Learning Directed Probabilistic Logical Models using Ordering-search

Daan Fierens, Jan Ramon, Maurice Bruynooghe, and Hendrik Blockeel

K.U.Leuven, Dept. of Computer Science, Celestijnenlaan 200A, 3001 Heverlee, Belgium,
{Daan.Fierens, Jan.Ramon, Maurice.Bruynooghe,
Hendrik.Blockeel}@cs.kuleuven.be

Abstract. There is an increasing interest in upgrading Bayesian networks to the relational case, resulting in so-called directed probabilistic logical models. In this paper we discuss how to learn non-recursive directed probabilistic logical models from relational data. This problem has already been tackled before by upgrading the structure-search algorithm for learning Bayesian networks. In this paper we propose to upgrade another algorithm, namely ordering-search, since for Bayesian networks this was found to work better than structure-search. We have implemented both algorithms for the formalism Logical Bayesian Networks and are currently working on an experimental comparison of both algorithms.

Keywords: probabilistic logical models, structure learning, Bayesian networks, probability trees

1 Introduction

There is an increasing interest in probabilistic logical models as can be seen from the variety of formalisms that have recently been introduced for describing such models. Many of these formalisms deal with directed models that are upgrades of Bayesian networks to the relational case. Learning algorithms have been developed for several such formalisms [3, 5, 6]. Most of these algorithms are essentially upgrades of the *structure-search* algorithm for Bayesian networks. Recently, Teyssier and Koller [9] introduced an alternative algorithm for learning Bayesian networks, *ordering-search*, and found this to perform at least as well as structure-search while usually being faster. This motivates us to investigate how ordering-search can be upgraded to the relational case. More precisely, we show how to use ordering-search for learning non-recursive directed probabilistic logical models. In this paper we use the formalism Logical Bayesian networks (LBNs) but the proposed approach is also valid for related formalisms such as Probabilistic Relational Models, Bayesian Logic Programs and Relational Bayesian Networks [1]. This paper describes work in progress. Our current work is to experimentally validate our approach and compare it to structure-search.

We now first discuss Bayesian networks and LBNs. Then we discuss the problem of learning non-recursive LBNs from data and how to solve this by upgrading the ordering-search algorithm. Finally we discuss related work and some directions for future work.

2 Bayesian Networks

A Bayesian network is a compact specification of a joint probability distribution on a set of random variables under the form of a directed acyclic graph (the 'structure') and a set of conditional probability distributions (CPDs). When learning from data the goal is usually to find the structure and CPDs that maximize a certain scoring criterion (such as likelihood or minimum description length). There exist several approaches for learning Bayesian networks. We now review two of the most important approaches.

The most traditional approach is *structure-search* [4], which is basically hill climbing through the space of possible structures. Starting from an initial structure, a number of possible refinements of this structure are evaluated and the refinement that yields the structure with the highest score is chosen. A refinement of a structure is usually obtained by adding, deleting or reversing an edge (of course only acyclic structures are allowed). The above process is then repeated until convergence.

An alternative approach called *ordering-search* is based on the observation that it is relatively easy to learn a Bayesian network if an ordering on the set of random variables is given [9]. Such an ordering eliminates the possibility of cycles and makes it possible to decide for each variable X separately which variables, from all variables preceding it in the ordering, are its parents. This can simply be done by learning a CPD for X under the assumption that 'selective' CPDs are used, i.e. CPDs that select from all candidate inputs the relevant inputs (for instance conditional probability tables with a bound on the number of effective inputs). Since this determines both the parents and the CPD of X, applying this procedure to every variable yields a complete specification of a Bayesian network. However, the score of this Bayesian network depends heavily on the quality of the ordering that is used. Often the optimal ordering is not known in advance. Hence, the idea of ordering-search is to perform hill-climbing through the space of possible orderings, in each step applying the above procedure.

Teyssier and Koller [9] experimentally compared structure-search and ordering-search and found that ordering-search is always at least as good as structure-search and usually faster. As an explanation of these results, Teyssier and Koller note that the search space of orderings is smaller than the search space of structures and that ordering-search does not need acyclicity tests, which are costly if there are many variables.

3 Logical Bayesian Networks

In this section we briefly discuss the formalism Logical Bayesian Networks [1].

