pygobnilp manual (version 1.0)

James Cussens
University of York

Thursday 30th January, 2020

Contents

1 Introduction

This manual aims to explain how to use pygobnilp, the Python version of the GOBNILP algorithm for learning Bayesian network structure. The focus in this manual is on using the Python script `rungobnilp.py`. If you are interested in using pygobnilp interactively (either using the Python interpreter or via R) please consult the Jupyter notebooks which can be viewed at `https://nbviewer.jupyter.org/urls/bitbucket.org/jamescussens/pygobnilp/raw/master/notebooks/GOBNILP_notebooks.ipynb`. API documentation is available here: `https://pygobnilp.readthedocs.io/en/latest/`.

pygobnilp is slower than the C version of GOBNILP, which is available via the main GOBNILP page: `https://www.cs.york.ac.uk/aig/sw/gobnilp/`. It is however easier for trying out new ideas and has more 'scoring' functions implemented for it.

2 Installation

2.1 Dependencies

pygobnilp depends on (1) a number of Python packages (scipy, pygraphviz, matplotlib, networkx, pandas, numpy, scikit-learn and numba) and (2) the Gurobi MIP solver. pygraphviz also requires graphviz `https://www.graphviz.org/` to be installed.

Although one can install all these separately the easier option is to install Anaconda Python and Gurobi together. Just go here: `https://www.gurobi.com/get-anaconda/`. Installing Anaconda will get you most of the required packages but not (at present) pygraphviz, which, once Anaconda is in place, you can install with: `conda install pygraphviz`. graphviz is not a Python package and has to be installed separately (if you do not already have it on your system).

Gurobi is a commercial system and requires a licence to run. However, an academic licence is free, see `https://www.gurobi.com/academia/academic-program-and-licenses/`.

2.2 Installing pygobnilp

pygobnilp consists of a Python package (called `pygobnilp`) and the Python script `rungobnilp.py`. The `pygobnilp` package contains two modules `gobnilp.py` and `scoring.py` (as well as an empty `__init__.py` file). So one installation option is to just to download or clone the pygobnilp git repository `https://bitbucket.org/jamescussens/pygobnilp` and then just make sure that the `pygobnilp` directory is in a location where your Python installation looks for Python packages. Note that this will get you the current development version of pygobnilp which may differ (hopefully in a positive way!) to the version (1.0) described here. One advantage of this approach is that the pygobnilp git repository also contains some data files and constraint files which are mentioned in the Jupyter notebooks and used as examples in this manual.

Alternatively, you can install pygobnilp from PyPI:

```
pip install pygobnilp
```

This will install the `pygobnilp` package in wherever is the normal place for your Python installation. The script `rungobnilp.py` will also be downloaded. This route will always get you the latest stable version of pygobnilp, currently version 1.0.

