A Global Acyclicity Constraint for Bayesian Network Structure Learning

Hella-Franziska Hoffmann and Peter van Beek

David R. Cheriton School of Computer Science, University of Waterloo, Canada
`{hrhoffmann,vanbeek}@uwaterloo.ca`

Abstract. In this work we present a global acyclicity constraint that has applications in Bayesian network structure learning. Given a set of vertices $\mathcal{V} = \{V_1, \ldots, V_n\}$ and a set of potential parent sets $\mathcal{P}(V_i)$ (subsets of $\mathcal{V} \setminus \{V_i\}$) for each vertex V_i, we consider a constraint model with variables x_i corresponding to vertices V_i and domains D_i equal to $\mathcal{P}(V_i)$. The acyclicity constraint requires every parent set assignment to induce a directed acyclic graph. We describe a pre-pruning process and give a polynomial-time filtering algorithm that achieves arc consistency for the constraint. We then illustrate how this propagator can be integrated into a branch-and-bound framework to exactly learn Bayesian network structures from complete discrete data.

1 Introduction

A Bayesian network (BN) is a probabilistic graphical model that provides a compact representation of a joint probability distribution of random variables. It consists of a labeled directed acyclic graph (DAG) whose vertices correspond to random variables and whose edges display conditional dependencies among the variables. Each vertex is labeled with a conditional probability distribution that specifies the dependence of the vertex on its parents in the DAG.

One of the main challenges is learning a BN structure from data. A standard approach is to define a score for each candidate BN that measures how well it is supported by the observed data and then search for a BN with maximum score. The resulting optimization problem is known to be NP-hard ([1]) for any reasonable scoring function, even if we limit the number of parents per vertex in the DAG to two.

Heuristic and exact algorithms for this problem have been studied extensively in recent time. Most heuristic approaches are based on local search strategies that fail to guarantee optimality. Most exact algorithms are based on a dynamic programming (e.g. [7,9]) or a mixed integer linear programming approach (e.g. [6]) that are only applicable to small settings (up to 30 variables). Cussens [2] applied a cutting plane algorithm to larger problem instances (up to 60 variables) but it may require several hours of computation to find the optimal network.

De Campos et al. [3] and Etminani et al. [5] both proposed a branch-and-bound algorithm that uses constraints and, in contrast to the above described approaches, provides an any-time procedure; if stopped at any moment, it provides approximate solutions. Here we introduce a different branch-and-bound approach that uses a global acyclicity constraint propagator to speed up the search. The main difference to De Campos et al.'s approach is that while maintaining feasible upper bounds on the optimal score, their intermediate solutions may contain cycles. Their algorithm relies on branching over (forbidden) arcs in order to break cycles in intermediate solutions. Our approach maintains feasible acyclic solutions at all times while providing an easy way to incorporate side constraints.

Our work is also related to Grégoire Dooms work on a general graph constraint solver [4] which, among other constraints, includes a directed acyclic graph constraint over a single graph domain variable. However, this constraint and the corresponding propagator cannot be used to learn BNs efficiently as the number of feasible network structures (and thus the domain size for a graph domain variable) may be exponential in the input size.

Our work is still in progress and its practical applicability remains to be evaluated via experiments. We are convinced that our approach can compete with existing exact algorithms in terms of running time while providing any-time features and the possibility to incorporate side constraints. We also speculate that the filtering algorithm for the global acyclicity constraint may be useful in other applications that require the computation of a DAG.

Outline. The remainder of this paper is organized as follows. In Section 2, we establish the necessary background and notation. We describe a new constraint programming approach to BN structure learning in Section 3 and conclude with a short discussion of future work in Section 4.

2 Background

A Bayesian network can be defined as a triple $(\mathcal{G}, \mathcal{X}, P)$, where $\mathcal{G}$ is a directed acyclic graph with vertices corresponding to random variables $\mathcal{X}$ and P is a collection of conditional probability distributions; see [9] for an example.

