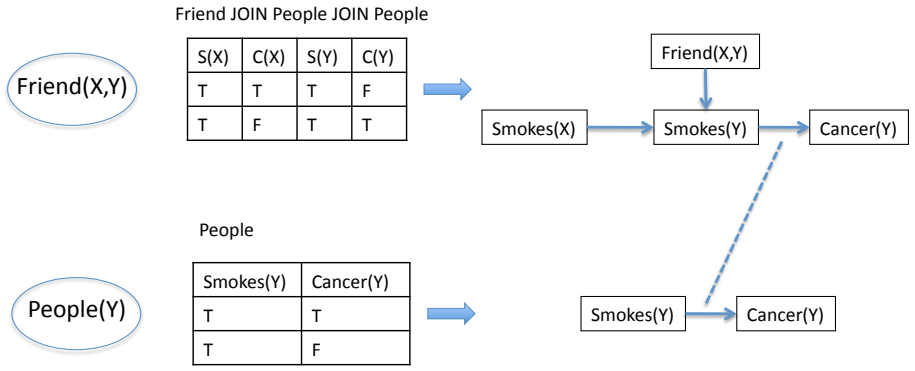

Fig. 6 The 2-net lattice associated with the DB instance of Figure 1. The figure shows the data tables associated with the only entity table *People* and the only relationship table *Friend*. The block arrow indicates that the output of a single-table Bayes net learner on the data table is the Bayes net shown. The dashed line that connects the two edges $Smokes(Y) \rightarrow Cancer(Y)$ indicates that this edge is propagated from the lower-level Bayes net to the higher-level Bayes net.

for pairs of people that are friends (i.e., conditional on $Friend(X, Y) = T$), the algorithm adds an edge $Friend(X, Y) \rightarrow Smokes(Y)$ (Line 11). In this example, functor node $Cancer(X)$ is disconnected, so the figure does not show it.

Discussion. The learn-and-join algorithm finds a structure that maximizes the pseudo-likelihood described in Section 3.2 [10]. Khosravi *et al.* discuss the time complexity of the basic learn-and-join algorithm and show that the edge-inheritance constraint essentially keeps the model search space size constant even as the number of nodes considered grows with larger table joins. For the learn-and-join algorithm, the main computational challenge in scaling to larger table joins is therefore not the increasing number of columns (attributes) in the join, but only the increasing number of rows (tuples). The main functor constraint contributes further to decreasing the search space. For instance, suppose that we have k duplicate nodes and n nodes in total. Then for each duplicate node, there are $2(n-1)$ possible directed adjacencies. The main functor constraint eliminates a possible direction for adjacencies involving duplicate nodes, hence removes $k(n-1)$ directed adjacencies from consideration.

6 Evaluation

All simulations were done on a QUAD CPU Q6700 with a 2.66GHz CPU and 8GB of RAM. Our code and datasets are available on the world-wide web [25]. We made use of the following existing implementations.

Single Table Bayes Net Search GES search [26] with the BDeu score as implemented in version 4.3.9-0 of CMU's Tetrad package (structure prior uniform, ESS=10; [27]).

MLN Parameter Learning The default weight training procedure [28] of the Alchemy package [29], Version 30.

MLN Inference The MC-SAT inference algorithm [30] to compute a probability estimate for each possible value of a descriptive attribute for a given object or tuple of objects.

Algorithms. We compared three structure learning algorithms.

MBN An MLN structure is learned using the extended learn-and-join algorithm (Section 5). The weights of clauses are learned using Alchemy. This method is called MBN for "moralized Bayes Net" by Khosravi *et al.* [11].

LHL Lifted Hypergraph Learning [31] uses relational path finding to induce a more compact representation of data, in the form of a hypergraph over clusters of constants. Clauses represent associations among the clusters.

LSM Learning Structural Motifs [32] uses random walks to identify densely connected objects in data, and groups them and their associated relations into a motif.