A Logical Bayesian Network or LBN is essentially a specification of a Bayesian network conditioned on some logical input predicates describing the domain of discourse. For instance, when modelling the traditional 'university' example [3], we would use predicates $student/1$, $course/1$, $prof/1$, $teaches/2$ and $takes/2$ with their obvious meanings. The semantics of an LBN is that, given an interpretation of these logical predicates, the LBN induces a particular Bayesian network.

In LBNs random variables are represented as ground atoms built from certain special predicates, the *probabilistic predicates*. For instance, if $intelligence/1$ is a probabilistic predicate then the atom $intelligence(ann)$ is called a probabilistic atom and

represents a random variable. Apart from sets of logical and probabilistic predicates an LBN basically consists of three parts: a set of random variable declarations, a set of conditional dependency clauses and a set of logical CPDs. We now explain this further.

For the university example the random variable declarations are the following.

```
random(intelligence(S)) <- student(S).
random(ranking(S)) <- student(S).
random(difficulty(C)) <- course(C).
random(rating(C)) <- course(C).
random(ability(P)) <- prof(P).
random(popularity(P)) <- prof(P).
random(grade(S,C)) <- takes(S,C).
random(satisfaction(S,C)) <- takes(S,C).
```

In these clauses $random/1$ is a special predicate. Informally, the first clause, for instance, should be read as "$intelligence(S)$ is a random variable if S is a student". Formally, in the Bayesian network for a particular interpretation of the logical predicates, there is a node for each ground probabilistic atom p for which $random(p)$ holds.

The conditional dependency clauses for the university example are the following.

```
grade(S,C) | intelligence(S), difficulty(C).
ranking(S) | grade(S,C).
satisfaction(S,C) | grade(S,C), ability(P) <- teaches(P,C).
rating(C) | satisfaction(S,C).
popularity(P) | rating(C) <- teaches(P,C).
```

Informally, the first clause should be read as "the grade of a student S for a course C depends on the intelligence of S and the difficulty of C". There are also more complex clauses such as the last clause that should be read as "the popularity of a professor P depends on the rating of a course C if P teaches C". In this clause, $popularity(P)$ is called the head, $rating(C)$ the body and $teaches(P, C)$ the context. Formally, in the induced Bayesian network there is an edge from a node p_{parent} to a node p_{child} if there is a ground instance of a dependency clause with p_{child} in the head and p_{parent} in the body, with true context and with $random(p)$ true for each probabilistic atom p.

To quantify the dependencies specified by the conditional dependency clauses, LBNs use so-called logical CPDs. In this paper we represent logical CPDs under the form of logical probability trees in TILDE [2], see Section 5.2 for an example. These logical CPDs can be used to determine the CPDs in the induced Bayesian network.

The *predicate dependency graph* of an LBN is the graph that contains a node for each probabilistic predicate and an edge from a node p_1 to a node p_2 if the LBN contains a conditional dependency clause with predicate p_2 in the head and p_1 in the body. An LBN is called *non-recursive* if its predicate dependency graph is acyclic and *recursive* otherwise. Note that for non-recursive LBNs the induced Bayesian network (for any interpretation of the logical predicates) is always acyclic.

4 Learning Logical Bayesian Networks: Problem Description

When learning LBNs, the random variable declarations are usually known. The conditional dependency clauses and the logical CPDs can then be learned from a dataset of examples, where each example consists of two parts: an interpretation of the logical predicates and an assignment of values to all ground random variables (as determined by the random variable declarations). The goal of learning is to find the clauses and logical CPDs that maximize the scoring criterion. In this paper we focus on learning *non-recursive* LBNs. We briefly discuss learning recursive LBNs in Section 6.

Some LBNs have syntactic variants. This is important since syntactic variants could cause redundancy in the learning algorithm. In the case of LBNs, the main cause of the existence of variants is that multiple conditional dependency clauses with the same head are allowed. A consequence of this is that each conditional dependency clause with a particular head and with multiple probabilistic atoms in the body can be translated into an equivalent set of conditional dependency clauses each with the same head but with only one probabilistic atom in the body (the equivalence is with respect to a particular set of random variable declarations). Hence, when learning we will only consider LBNs with *conditional dependency clauses with only one probabilistic atom in the body*. We now explain this in more detail.