In this paper we consider the Bayesian network (BN) structure learning problem. Given complete data over attributes and a scoring function σ, learn a BN structure (DAG) that maximizes the score. Throughout this paper we assume that the scoring function is decomposable. That is, the total score of a BN structure $\mathcal{G}$ can be expressed as the sum of local scores $\sigma_i(\mathcal{G})$ for each vertex V_i in $\mathcal{G}$. Each of these local scores $\sigma_i(\mathcal{G})$ only depends on the parent set of V_i in $\mathcal{G}$. Thus, we can compute local scores $c(V_i, \mathcal{S})$ for each candidate parent set $\mathcal{S}$ of each vertex V_i in a preprocessing step and use them to evaluate the overall quality of candidate BNs.

3 Constraint Programming Approach

In this section we describe a new approach to BN structure learning using constraints. We give a constraint model in Section 3.1. In Section 3.2, we identify some properties of satisfying parent set assignments and describe a propagation technique that can be used to pre-prune the domains. In section 3.3, we present a filtering algorithm that achieves generalized arc-consistency for the acyclicity constraint in polynomial time and discuss the integration of the propagators into a branch-and-bound framework in Section 3.4.

3.1 Constraint Model

Given a set of vertices $\mathcal{V} = \{V_1, \ldots, V_n\}$, a set $\mathcal{P}(V_i) \subseteq 2^{\mathcal{V} \setminus \{V_i\}}$ of potential parent sets for each vertex $V_i \in \mathcal{V}$ and a score $c(V_i, \mathcal{S})$ for every $V_i \in \mathcal{V}$ and every $\mathcal{S} \in \mathcal{P}(V_i)$, we define a variable x_i for every vertex $V_i \in \mathcal{V}$ and set its domain D_i to the candidate set $\mathcal{P}(V_i)$. The model uses a single global *acyclicity constraint* called *acyclic*$(x_1, \ldots, x_n)$ that requires any feasible assignment of a parent set $\mathcal{S}_i \in \mathcal{P}(V_i)$ to each of the vertices V_i to not induce a directed cycle in the corresponding directed graph, i.e. $\mathcal{G} = (\mathcal{V}, \bigcup_{i=1}^{n} \{(P, V_i) : P \in \mathcal{S}_i\})$ is a DAG. The goal is to find a parent set assignment with maximum score that satisfies the acyclicity constraint.

3.2 Pre-Pruning of the Domains

We start with a simple observation. Suppose vertex V_1 is included in every candidate parent set for V_2, then we know that the edge (V_1, V_2) must exist in any DAG corresponding to a feasible parent set assignment. This implies that (V_2, V_1) cannot be an edge of a valid DAG. Thus, we can remove all parent sets that contain V_2 from x_1's domain. We can further prune the domains in a similar way by ruling out larger cycles; identify edges that are included in any feasible parent set assignment, then remove all parent sets from the variable domains that would introduce cycles with repect to these edges.

Identifying necessary edges for the parent set assignment of a vertex V_1 takes $O((m-1)k)$ time where m is the number of potential parent sets in $\mathcal{P}(V_1)$ and k is an upper bound on the number of vertices per candidate parent set. Thus, identifying the *backbone* of all necessary edges takes $O(nmk)$ time.

Note that there is no valid parent set assignment for a given instance if the corresponding backbone contains a directed cycle. We should therefore first check whether the backbone contains any directed cycles. This can simply be done using a depth-first search. Any time we remove a parent set $\mathcal{S}$ from the candidate set of a vertex V_1, new parents can become necessary for V_1 with respect to the resulting candidate set. In this case we can add the corresponding necessary in-arcs to the backbone structure and repeat the pruning process.

Unfortunately, this procedure does not necessary achieve arc consistency, as there are instances in which not even a single edge may be forced by the given candidate parent sets.

