# GOBNILP: Learning Bayesian network structure with integer programming

**James Cussens** JAMES.CUSSENS@BRISTOL.AC.UK
*Dept of Computer Science, University of Bristol, UK*

## Abstract

The GOBNILP system for learning Bayesian networks is presented. Both the Python and C implementations are discussed. The usefulness of learning multiple BNs is highlighted. Current work on 'pricing in' new integer programming variables is presented.

**Keywords:** Bayesian network structure learning, integer programming, C, SCIP, Python

## 1. Introduction

Score-based Bayesian network structure learning (BNSL) is an NP-hard task—even when there are no latent or selection variables, the score function is decomposable into local scores and when at most two parents are allowed for each node in the learned BN. A natural approach is thus to resort to approximate learning where there is no guarantee that the learned network has the highest score. GOBNILP takes a different approach and attempts to find a guaranteed optimal BN. If it fails in this task it can return the best BN it has found and an upper bound on an optimal solution.

GOBNILP casts score-based BNSL as an integer programming (IP) problem. This allows GOBNILP to delegate much of the hard work of solving to existing IP solvers. The declarative nature of the IP formulation is also advantageous. An IP problem is just a collection of variables and constraints; if the user wishes to add constraints on what BNs can be learned these can be added to the constraint store with no need to change the underlying solver.

In this paper the main features of GOBNILP will be described. GOBNILP has two separate implementations: a version written in C which uses SCIP Gamrath et al. (2020) for IP solving and a version in Python which uses Gurobi. The latest versions of both systems are available as git repositories at `https://bitbucket.org/jamescussens/gobnilp/` and `https://bitbucket.org/jamescussens/pygobnilp/` respectively. This paper describes commit 518d397 for the C version and commit a3108d8 for the Python version. Note that the latest C version only works with SCIP 7.0.0 (or 7.0.1) and that the Python version assumes that Anaconda Python is being used. Of course, SCIP must be installed for the C version and Gurobi for the Python version. SCIP is free for academic use and a Gurobi licence is also free for academic use. Both versions of GOBNILP are distributed under the GPL Version 3.

## 2. Python version

I will refer to the Python version of GOBNILP as PYGOBNILP. The PYGOBNILP manual explains how to install PYGOBNILP and what its dependencies are so that will not be explored here. There are three main ways of using PYGOBNILP:

1. Using the Python script `rungobnilp.py`

```
Use Gurobi for Bayesian network learning

positional arguments:
  data_source           File containing data or local scores

optional arguments:
  --score SCORE         Name of scoring function used for computing local
                        scores. Must be one of the following: BDeu, BGe,
                        DiscreteLL, DiscreteBIC, DiscreteAIC, GaussianLL,
                        GaussianBIC, GaussianAIC, GaussianL0. (default: BDeu)
  --palim PALIM, -p PALIM
                        Maximum size of parent sets. (default: 3)
  --nsols NSOLS, -n NSOLS
                        Number of BNs to learn (default: 1)
  --kbest               Whether the nsols learned BNs should be a highest
                        scoring set of nsols BNs. (default: False)
  --mec                 Make only one BN per Markov equivalence class
                        feasible. (default: False)
  --polytree            Ensure that the learned DAG(s) is/are polytrees.
                        (default: False)
  --chordal             Ensure that the learned DAG(s) has/have no
                        immoralities. (default: False)
  --edge_penalty EDGE_PENALTY
                        The local score for a parent set with p parents will
                        be reduced by p*edge_penalty. (default: 0.0)
  --gurobi_params GUROBI_PARAMS [GUROBI_PARAMS ...]
                        Gurobi parameter settings. (default: None)
```

Figure 1: A subset of PYGOBNILP command line arguments

2. Calling the PYGOBNILP API, either from a Python module or the Python interpreter.

3. From within an R session using the `reticulate` R library.

A set of Jupyter notebooks, which can be viewed online, provides a number of examples of interactive use of PYGOBNILP using both Python and R. Fig 1 shows a subset of the command line arguments for PYGOBNILP (which can be viewed using `python rungobnilp.probability -h`). Fig 1 provides a rough idea of how PYGOBNILP can be used: input can be data (discrete or continuous) or precomputed local scores. Four scoring functions are available for discrete data, and five for continuous data. A number of constraints can be used: a limit on parent set size, and whether the learned BN must be a polytree and/or chordal (containing no immoralities). More complex constraints (e.g. conditional independence constraints) are available using the PYGOBNILP API. Note that, using the `gurobi_params` option, the user can influence the solving approach of the underlying Gurobi solver; for example asking Gurobi to put more effort into quickly finding a reasonably good BN at the expense of making progress towards finding a provably optimal one.