For each LBN with conditional dependency clauses with multiple probabilistic atoms in the body, an equivalent LBN can be obtained by replacing each such clause C by a set of equivalent clauses. Concretely, for each probabilistic atom in the body of C we create a new clause with the same head as C, with only that atom in the body and with as context the context of C plus the condition that all other probabilistic atoms in the body of C are random variables. As an illustration, consider the following LBN.

```
random(ranking(S)) <- student(S).
random(intelligence(S)) <- student(S).
random(thesis_score(S)) <- student(S), in_master(S).
ranking(S) | intelligence(S), thesis_score(S).
```

Note that in this LBN ranking and intelligence are defined for students but thesis-score is defined only for particular students, namely master students. Hence the dependency of ranking on intelligence and thesis-score only holds for master students. The conditional dependency clause can be transformed into the following set of equivalent clauses.

```
ranking(S) | intelligence(S) <- random(thesis_score(S)).
ranking(S) | thesis_score(S) <- random(intelligence(S)).
```

By applying the random variable declarations, these clauses can be rewritten as follows.

```
ranking(S) | intelligence(S) <- student(S), in_master(S).
ranking(S) | thesis_score(S) <- student(S).
```

These clauses can be further simplified by noting that the condition that S is a student is redundant since ranking is only defined for students and hence the clauses will only be used with S being a student. We finally obtain the following clauses.

```
ranking(S) | intelligence(S) <- in_master(S).
ranking(S) | thesis_score(S).
```

Note that the second clause does not specify in the context that S should be a master student. This is indeed not needed since thesis-score is only defined for master students and hence the clause only applies to master students anyway.

5 Ordering-search for Non-recursive Logical Bayesian Networks

The *structure-search* algorithm for Bayesian networks has already been upgraded to the relational case for several formalisms [3, 5, 6]. Some differences between the relational case and the case of Bayesian networks are that other refinement operators for the structure are needed (for LBNs these could be adding a new conditional dependency clause, deleting an existing clause and swapping the head and the body of an existing clause), more complex CPDs are needed (such as combining rules [5, 6] or aggregates [3]) and the scoring criteria have to be adapted (such as the Bayesian Information Criterion [6]).

Since for Bayesian networks *ordering-search* performs at least as well as structure-search it is interesting to investigate how ordering-search can be upgraded to the relational case. In this section we show how to do this for non-recursive LBNs.

5.1 Algorithm Overview

The ordering-search algorithm for learning non-recursive LBNs basically corresponds to hillclimbing through the space of possible orderings on the set of probabilistic predicates. This algorithm is shown on a high-level in Figure 1. In this algorithm the score of an ordering is defined as the score of the LBN that is learned for that ordering. The neighbourhood of an ordering $O_{current}$ is defined as the set of all orderings that can be obtained by swapping a pair of adjacent predicates in $O_{current}$ (this is similar to what happens for Bayesian networks [9]). As this algorithm only converges to a local optimum, in practice we run this algorithm several times each time with a different initial random ordering and we retain the best result.

As for Bayesian networks, this algorithm can be implemented in an efficient way when the scoring criterion is 'decomposable', as is the case for all criteria we consider. For a decomposable scoring criterion the score of an ordering is the sum of the scores of all logical CPDs. Hence $\Delta score(O_{cand})$ only depends on the score of the logical CPDs that are different for O_{cand} than for $O_{current}$. In our algorithm there are typically only two such logical CPDs (since the orderings in the neighbourhood of $O_{current}$ are obtained by swapping two adjacent predicates, which only changes the logical CPDs for these two predicates). Hence computing $\Delta score(O_{cand})$ only requires learning two new logical CPDs. Moreover, many of the score-changes $\Delta score(O_{cand})$ that are computed during one iteration of the repeat loop are still valid and can be reused without extra computations during the next iterations of the loop.

Apart from the fact that we consider orderings on the set of probabilistic predicates instead of on the set of random variables, there are two more differences between the algorithm for LBNs and the algorithm for Bayesian networks. The first difference is that

```
% find a good ordering:
O_current = random ordering on the set of probabilistic predicates
repeat until convergence
    for each O_cand ∈ neighbourhood(O_current)
        compute Δscore(O_cand) = score(O_cand) − score(O_current)
    end for
    O_current = argmax(Δscore(O_cand))
end repeat
% for the final ordering, extract the conditional dependency clauses:
for each probabilistic predicate p
    extract clauses from logical CPD of p for O_current
end for
```

Fig. 1. The ordering-search algorithm for non-recursive LBNs.

in the case of LBNs we use logical CPDs represented as logical probability trees instead of simple propositional CPDs. The second difference is that in the case of LBNs, once we found the optimal ordering and the logical CPDs for this ordering, we need an extra step to extract the conditional dependency clauses from these logical CPDs. In the next sections we discuss these two issues in more detail.