3.3 A Polynomial-Time Filtering Algorithm

In this section we describe a propagator for the acyclicity constraint that achieves generalized arc consistency in polynomial time. We first present and analyze an

Algorithm 1: Checking Satisfiability

```
Input: Vertex set $\mathcal{V}$, set $\mathcal{P}(V_i)$ of potential parent sets for each vertex $V_i$ in $\mathcal{V}$.
Output: True if there is a feasible parent set assignment and false otherwise.
        The variables $\mathcal{S}_i$ represent a feasible assignment if there exists one.
$k \leftarrow 0$;
$\mathcal{S}_i \leftarrow nil$ for all $V_i \in \mathcal{V}$;
while $\bigcup_{j=0}^{k-1} W^j \neq \mathcal{V}$ do
    $W^k \leftarrow \emptyset$;
    for all vertices $V_i$ not in $\bigcup_{j=0}^{k-1} W^j$ do
        if $V_i$ has a candidate parent set $\mathcal{S} \in \mathcal{P}(V_i)$ with $\mathcal{S} \subseteq \bigcup_{j=0}^{k-1} W^j$ then
            $\mathcal{S}_i \leftarrow \mathcal{S}$;
            $W^k \leftarrow W^k \cup \{V_i\}$;
    if $W^k = \emptyset$ then return false;
    $k \leftarrow k + 1$;
return true;
```

algorithm that checks satisfiability for given candidate parent sets. We then explain how this algorithm can be used to achieve generalized arc consistency.

Checking Satisfiability. Algorithm 1 can check whether a collection of potential parent sets allows a feasible parent set assignment, i.e. an assignment that represents a DAG. Its correctness is based on the following well-known property of directed acyclic graphs.

Proposition 1. *Let* $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ *be a directed graph and let* $P_{\mathcal{G}}(v)$ *denote the parent set of vertex* v *in* $\mathcal{G}$*. Then* $\mathcal{G}$ *is acyclic if and only if for every non-empty subset* $C \subset \mathcal{V}$ *there is at least one vertex* v *with* $P_{\mathcal{G}}(v) \cap C = \emptyset$*.*

The algorithm works as follows. First, it searches potential sources for the DAG, i.e. vertices for which $\emptyset$ is a potential parent set. We store these vertices in W^0. Note that if a directed graph does not have a source, it must contain a cycle by Proposition 1. Thus, if W^0 remains empty, any possible parent set assignment will contain a cycle. Hence, there is no parent set assignment satisfying the acyclicity constraint. In the next iteration, the algorithm searches for vertices that have a candidate parent set consisting of potential sources only. These vertices form set W^1. Again, if there are no such vertices, then no vertex in $\mathcal{V} \setminus W^0$ has a candidate parent set completely outside $\mathcal{V} \setminus W^0$, which violates the acyclicity characterization of Proposition 1. Thus, there is no consistent parent set assignment. We continue this process until all vertices are included in one of the W^k sets or until we find a contradicting set $\mathcal{V} \setminus (\bigcup_{i=0}^{k} W^i)$ for some k.

Theorem 1. *We can test satisfiability in* $O(n^2 m)$ *time, where* n *is the number of vertices and* m *is an upper bound on the number of candidate parent sets per vertex.*

Achieving Generalized Arc Consistency. The algorithm for checking satisfiability can be used to achieve generalized arc consistency. The main idea is that we can use the satisfiability check to test for each vertex whether every given parent set $\mathcal{S}_i$ in the candidate set $\mathcal{P}(V_i)$ has a support. We simply substitute the set of potential parent sets $\mathcal{P}(V_i)$ for V_i by the set $\{\mathcal{S}_i\}$. A satisfiability check on the resulting instance successfully tests whether there is a consistent parent set assignment containing $V_i \leftarrow \mathcal{S}_i$.

We run this test for every possible value in every domain. If we find a parent set that cannot appear in any feasible solution, we remove this parent set from the corresponding domain. Otherwise, if there is no such parent set, then all of the domain elements have a support and hence the instance is arc-consistent. Lallouet and Legtchenko [8] used a similar technique to achieve arc consistency provided that they are given certificates to check satisfiability.