We chose LSM and LHL because they are the most recent MLN structure learning methods that can potentially learn recursive dependencies.[3]

Performance Metrics. We use 3 performance metrics: Runtime, Accuracy (ACC), and Conditional log likelihood (CLL). Runtime includes structure learning and parameter learning time. ACC and CLL have been used in previous studies of MLN learning [34,31]. The CLL of a ground atom in a database given an MLN is its log-probability given the MLN and the information in the database. Accuracy is evaluated using the most likely value for a ground atom. For ACC and CLL the values we report are averages over all attribute predicates. We evaluate the learning methods using 5-fold cross-validation as follows. We formed 5 subdatabases for each by randomly selecting entities from each entity table and restricting the relationship tuples in each subdatabase to those that involve only the selected entities [11]. The models were trained on 4 of the 5 subdatabases, then tested on the remaining fold. We report the average over the 5 runs, one for each fold.

Synthetic Data. We manually created a small dataset (about 1000 tuples) for a University domain [13], including a Friendship self-relationship among students. The dataset features a strong autocorrelation for the gpa of friends and for the coffee habits of friends. Table 2 shows the results.

Real-World Data. We use the *Mondial Database.* This dataset contains data from multiple geographical web data sources [35]. We follow the modification of [36], and use a subset of the tables and features. Our dataset includes a self-relationship table *Borders* that relates two countries.

Results. Neither of the Markov Logic methods LHL nor LSM discovered any recursive dependencies. In contrast, the learn-and-join algorithm discovered the dependencies displayed in Table 4 using clausal notation. The dependency

$$religion(X) \leftarrow continent(X), Border(X, Y), religion(Y)$$

[3] The gradient boosting algorithm of Khot *et al* is even more recent, but is restricted to learn only non-recursive clauses [33].

University+	MBN	LSM	LHL
Time (seconds)	12	**1**	2941
Accuracy	**0.86**	0.44	0.47
CLL	**-0.89**	-2.21	-4.68

Table 2 Results on synthetic data.

Mondial	MBN	LSM	LHL
Time (seconds)	50	**2**	15323
Accuracy	**0.43**	0.26	0.26
CLL	**-1.39**	-1.43	-3.69

Table 3 Results on Mondial.

Database	Recursive Dependency Discovered
University	$gpa(X) \leftarrow ranking(X), grade(X,Y), registered(X,Y), Friend(X,Z), gpa(Z)$
University	$coffee(X) \leftarrow coffee(Y), Friend(X,Y)$
Mondial	$religion(X) \leftarrow continent(X), Border(X,Y), religion(Y)$
Mondial	$continent(X) \leftarrow Border(X,Y), continent(Y), gdp(X), religion(Y)$

Table 4 Dependencies discovered by the autocorrelation extension of the learn-and-join algorithm.

is a real-world example of the merging phenomenon discussed in Section 4.2. The learn-and-join algorithm analyzes the country table to find the dependency $religion(X) \leftarrow continent(X)$. Intuitively, the continent of a country predicts its religion. It then joins the Country table with the Borders relationship table to find the recursive dependency $religion(X) \leftarrow Border(X,Y), religion(Y)$. Intuitively, the religion of a country is correlated with the religion of its neighbors. As required by the Bayes net format, the two dependencies are merged to form a single set of parents $continent(X), Border(X,Y), religion(Y)$.

The predictive accuracy using MLN inference was much better in the moralized model (average accuracy improved by 25% or more). This indicates that the discovered recursive dependencies are important for improving predictions.

Both MBN and LSM are fast. The speed of LSM is due to the fact that its rules are mostly just the unit clauses that model marginal probabilities (e.g., $intelligence(S, 1)$).

Main Functor Constraint. Our last set of simulations examines the impact of the main functor constraint. A common way to learn recursive dependencies in multi-relational data mining is to duplicate the entity tables involved in a self-relationship as follows [37,3]. For instance for a self-relationship $Friend(U_1, U_2)$ with two foreign key pointers to an entity table *User*, we introduce a second entity table $User_{aux}$, which contains exactly the same information as the original *User* table. Then the *Friend* relation is rewritten as $Friend(U_1, U_{aux})$, where the second copy of the User table is treated as a different entity table from the original one. On the duplication approach, the Bayes net learning algorithm treats the variables U_1 and U_{aux} as separate variables, which we expect would lead to learning valid but redundant edges.

Figure 7 illustrates this phenomenon on the University dataset. The graph learned without the main functor constraint is much denser than the graph learned with the main functor constraint. Without the constraint, the learn-and-join algorithm learns 44 edges, whereas with the constraint, it learns 32 edges. Figure 7 shows various redundant edge pairs, for example an edge $Intelligence(S) \rightarrow ranking(S)$ and $Intelligence(S_{aux}) \rightarrow ranking(S_{aux})$.