5.2 Learning Logical CPDs

An LBN needs one logical CPD for each probabilistic predicate. A logical CPD quantifies the dependencies in an LBN (like combining rules do in some other formalisms [5, 6]). We represent logical CPDs as *logical probability trees* in TILDE [2]. The internal nodes in the tree for a probabilistic atom p_{target} can contain a) tests on the values of probabilistic atoms that are parents of p_{target} according to the conditional dependency clauses, b) conjunctions of logical literals, and c) combinations of the two. Leaves contain probability distributions on the values of p_{target}. An example of a tree for $satisfaction(S, C)$ is shown in Figure 2.

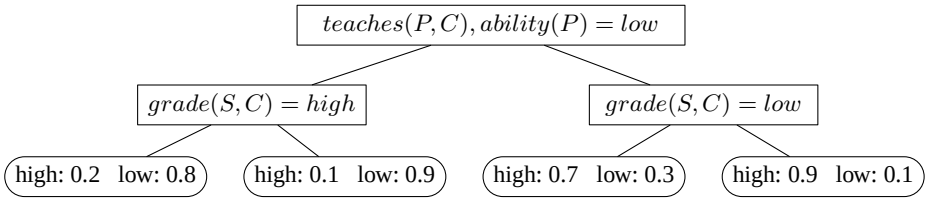

Fig. 2. Example of a logical CPD for $satisfaction(S, C)$. When the test in an internal node succeeds the left branch is taken, when it fails the right branch is taken.

Logical probability trees like the one in Figure 2 can be learned using the standard probability tree algorithms in TILDE [1][2]. We currently consider two ways of learning and scoring trees. We can learn trees using the BIC criterion which naturally leads to using BIC as a scoring criterion. Alternatively, because BIC in some cases performs quite poorly, we could learn trees using the randomization approach of Fierens et al. [2] and simply use likelihood as a scoring criterion. Likelihood (of the training data) is not a good criterion when unselective CPDs are used (since it would give a fully connected model, leading to overfitting). However with selective CPDs, such as probability trees, using likelihood might work provided that the CPDs are selective enough to prevent overfitting.

5.3 Extracting the Conditional Dependency Clauses from the Logical CPDs

When we have found a good ordering and the logical CPDs for this ordering, we still have to extract a set of conditional dependency clauses from these logical CPDs in order to obtain the learned LBN. Below we explain how to extract the clauses from a logical probability tree. To obtain a full LBN, this procedure has to be applied to the probability tree for each probabilistic predicate.

When extracting conditional dependency clauses from a logical probability tree with as target the probabilistic atom p_{target}, we want to find a set of clauses that is *consistent* with the tree. With this we mean that the tree should never test any probabilistic atom that is not a parent of p_{target} according to the set of clauses. Moreover, we want to find the most specific set of clauses that is consistent with the tree.

When extracting clauses from a tree, we create a clause for each test on a probabilistic atom in each internal node of the tree. Call the atom that is tested p_{test} and the node N. In the most general case, apart from the test on p_{test}, the node N can contain a number of tests on other probabilistic atoms and a conjunction of logical literals. Call this conjunction l. We then create a clause of the form $p_{target} \mid p_{test} \leftarrow l, path(N)$, where $path(N)$ is a conjunction of logical literals that describes the path from the root to N. Each node on this path can contribute a number of logical literals to $path(N)$. A succeeded node (i.e. a node for which the succeeding branch of the tree was chosen in the path) contributes the conjunction of all logical literals that it contains. A failed node that does not contain any tests on probabilistic atoms contributes the negation of the conjunction of all logical literals that it contains. A failed node that contains a test on a probabilistic atom does not contribute to the path[2].

