Structured Set Variable Domains in Bayesian Network Structure Learning using Constraint Programming

Fulya Trösser, [1] Simon de Givry, [1] George Katsirelos [2]
[1] Université Fédérale de Toulouse, ANITI, INRAE, UR 875, 31326 Toulouse, France
[2] MIA Paris-Saclay, INRAE, AgroParisTech, France
fulya.trosser@hotmail.com, simon.degivry@inrae.fr, gkatsi@gmail.com.com

Abstract

Constraint programming is a state of the art technique for learning the structure of Bayesian Networks from data (Bayesian Network Structure Learning – BNSL). However, scalability both for CP and other combinatorial optimization techniques for this problem is limited by the fact that the basic decision variables are set variables with domain sizes that may grow super polynomially with the number of random variables. Usual techniques for handling set variables in CP are not useful, as they lead to poor bounds. In this paper, we propose using decision trees as a data structure for storing sets of sets to represent set variable domains. We show that relatively simple operations are sufficient to implement all propagation and bounding algorithms, and that the use of these data structures improves scalability of a state of the art CP-based solver for BNSL.

Introduction

Bayesian Networks (BNs) are directed probabilistic graphical models, which can describe a normalized joint probability distribution over a potentially large set of random variables, by exploiting conditional independence to decompose the function. Learning the structure of BNs from data (the Bayesian Network Structure Learning problem, BNSL) is a challenging combinatorial optimization problem. There exist constraint-based approaches to learn BNs, which use local conditional independence tests, and score-based approaches, which use a decomposable score function to score each potential structure and aim to find the structure that minimizes this score. The former are known to be efficient, but have trouble with noisy data. The latter yield a known to be NP-hard problem (Chickering 1995), which additionally has proved very challenging in practice.

There exist complete methods for score-based BNSL based on dynamic programming (Silander and Myllymäki 2006), heuristic search (Yuan and Malone 2013; Fan and Yuan 2015), maximum satisfiability (Berg, Järvisalo, and Malone 2014), branch-and-cut (Bartlett and Cussens 2017) and constraint programming (van Beek and Hoffmann 2015; Trösser, de Givry, and Katsirelos 2021). Branch-and-cut and constraint programming have proven to be the most successful of these methods. However, scaling them up remains challenging. One challenge has to do with the decomposition of the scoring functions: these assign a score to each potential set of parents of each vertex and the score of a specific structure is the sum of the scores of each parent set. This means that the objective function must have a term for each potential parent set, a potentially exponential number of terms. There are various methods by which this number is made manageable, but it is still among the greatest obstacles to scalability. Moreover, the best solvers, ILP-based GOBNILP (Bartlett and Cussens 2017), and CP-based ELSA (Trösser, de Givry, and Katsirelos 2021) also explicitly have this set of parent sets in other parts of the model as well, in the case of ELSA as domains of variables.

Here, we propose exploiting the fact that these domains are structured, i.e., that each value is a set. Specifically, we show that we can represent potential parent sets as paths on decision trees and that using these decision trees we can answer queries more efficiently than by traversing a list of domain values. This feature has not been exploited in BNSL in the past and allows us to solve large instances more efficiently.

CP-based BNSL

Learning a BN from a set of multivariate discrete data using the score based method uses a decomposable scoring function (such as BIC (Schwarz 1978; Lam and Bacchus 1994) or BDeu (Buntine 1991; Heckerman, Geiger, and Chickering 1995)) which assigns, based on the data, a score to each potential parent set of each vertex. The BNSL problem is the problem of finding the structure G which minimizes this scoring function.

The number of candidate parent sets can in principle be exponentially large, but it is typically kept in check. For one, the BIC scoring function (Schwarz 1978; Lam and Bacchus 1994) guarantees that the number of candidate parent sets grows only logarithmically with the size of the data set. Second, there exist dedicated pruning rules (de Campos and Ji 2010; de Campos et al. 2018) which reduce the set further. As a last resort, an upper bound can be placed on the cardinality of parent sets. This is necessary especially in larger instances, where it is necessary to limit cardinality to as low as 3 in some cases.

ELSA (Trösser, de Givry, and Katsirelos 2021) is a CP-based solver for the BNSL, based on the CPBayes solver (van Beek and Hoffmann 2015). In the model it uses, for each random variable X, there exists a corresponding CSP variable P_X whose domain is the set of candidate parent sets of X. There exists an acyclicity constraint over these which requires that their instantiation yields an acyclic graph. Additionally, ELSA computes lower bounds by approximately solving a linear relaxation of an ILP which was proposed by Bartlett and Cussens (Bartlett and Cussens 2017) for the GOBNILP solver. That ILP is exponentially large, as it contains an exponential number of so-called *cluster constraints*. Hence, following GOBNILP, ELSA starts with none of the cluster constraints in the linear relaxation and then adds only some of those that can improve the dual bound. Finding those is NP-hard, so ELSA uses a polynomial time algorithm which can identify a strict subset of all improving cluster constraints.

In both the acyclicity constraint and in discovering improving cluster cuts, the most expensive queries performed on domains are, respectively:

1. Does there exist a domain value S such that $S \subseteq T$ for some T?
2. Does there exist a domain value S with reduced cost 0 such that $S \subseteq T$ for some T?

Decision Trees as Domain Store

The set of sets that are in a domain can be seen as the set of solutions of a propositional formula, in which we have a propositional variable for each element of the universe. Therefore, knowledge compilation languages can be used to represent a domain.