Theorem 2. *We can enforce arc consistency for the acyclicity constraint in* $O(n^3m^2)$ *steps.*

3.4 The Branch-and-Bound Framework

So far we have described domain pruning techniques for the proposed acyclicity constraint. In order to solve the optimization problem that arises in BN structure learning, we integrate the propagators into a branch-and-bound framework.

The pre-pruning step described in Section 3.2 can easily be used iteratively in a branch-and-bound algorithm as all necessary parents corresponding to the original candidate sets stay necessary for any pruned version of the candidate sets. Thus, it suffices to check whether any in-arcs not already in the backbone graph become necessary. Hence, the more necessary edges we have identified, the less time we need per iteration. An important part of future work is to empirically asses the running time in practice. It might be beneficial to apply this pruning technique only for a certain subset of vertices, restrict the number of iterations or only look for cycles with bounded length.

We speculate that branching over the domain of one vertex in each level of the tree would work best with our filtering algorithm. Running Algorithm 1 on the pre-pruned domains will provide a first (sub-optimal) feasible parent set assignment and create initial sets W^j that can be used to find a good odering of the vertices for the branching process. It is best to first branch over the domain of vertices V_i in the W^j with largest index, as the least changes have to be made to check feasibility of the resulting CSP and these vertices are likely to have smaller domains compared to vertices in any of the lower order W^j sets. Analogously to Etminani et al. [5] we consider the vertices in each level in the order that maximizes the score of the forced parent sets. This way we can use additional bounding rules based on the score like the ones described in [5].

Note that the pruning methods described above all worked independently of the respective scores. There are simple, widely used pre-processing steps that rule out suboptimal candidate parent sets based on their score; see e.g. Lemmas in [3]. We will use these rules in our implementation to prune the initial domains.

Of course, the model also allows us to incorporate additional constraints that are relevant for specific application. De Campos et al. [3] describe constraints that restrict the indegree of a vertex or force certain arcs. We can easily integrate these constraints to further reduce the search space.

4 Conclusion and Future Work

In this work we presented and analyzed the first filtering algorithm for an acyclicity constraint. We described a technique that can be used to pre-prune the domains. We presented an algorithm that can successfully check satisfiability of a given instance and used this satisfiability test to design a poly-time propagator for the acyclicity constraint. We then sketched how these algorithms could be integrated into a branch-and-bound framework to efficiently learn Bayesian network structures from data.

This is work in progress and many open problems remain. Most importantly, the algorithms should be implemented. An extensive empirical study is needed to assess the practical performance and competability of the proposed algorithms.

References

1. D. Chickering, C. Meek, and D. Heckerman. Large-sample learning of Bayesian networks is NP-hard. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 124–133, 2003.
2. J. Cussens. Bayesian network learning with cutting planes. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 153–160, 2011.
3. C. P. de Campos, Z. Zeng, and Q. Ji. Structure learning of bayesian networks using constraints. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 113–120, 2009.
4. G. Dooms. *The CP(Graph) Computation Domain in Constraint Programming*. PhD thesis, Université Catholique de Louvain, Belgium, 2006.
5. K. Etminani, M. Naghibzadeh, and A. R. Razavi. Globally optimal structure learning of bayesian networks from data. In *Proceedings of the 20th International Conference on Artificial Neural Networks: Part I*, ICANN'10, pages 101–106, Berlin, Heidelberg, 2010. Springer-Verlag.
6. T. Jaakkola, D. Sontag, A. Globerson, and M. Meila. Learning Bayesian network structure using lp relaxations. *Journal of Machine Learning Research - Proceedings Track*, 9:358–365, 2010.
7. M. Koivisto and K. Sood. Exact Bayesian structure discovery in Bayesian networks. *J. Mach. Learn. Res.*, 5:549–573, 2004.
8. A. Lallouet and A. Legtchenko. From satisfiability to consistency through certificates: application to partially defined constraints. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pages 415–416, New York, NY, USA, 2006. ACM.
9. T. Silander and P. Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI-06)*, pages 445–452, 2006.