Figure 8 shows a similar pattern for the Mondial data. The graph learned without the main functor constraint is much denser than the graph learned with

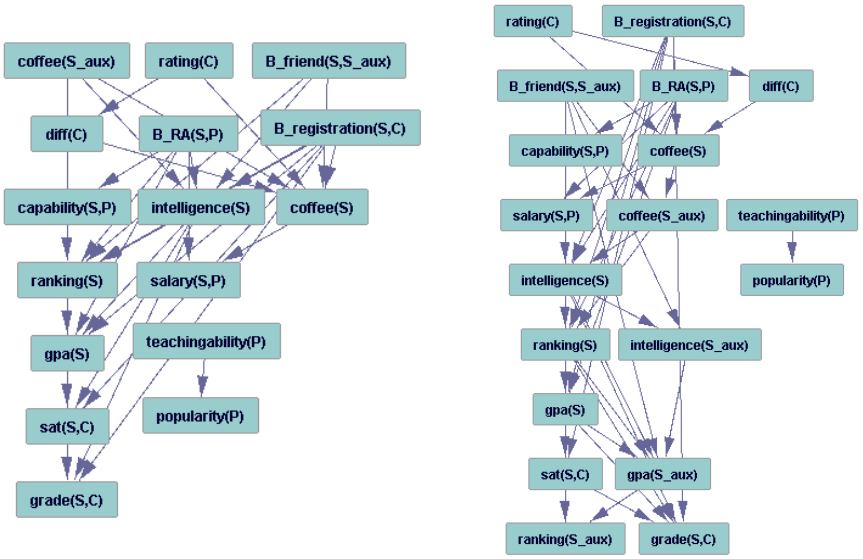

Fig. 7 Left: A parametrized Bayes net learned for the University database *with* the main functor constraint. This prevents auxiliary functor nodes, such as $ranking(S_{aux})$from having parents. As a result, some auxiliary functor nodes have no adjacencies at all and are not included in the graph. Right: A parametrized Bayes net learned for the University database *without* the main functor constraint. The resulting graph is much denser and contains duplicate edges.

the main functor constraint. Without the constraint, the learn-and-join algorithm learns 25 edges, whereas with the constraint, it learns 19 edges. Figure 8 shows various redundant edge pairs, for example an edge $govern(C) \rightarrow population(C)$ and $govern(C_{aux}) \rightarrow population(C_{aux})$.

We report the following quantitative measures of the differences.

SLtime(s) Structure learning time in seconds
Numrules Number of clauses in the Markov Logic Network excluding rules with weight 0.
AvgLength The average number of atoms per clause.
AvgAbWt The average absolute weight value.

Table 5 shows the results for University and the Mondial datasets. *Constraint* is the learn-and-join algorithm with the main functor constraint, whereas *Duplicate* is the learn-and-join algorithm applied naively to the duplicate tables without the constraint. As expected, the constraint speeds up structure learning, appreciably in the case of the larger Mondial dataset. The number of clauses is significantly less (50-60), while on average clauses are longer. The size of the weights indicates that the main functor constraint focuses the algorithm on the important rules. As expected from our theoretical analysis, the redundant edges do not improve predictive performance.