We use decision trees here, in particular binary decision trees with implied literals, inspired by a similar technique in BDDs (Lai, Liu, and Wang 2013). The main use of decision trees is in machine learning for classification, but their use as a data structure for representing sets of sets (or, equivalently, a knowledge compilation language) is straightforward.

To set this in the more familiar use of decision trees for classification, observe that we can set the features to be the variables of the indicator function of the sets in the domain and the classes as $in\text{-}set$ and $not\text{-}in\text{-}set$.

One difference is that, in machine learning, the objective is not only to construct models that perform well on the training set, but that also generalize. Hence, it is not only acceptable, but also desirable to misclassify some samples in training sets, if that means a smaller and hence more general decision tree. In our setting, however, where we use decision trees to model a Boolean function, we accept no error. So no two sets that belong to different classes, i.e., one in $in\text{-}set$ and one in $not\text{-}in\text{-}set$, are allowed to both be consistent with the same leaf node.

Subset queries To answer the queries given above, we perform a depth first (DFS) traversal of the tree. At each node n, we check the feature associated, which is a variable in our case. If that is not in T, we only allow DFS to follow the outgoing arc labeled with false. Otherwise, we allow DFS to follow both outgoing arcs. If we reach a leaf labeled with class $in\text{-}set$, we stop and report success. If we exhaust the search without reaching a leaf, we report failure.

This procedure can be used to answer subset queries of both types. For type 1, we mask away sets that have been removed. For queries of the second type, we mask away those sets whose reduced cost is greater than 0.

Summary of Experimental Results

We implemented decision trees as the domain representation on top of ELSA and denote this solver ELSA^{IG}. We compared it against the previous version of ELSA GOBNILP and CPBayes.

We observed that the use of decision trees has little effect, either positive or negative, for the smaller instances, but it makes a great difference in the larger instances. In particular, ELSA^{IG} is the only solver that can prove optimality for several datasets. For some datasets where ELSA was significantly worse than CPBayes, ELSA^{IG} either closes the gap back down or is faster than CPBayes. However, ELSA^{IG} regresses with respect to ELSA in some datasets where the benefit of the decision trees in terms of the reduction of the cost in answering the subset queries is comparatively reduced, so it is not enough to overcome other overheads.

With respect to GOBNILP, ELSA^{IG} mostly outperforms it, but there are some instances where neither ELSA nor ELSA^{IG} can match it. It seems, however, that ELSA^{IG} is overall the best performer.

Despite these improvements, answering subset queries is still the most time consuming operation performed by the solver. Moreover, the fact remains that decision trees as a knowledge compilation language are fairly weak in terms of conciseness. It remains an open question whether we can overcome the issues with ROBDDs or even DNNFs to achieve even more significant speedups.

References

Bartlett, M.; and Cussens, J. 2017. Integer Linear Programming for the Bayesian network structure learning problem. *Artificial Intelligence*, 258–271.

Berg, J.; Järvisalo, M.; and Malone, B. 2014. Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In *Artificial Intelligence and Statistics*, 86–95. PMLR.

Buntine, W. 1991. Theory refinement on Bayesian networks. In *Proc. of UAI*, 52–60. Elsevier.

Chickering, D. M. 1995. Learning Bayesian Networks is NP-Complete. In *Proc. of Fifth Int. Workshop on Artificial Intelligence and Statistics (AISTATS)*, 121–130. Key West, Florida, USA.

de Campos, C. P.; and Ji, Q. 2010. Properties of Bayesian Dirichlet Scores to Learn Bayesian Network Structures. In *Proc. of AAAI-10*. Atlanta, Georgia, USA.

de Campos, C. P.; Scanagatta, M.; Corani, G.; and Zaffalon, M. 2018. Entropy-based pruning for learning Bayesian networks using BIC. *Artificial Intelligence*, 260: 42–50.

Fan, X.; and Yuan, C. 2015. An Improved Lower Bound for Bayesian Network Structure Learning. In *Proc. of AAAI-15*. Austin, Texas.

Heckerman, D.; Geiger, D.; and Chickering, D. M. 1995. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3): 197–243.

Lai, Y.; Liu, D.; and Wang, S. 2013. Reduced ordered binary decision diagram with implied literals: A new knowledge compilation approach. *Knowledge and Information Systems*, 35(3): 665–712.

Lam, W.; and Bacchus, F. 1994. Using New Data to Refine a Bayesian Network. In *Proc. of UAI*, 383–390.

Schwarz, G. 1978. Estimating the dimension of a model. *The Annals of Statistics*, 6(2): 461–464.

Silander, T.; and Myllymäki, P. 2006. A Simple Approach for Finding the Globally Optimal Bayesian Network Structure. In *Proc. of UAI'06*. Cambridge, MA, USA.

Trösser, F.; de Givry, S.; and Katsirelos, G. 2021. Improved Acyclicity Reasoning for Bayesian Network Structure Learning with Constraint Programming. In *Proceedings of IJCAI*, 4250–4257.

van Beek, P.; and Hoffmann, H.-F. 2015. Machine learning of Bayesian networks using constraint programming. In *Proc. of International Conference on Principles and Practice of Constraint Programming*, 429–445. Cork, Ireland.

Yuan, C.; and Malone, B. 2013. Learning Optimal Bayesian Networks: A Shortest Path Perspective. *J. of Artificial Intelligence Research*, 48: 23–65.