$$G_1 = \begin{array}{ccc} X_1 & & X_4 \\ \downarrow & & \downarrow \\ X_2 & \leftarrow L \rightarrow & X_3 \end{array}$$

Figure 2: BN with a latent variable

```
**********                              **********
BN has score -244249.877755693          BN has score -244249.877842469
**********                              **********
X1<- -48716.91294283306                  X1<- -48716.91294283306
X4<- -67090.7803124027                   X4<-X3 -66037.14696747699
X2<-X1 -60747.72209196292                X3<- -69148.94959180673
X3<-X1,X2,X4 -67694.46240849455          X2<-X1,X3,X4 -60346.8683403523
```

Figure 3: DAGs learned from 'non-DAG' data

Finding multiple BNs (using the `nsols` option) comes 'for free' since we are using an IP solver. Using the `kbest` flag we can ensure that the $k$ returned BNs are the $k$ highest-scoring BNs (assuming PYGOBNILP manages to solve to optimality). By setting the `mec` flag we can ensure that only one BN from each Markov equivalence class is returned.

The ability to learn a set of high-scoring BNs opens up the possibility of using a DAG-learning system (like GOBNILP) to learn more general conditional independence (CI) models. This is because any CI model (not even just those corresponding to probability distributions) can be represented by a set of DAGs (e.g. a set where the DAGs are in 1-1 correspondence with the CI relations that obtain in the model Studený (2005).)

For example, the DAG $G_1$ in Fig 2 with latent variable $L$ is not Markov equivalent to any DAG on $X_1, X_2, X_3, X_4$. Asking PYGOBNILP to learn non-Markov equivalent BNs from 100,000 $(X_1, X_2, X_3, X_4)$ datapoints drawn from $G_1$ using the BIC score results in the top two (equally) highest-scoring DAGs shown in Fig 3. Both DAGs 'overfit' the simulated data: no DAGs fits exactly and underfitting attracts a big penalty. The true independence model, which can be represented by a maximal ancestral graph (MAG), is the union of the CI models represented by the two learned DAGs.

## 3. C version using SCIP

The C/SCIP version of GOBNILP will be called SCIPGOBNILP. SCIPGOBNILP is typically faster than PYGOBNILP although harder to install and run. In addition, the current version of SCIPGOBNILP does a poorer job of learning multiple DAGs than PYGOBNILP. Despite these differences the same basic approach to BNSL is taken in both implementations Bartlett and Cussens (2017).

A big advantage of using SCIP is that it provides a framework for *pricing* which is the creation of IP variables during the course of solving an IP. Both PYGOBNILP and SCIPGOBNILP generate *constraints* on the fly, but only SCIP and not Gurobi provides proper support for pricing.

In GOBNILP each possible parent set for a BN variable is encoded as a binary IP variable (whose objective coefficient is the relevant 'local score'). In the context of BNSL, pricing corresponds to

```
 time | cons|cols  |rows |cuts |
p 0.4s|2067 | 1923 |2064 |   0 |
  0.5s|2067 | 1984 |2064 |   0 |
  1.7s|2067 | 2043 |2064 |   0 |
  1.7s|2067 | 2083 |2110 |  46 |
..
L12.8s|2064 | 2212 |2136 | 411 |
```

Figure 4: Abbreviated SCIPGOBNILP output: creating new parent sets during solving

creating new parent sets (= IP variables) during the course of solving. The advantage of delaying IP variable creation in this way is that one can use the solution of the current linear relaxation (specifically the dual values associated with that solution) to constrain the search for useful parent sets: only those which might possibly lead to a better solution to the linear relaxation are worth adding. This may provide a solution to an existing bottleneck where for some problems, to ensure a truly globally optimal solution, very many IP variables (= parent sets) must be created before solving even begins.

Although not yet ready for the main branch of SCIPGOBNILP a working version of SCIPGOBNILP that uses pricing does exist. Output showing pricing is shown in Fig 4 (pricing branch, commit 20030cc). This is for the `alarm_100` dataset with a parent set size limit of 4 and where initially only parent sets of size at most 2 are created. Solving starts with 1923 binary variables only 591 of which are parent set variables; the other 1332 variables are indicator variables for arrows and adjacencies. By the end of solving an additional 2212-1923=289 parent set variables have been created making 880 parent set variables in total. In contrast, without pricing 1293 parent set variables are created.

## References

M. Bartlett and J. Cussens. Integer linear programming for the bayesian network structure learning problem. *Artificial Intelligence*, 244:258 – 271, 2017. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2015.03.003. URL http://www.sciencedirect.com/science/article/pii/S0004370215000417. Combining Constraint Solving with Mining and Learning.

G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020. URL http://www.optimization-online.org/DB_HTML/2020/03/7705.html.

M. Studený. *Probabilistic Conditional Independence Structures*. Springer, 2005.