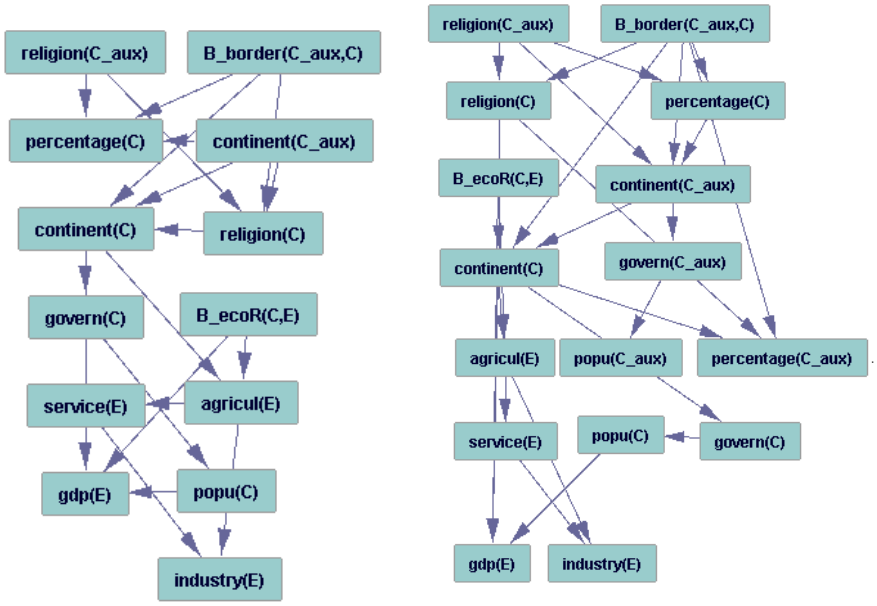

Fig. 8 Left: A parametrized Bayes net learned for the Mondial database *with* the main functor constraint. This prevents auxiliary functor nodesfrom having parents. As a result, some auxiliary functor nodes have no adjacencies at all and are not included in the graph. Right: A parametrized Bayes net learned for the University database *without* the main functor constraint. The resulting graph is much denser and contains duplicate edges.

University+	Constraint	Duplicate
SLtime(s)	**3.1**	3.2
# Rules	**289**	350
AvgLength	4.26	4.11
AvgAbWt	**2.08**	1.86
ACC	**0.86**	**0.86**
CLL	**-0.89**	**-0.89**

Mondial	Constraint	Duplicate
SLtime(s)	**8.9**	13.1
# Rules	**739**	798
AvgLength	3.98	3.8
AvgAbWt	0.22	**0.23**
ACC	**0.43**	**0.43**
CLL	**-1.39**	**-1.39**

Table 5 Comparison to study the effects of removing Main Functor Constraints. Left:University+ dataset. Right: Mondial dataset.

7 Conclusion and Future Work

An effective structure learning approach has been to upgrade propositional Bayes net learning for relational data. We presented a new method for applying Bayes net learning for *recursive dependencies* based on a recent pseudo-likelihood score and a new normal form theorem. The pseudo-likelihood score quantifies the fit of a recursive dependency model to relational data, and allows us to apply efficient model search algorithms. A new normal form eliminates potential redundancies that arise when predicates are duplicated to capture recursive relationships. In evaluations our structure learning method was very efficient and found recursive dependencies that were missed by structure learning methods for undirected models.

In our simulations, we considered recursive dependencies among attributes only. In future work, we aim to apply our results to learning recursive relationships among links (e.g., $\mathit{Friend}(X, Y)$ and $\mathit{Friend}(Y, Z)$ predicts $\mathit{Friend}(X, Z)$). Our theoretical results (Proposition 1 and Theorem 1) apply to link dependencies as well. However, as far as implementation goes, the current version of the learn-and-join algorithm is restricted to dependencies among attributes only.

8 Proofs

Proof Outline for Proposition 1. The result assumes that no functor node contains the same variable twice. This assumption does not involve a loss of modelling power because a functor node with a repeated variable can be rewritten using a new functor symbol (provided the functor node contains at least one variable). For instance, a functor node $\mathit{Friend}(X, X)$ can be replaced by the unary functor symbol $\mathit{Friend}_{self}(X)$.

($\Leftarrow$) If B is strictly stratified, then so is the ground graph $\overline{B}$, using the same level mapping. Since each child node is ranked less than its parent, there can be no cycle in $\overline{B}$.

($\Rightarrow$) Suppose that B is not strictly stratified. Then there are distinct fnodes $f(\boldsymbol{\tau}), f(\boldsymbol{\tau}')$ for the same functor such that $f(\boldsymbol{\tau})$ is an ancestor of $f(\boldsymbol{\tau}')$ in B. Since they are distinct fnodes, they disagree on at least one variable argument. Without loss of generality, let $f(\boldsymbol{\tau}) = f(X, \cdot)$ and $f(\boldsymbol{\tau}') = f(Y, \cdot)$, where $X \neq Y$. Pick any two distinct members a, b of the common population associated X, Y. First instantiate $f(X, \cdot)$ as a ground node $f(a, \cdot)$ and $f(Y, \cdot)$ as $f(b, \cdot)$. Then the ground graph $\overline{B}$ contains a directed path

$$f(a, \cdot) \rightarrow \cdots \rightarrow f(b, \cdot).$$

Second, instantiate $f(X, \cdot)$ as $f(b, \cdot)$ and $f(Y, \cdot)$ as $f(a, \cdot)$. Then the ground graph $\overline{B}$ contains a directed path

$$f(b, \cdot) \rightarrow \cdots \rightarrow f(a, \cdot).$$

Therefore the ground graph contains a directed cycle from $f(a, \cdot)$ to $f(b, \cdot)$ and back again, which establishes the claim.