3 Running pygobnilp

This section explains how to use pygobnilp by running the Python script `rungobnilp.py`. This script has very many optional command line arguments which can be listed by running `python rungobnilp.probability --help` producing the following output:

```
usage: rungobnilp.py [-h] [--noheader] [--comments COMMENTS] [--delimiter]
                     [--end END] [--score SCORE] [--k K] [--ls]
                     [--standardise] [-p PALIM] [--alpha ALPHA]
                     [--alpha_mu ALPHA_MU] [--alpha_omega ALPHA_OMEGA] [-s]
                     [-n NSOLS] [--kbest] [--mec] [--consfile CONSFILE]
                     [--settingsfile SETTINGSFILE] [--nopruning]
                     [--edge_penalty EDGE_PENALTY] [--noplot] [--noabbrev]
                     [--output_scores OUTPUT_SCORES] [-o OUTPUT_STEM]
                     [--nooutput_dag] [--nooutput_cpdag]
                     [--output_ext OUTPUT_EXT] [-v VERBOSE] [-g]
                     [--gurobi_params GUROBI_PARAMS [GUROBI_PARAMS ...]]
                     data_source

Use Gurobi for Bayesian network learning

positional arguments:
  data_source           File containing data or local scores

optional arguments:
  -h, --help            show this help message and exit
  --noheader            For continuous data only: The first non-comment line
                        in the input file does not list the variables.
                        (default: False)
  --comments COMMENTS   For continuous data only: Lines starting with this
                        string are treated as comments. (default: #)
  --delimiter           For continuous data only: String used to separate
                        values. If not set then whitespace is used. (default:
                        None)
  --end END             End stage for learning. If set to 'local scores'
                        execution stops once local scores are computed
                        (default: output written)
  --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)
  --k K                 Penalty multiplier for penalised log-likelihood scores
                        (eg BIC, AIC) or tuning parameter ('lambda^2) for l_0
                        penalised Gaussian scoring (as per van de Geer and
                        Buehlmann) (default: 1)
```

```
--ls                  For Gaussian scores, make unpenalised score -(1/2) *
                      MSE, rather than log-likelihood (default: False)
--standardise         Standardise continuous data. (default: False)
-p PALIM, --palim PALIM
                      Maximum size of parent sets. (default: 3)
--alpha ALPHA         The equivalent sample size for BDeu local score
                      generation. (default: 1.0)
--alpha_mu ALPHA_MU   Imaginary sample size value for the Normal part of the
                      normal-Wishart prior for BGe scoring. (default: 1.0)
--alpha_omega ALPHA_OMEGA
                      Degrees of freedom for the Wishart part of the normal-
                      Wishart prior for BGe scoring. Must be at least the
                      number of variables. If not supplied 2 more than the
                      number of variables is used. (default: None)
-s, --scores          The input consists of pre-computed local scores (not
                      data) (default: False)
-n NSOLS, --nsols 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)
--consfile CONSFILE   A file (Python module) containing user constraints.
                      (default: None)
--settingsfile SETTINGSFILE
                      A file (Python module) containing values for the
                      arguments for Gobnilp's 'learn' method Any such values
                      override both default values and any values set on the
                      command line. (default: None)
--nopruning           No pruning of provably sub-optimal parent sets.
                      (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)
--noplot              Prevent learned BNs/CPDAGs being plotted. (default:
                      False)
--noabbrev            When plotting DO NOT to abbreviate variable names to
                      the first 3 characters. (default: False)
--output_scores OUTPUT_SCORES
                      Name of a file to write local scores (default: None)
-o OUTPUT_STEM, --output_stem OUTPUT_STEM
                      Learned BNs will be written to 'output_stem.ext' for
                      each extension defined by `output_ext`. If multiple
                      DAGs have been learned then output files are called
                      'output_stem_0.ext', 'output_stem_1.ext' ... No DAGs
                      are written if this is not set. (default: None)
```

```
--nooutput_dag        Do not write DAGs to any output files (default: False)
--nooutput_cpdag      Do not write CPDAGs to any output files (default:
                      False)
--output_ext OUTPUT_EXT
                      Comma separated file extensions which determine the
                      format of any output DAGs or CPDAGs. (default: pdf)
-v VERBOSE, --verbose VERBOSE
                      How much information to show when adding variables and
                      constraints (and computing scores) (default: 0)
-g, --gurobi_output   Whether to show output generated by Gurobi. (default:
                      False)
--gurobi_params GUROBI_PARAMS [GUROBI_PARAMS ...]
                      Gurobi parameter settings. (default: None)
```

3.1 Simple usage

The simplest way to use pygobnilp is to learn a single Bayesian network from discrete data using default parameters. Suppose `discrete.dat` is a file containing discrete data, then doing this:

```
python rungobnilp.py discrete.dat
```

will learn a BN with maximal BDeu score under default parameters. A textual representation of the BN and associated CPDAG and a plot will be displayed, but no output files written. In the generated plot the DAG will be shown with black and red arrows. Red arrows are arrows that have the same orientation in every Markov equivalent DAG, black arrows are those without this property (they are *reversible*). The plot should remain visible until you dismiss it.