As an example, consider the probability tree in Figure 2. For this tree, p_{target} is $satisfaction(S, C)$. For the root node, p_{test} is $ability(P)$, l is $teaches(P, C)$ and the path is empty. For the internal node below the root to the left, p_{test} is $grade(S, C)$, l is empty and the path is $teaches(P, C)$. For the node below the root to the right, p_{test} is $grade(S, C)$ and l and the path are both empty. The three resulting clauses for these nodes are respectively the following.

```
satisfaction(S,C) | ability(P) <- teaches(P,C).
```

[1] Internally in TILDE, tests like $grade(S, C) = low$ are represented as $grade(S, C, low)$.

[2] Letting it contribute the negation of its logical literals could be inconsistent since we cannot be sure that it were the logical literals that caused the failure and not the probabilistic tests.

```
satisfaction(S,C) | grade(S,C) <- teaches(P,C).
satisfaction(S,C) | grade(S,C).
```

The second clause is redundant (it is a special case of the third) and can be dropped.

6 Relation to Generalized Ordering-search

Recently we upgraded ordering-search for learning *recursive* LBNs [7]. We called the resulting algorithm generalized ordering-search. The main idea behind this algorithm is that to model recursive dependencies one has to consider orderings on the set of ground random variables. This is more complex than considering orderings on the set of probabilistic predicates as we propose in this paper. The reason for this is that the set of predicates is fixed whereas the set of ground random variables depends on the domain of discourse (i.e. the interpretation of the logical predicates). Hence, in generalized ordering-search we cannot simply learn a fixed ordering (as we do in this work for the set of predicates) but have to learn a model of the ordering as a function of the logical predicates.

In principle, generalized ordering-search can also be used to learn *non-recursive* LBNs. However, in this respect generalized ordering-search has a number of disadvantages as compared to the algorithm proposed in this paper. One disadvantage is that it does not learn an LBN 'in closed form' in the sense that it does not learn a set of conditional dependency clauses but rather gives a procedure for determining the induced Bayesian network given any possible interpretation of the logical predicates. Another disadvantage is that it deviates quite far from the propositional ordering-search algorithm. For instance, when applied on propositional data generalized-ordering search does not correspond to the original propositional ordering-search algorithm, while this is the case for our algorithm. This might make generalized-ordering search harder to understand for people familiar with the propositional ordering-search algorithm.

7 Conclusion and Future Work

We showed how to learn non-recursive directed probabilistic logical models by upgrading the ordering-search algorithm for Bayesian networks. We have implemented this algorithm and the structure-search algorithm for LBNs and are currently working on an experimental comparison of both algorithms in terms of accuracy and compactness of the learned model in function of running time. We are currently using two datasets, an artificially generated dataset for the university domain discussed before in this paper and the UWCSE dataset [8], but we are interested in obtaining other relational datasets for which learning non-recursive LBNs is interesting.

Acknowledgements

Daan Fierens is supported by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT Vlaanderen). Jan Ramon and Hendrik Blockeel are post-doctoral fellows of the Fund for Scientific Research (FWO) of Flanders.

References

1. D. Fierens, H. Blockeel, M. Bruynooghe, and J. Ramon. Logical Bayesian networks and their relation to other probabilistic logical models. In *Proceedings of the 15th International Conference on Inductive Logic Programming*, volume 3625 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2005.
2. D. Fierens, J. Ramon, H. Blockeel, and M. Bruynooghe. A comparison of pruning criteria for learning trees. Technical Report CW 488, Department of Computer Science, Katholieke Universiteit Leuven, April 2007. http://www.cs.kuleuven.be/∼daanf/CW488.pdf
3. L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning Probabilistic Relational Models. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, pages 307–334. Springer-Verlag, 2001.
4. D. Heckerman, D. Geiger, and D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.
5. K. Kersting and L. De Raedt. Towards combining inductive logic programming and Bayesian networks. In *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Computer Science*, pages 118–131. Springer-Verlag, 2001.
6. S. Natarajan, W. Wong, and P. Tadepalli. Structure refi nement in First Order Conditional Influence Language. In *Proceedings of the ICML workshop on Open Problems in Statistical Relational Learning*, 2006.
7. J. Ramon, T. Croonenborghs, D. Fierens, H. Blockeel, and M. Bruynooghe. Generalized ordering-search for learning directed probabilistic logical models. 2007. Submitted to Machine Learning, special issue on Inductive Logic Programming.
8. M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1–2):107–136, 2006.
9. M. Teyssier and D. Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *Proceedings of the 21st conference on Uncertainty in AI (UAI)*, pages 584–590. AUAI Press, 2005.