Proof of Theorem 1. This result assumes that functor nodes do not contain constants, which is true in typical statistical-relational models. Let B be a stratified Bayes net. Consider the first function symbol f at level 0. Enumerate its associated functors as $f(\boldsymbol{\tau}_1), \ldots, f(\boldsymbol{\tau}_k)$, such that for every i, j, if $i < j$, then $f(\boldsymbol{\tau}_i)$ is not a descendant of $f(\boldsymbol{\tau}_j)$ in B. This is possible since B is acyclic. For instance, if functor f is unary, we can order the associated functor nodes as $f(X_1) < f(X_2) < \cdots$.

For every edge $g(\boldsymbol{\sigma}) \rightarrow f(\boldsymbol{\tau}_j)$, where $j < k$, change the variables in $\boldsymbol{\sigma}$ to obtain a term $\boldsymbol{\sigma}_j$ such that the edge $g(\boldsymbol{\sigma}) \rightarrow f(\boldsymbol{\tau}_j)$ has exactly the same instantiations as the edge $g(\boldsymbol{\sigma}_j) \rightarrow f(\boldsymbol{\tau}_k)$. This is possible because the functors contain neither constants nor repeated variables. For instance, we change an edge

$$g(X) \rightarrow f(X)$$

to get the edge

$$g(Y) \rightarrow f(Y).$$

Finally, add all edges of the form $g(\boldsymbol{\sigma}_j) \rightarrow f(\boldsymbol{\tau}_k)$ to B and eliminate all edges into $f(\boldsymbol{\tau}_j)$, for $j < k$. The resulting graph B_0 has the same ground graph as B. It is in main functor format wrt f since $f(\boldsymbol{\tau}_k)$ is the only functor with function symbol f that may have parents. To see that B_0 is acyclic, note that by stratification $f = g$, so all new edges are from functors $f(\boldsymbol{\tau}_j)$ to $f(\boldsymbol{\tau}_k)$. So a cycle in B_0 implies that $f(\boldsymbol{\tau}_k)$ is an ancestor of $f(\boldsymbol{\tau}_j)$ in B, for $j < k$, which is a contradiction.

We now repeat the construction for level 1, 2, etc. The resulting graphs $B_1, B_2, \ldots$ are acyclic because when an edge $g(\boldsymbol{\sigma}_j) \rightarrow f(\boldsymbol{\tau}_k)$ is added, either g is at a lower level than f, or $g = f$, therefore $g(\boldsymbol{\sigma}_j)$ is not an ancestor of $f(\boldsymbol{\tau}_k)$. After completing the construction for the highest stratum, we obtain a graph B' in main functor form whose grounding is the same as that of B, for any database.