Discrete data is expected to have the following format (an example of which is given in Fig 1):

1. All fields should be separated by white space.
2. The first line of the file should be the names of the variables.
3. The second line should be the *arities* of the variables, where a variable's arity is the number of values it can have.
4. All lines after the second are datapoints where each datapoint is a sequence of (white space separated) values for the variables. Values can be any string, they do not have to be numbers.

3.1.1 Parent set size limits

It is important to realise that by default **there is a limit on each BN node having at most 3 parents**, so the learned BN may not have an optimal BDeu score. It is possible to alter this default limit using the `--palim` option:

```
A B C D E F
3 3 3 3 3 2
b c b a b b
b a c a b b
a a a a a a
a a a a b b
a a b c a a
c c a c c a
```

Figure 1: Discrete data for 6 variables with 6 datapoints. This is in the correct format for default parameters.

```
python rungobnilp.py --palim 99 discrete.dat
```

If there are fewer than 100 variables in `discrete.dat` then setting the limit this high effectively removes it. By raising the limit a higher scoring BN may be found, but learning may take considerably longer. Of course, one may have good reason to lower this limit; setting it to 1 leads to learning tree-shaped BNs.

3.1.2 Choosing a BN score

To use scores other than BDeu one should use the `--score` option. Allowed values for discrete data are BDeu, DiscreteLL, DiscreteBIC and DiscreteAIC. DiscreteLL looks for a BN with maximal fitted log-likelihood score, so without constraints, will always return a maximally connected BN. (Using DiscreteLL with a parent set size limit of 1 is effectively learning a Chow-Liu tree.)

For continuous data the options are currently: BGe, GaussianLL, GaussianBIC, GaussianAIC and GaussianL0. GaussianLL looks for a BN with maximal fitted Gaussian log-likelihood score, so without constraints, will always return a maximally connected BN. GaussianL0 is GaussianLL with an ℓ_0 penalty and is the "ℓ_0 penalized maximum likelihood" analysed by van de Geer and Bühlmann [2013].

You do not need to specify whether discrete or continuous data is being supplied, the choice of score determines which type is expected. An example of continuous data which is in the correct format as input with default parameter settings is given in Fig 2. The first line gives the variable names, comment lines start with '#' and variable values are separated by white space.

By setting parameters to non-default values continuous data in other formats can be used.

--noheader If set then the first non-comment line is not interpreted as variable names. Instead, the variable names $X1, X2, \ldots Xn$ will be used.

--comments Use this to set a different symbol for comment lines

--delimiter Use this to set a delimiter between values

```
A B C D E F G
# this is a comment
1.1 1.93 7.07 8.66 0.88 24.71 9.21
-0.24 11.33 24.3 23.35 7.04 36.81 3.67
1.85 3.03 11.08 11.05 3.83 22.01 2.4
0.83 3.85 11.22 11.93 1 23.28 6.08
```

Figure 2: Continuous data for 6 variables with 4 datapoints. This is in the correct format for default parameters.

Some scores have hyperparameters. All hyperparameters have default values and some of these can be altered:

--alpha The *equivalent sample size* for BDeu scoring. Default is 1.

--alpha_mu Imaginary sample size value for the Normal part of the normal-Wishart prior for BGe scoring. Default is 1.

--alpha_omega Degrees of freedom for the Wishart part of the normal-Wishart prior for BGe scoring. Must be at least the number of variables. Default is 2 more than the number of variables.

--k Penalty multiplier for penalised log-likelihood scores (e.g. BIC, AIC). This should be 1 for normal BIC/AIC. For ℓ_0 penalised Gaussian scoring this is the tuning parameter (λ^2). Default is 1.