References

1. Neville, J., Jensen, D.: Relational dependency networks. [19] chapter 8
2. Jensen, D., Neville, J.: Linkage and autocorrelation cause feature selection bias in relational learning. In: ICML. (2002)
3. Chen, H., Liu, H., Han, J., Yin, X.: Exploring optimization of semantic relationship graph for multi-relational Bayesian classification. Decision Support Systems **48:1** (July 2009) 112–121
4. Poole, D.: First-order probabilistic inference. In: IJCAI. (2003) 985–991
5. Getoor, L.G., Friedman, N., Taskar, B.: Learning probabilistic models of relational structure. In: ICML, Morgan Kaufmann (2001) 170–177
6. Kersting, K., de Raedt, L.: Bayesian logic programming: Theory and tool. [19] chapter 10 291–318
7. Ramon, J., Croonenborghs, T., Fierens, D., Blockeel, H., Bruynooghe, M.: Generalized ordering-search for learning directed probabilistic logical models. Machine Learning **70**(2-3) (2008) 169–188
8. Domingos, P., Richardson, M.: Markov logic: A unifying framework for statistical relational learning. [19]
9. Taskar, B., Abbeel, P., Koller, D.: Discriminative probabilistic models for relational data. In: UAI. (2002) 485–492
10. Schulte, O.: A tractable pseudo-likelihood function for Bayes nets applied to relational data. In: SIAM SDM. (2011) 462–473
11. Khosravi, H., Schulte, O., Man, T., Xu, X., Bina, B.: Structure learning for Markov logic networks with many descriptive attributes. In: AAAI. (2010) 487–493
12. Fierens, D., Ramon, J., Bruynooghe, M., Blockeel, H.: Learning directed probabilistic logical models: Ordering-search versus structure-search. In: ECML. (2007) 567–574
13. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: In IJCAI, Springer-Verlag (1999) 1300–1309
14. Apt, K.R., Bezem, M.: Acyclic programs. New Generation Comput. **9**(3/4) (1991) 335–364
15. Fierens, D.: On the relationship between logical bayesian networks and probabilistic logic programming based on the distribution semantics. In: ILP. (2009) 17–24
16. Getoor, L., Grant, J.: Prl: A probabilistic relational language. Machine Learning **62**(1-2) (2006) 7–31
17. Klug, A.C.: Equivalence of relational algebra and relational calculus query languages having aggregate functions. J. ACM **29**(3) (1982) 699–717
18. Ullman, J.D.: Principles of database systems. 2. Computer Science Press (1982)
19. Getoor, L., Tasker, B.: Introduction to statistical relational learning. MIT Press (2007)
20. Ngo, L., Haddawy, P.: Answering queries from context-sensitive probabilistic knowledge bases. Theor. Comput. Sci. **171**(1-2) (1997) 147–177
21. Koller, D., Pfeffer, A.: Learning probabilities for noisy first-order rules. In: IJCAI. (1997) 1316–1323

22. Wellman, M., Breese, J., Goldman, R.: From knowledge bases to decision models. Knowledge Engineering Review **7** (1992) 35–53
23. Heckerman, D., Meek, C., Koller, D.: Probabilistic entity-relationship models, PRMs, and plate models. [19]
24. Lifschitz, V.: Foundations of logic programming. Principles of Knowledge Representation, CSLI Publications (1996)
25. Khosravi, H., Man, T., Hu, J., Gao, E., Schulte, O.: Learn and join algorithm code. URL = `http://www.cs.sfu.ca/~oschulte/jbn/`.
26. Chickering, D.: Optimal structure identification with greedy search. Journal of Machine Learning Research **3** (2003) 507–554
27. The Tetrad Group: The Tetrad project (2008) http://www.phil.cmu.edu/projects/tetrad/.
28. Lowd, D., Domingos, P.: Efficient weight learning for Markov logic networks. In: PKDD. (2007) 200–211
29. Kok, S., Summer, M., Richardson, M., Singla, P., Poon, H., Lowd, D., Wang, J., Domingos, P.: The Alchemy system for statistical relational AI. Technical report, University of Washington. (2009) Version 30.
30. Poon, H., Domingos, P.: Sound and efficient inference with probabilistic and deterministic dependencies. In: AAAI. (2006)
31. Kok, S., Domingos, P.: Learning markov logic network structure via hypergraph lifting. In: ICML. (2009) 64–71
32. Kok, S., Domingos, P.: Learning Markov logic networks using structural motifs. In: ICML. (2010) 551–558
33. Khot, T., Natarajan, S., Kersting, K., Shavlik, J.W.: Learning markov logic networks via functional gradient boosting. In: ICDM. (2011) 320–329
34. Mihalkova, L., Mooney, R.J.: Bottom-up learning of Markov logic network structure. In: ICML, ACM (2007) 625–632
35. May, W.: Information extraction and integration: The mondial case study. Technical report, Universität Freiburg, Institut für Informatik (1999)
36. She, R., Wang, K., Xu, Y.: Pushing feature selection ahead of join. In: SIAM SDM. (2005)
37. Yin, X., Han, J., Yang, J., Yu, P.S.: Crossmine: Efficient classification across multiple database relations. In: Constraint-Based Mining and Inductive Databases. (2004) 172–195