--ls For Gaussian scores, make the unpenalised score -(1/2) * MSE (mean-squared error), rather than log-likelihood.

3.2 Learning multiple BNs

By default, pygobnilp learns a single BN which is optimal for the given score subject to any constraints (such as parent set size limit). If you wish to learn several BNs you should use the --nsols option. For example, this:

```
python rungobnilp.py --palim 99 discrete.dat --nsols 4
```

learns the optimal BN for the given input—and 3 other BNs. However, in this approach there is no guarantee that the 3 additional BNs are the next 3 best BNs. They are just three BNs that Gurobi (the underlying MIP solver) could find quickly.

To get, e.g. the top 4 BNs one must use the --kbest flag:

```
python rungobnilp.py --palim 99 discrete.dat --nsols 4 --kbest --nopruning
```

Note that in this example the flag --nopruning has been set. By default, pygobnilp will 'prune away' choices of parent sets for BN nodes which cannot possibly occur in an optimal BN. This speeds up learning considerably. However, when

searching for sub-optimal networks it is necessary to turn pruning off to ensure all possible BNs are available; this is what the --nopruning flag does. A more sophisticated approach to finding high-scoring but sub-optimal BNs (based on 'partial pruning') has been devised by Liao et al. [2019] but is not implemented in the current version of pygobnilp.

When learning multiple BNs one is typically only interested in finding one BN in each Markov equivalence class. To make this happen use the --mec flag:

```
python rungobnilp.py --palim 99 discrete.dat --nsols 4 --kbest --mec \
                     --nopruning
```

If you want to learn only BNs which are above a certain score then use the --gurobi_params optional parameter. Each argument supplied to --gurobi_params should be of the form param=val, where param is the name of a Gurobi Model parameter and val is an allowed value for that parameter. If 'val' contains a '.' then it is interpreted as a float, otherwise if it looks like a number it is treated as an integer, failing that it is treated as a string. Cutoff is a Gurobi Model parameter that can be set to exclude any solution worse than the supplied value. So doing:

```
python rungobnilp.py --palim 99 discrete.dat --nsols 40 --mec \
                     --nopruning --gurobi_params Cutoff=-24050.0
```

will return at most 40 non-Markov equivalent BNs with a score better than -24050.0. Fewer than 40 will be returned if there are not that many with a sufficiently high score. Note that `--kbest` was not used in this example, so the BNs returned may not be the best 40, although clearly if fewer than 40 were returned we have all allowed BNs and thus the `--kbest` would only affect the order in which BNs were returned.

3.3 Local score input and output

Internally pygobnilp computes *local scores* from the data before it starts the search for an optimal BN. There is a local score for each choice of parent set for each BN variable (although typically not all of these are computed due to pruning). pygobnilp can learn BNs directly from pre-computed local score files. Suppose `asia_10000.dat.3.jkl` was a file containing local scores then pygobnilp can take these local scores, rather than some data, as the input by using the --scores parameter:

```
python rungobnilp.py asia_10000.dat.3.jkl --scores
```

(**NB** --scores is a flag (takes no arguments) indicating that the input is local scores, whereas --score is a parameter specifying the name of a local scoring function to use.) Learning is quicker, often considerably so, when learning from local scores since pygobnilp does not compute local scores rapidly. (The C version of GOBNILP is much faster in this respect).

Local scores can also be output using the --output_scores parameter. Doing the following will output the local scores computed from the dataset discrete.dat to a file called discrete.scores

```
python rungobnilp.py discrete.dat --output_scores discrete.scores
```

If you just want to output local scores and not learn a BN you can use the --end parameter, to stop pygobnilp once local scores have been computed:

```
python rungobnilp.py discrete.dat --output_scores discrete.scores --end 'local scores'
```

Note that it is necessary to put quotes around the argument to --end since it contains a space character. The next section describes the file format for local scores.

3.3.1 Local score file format

- The first line is the total number of BN variables.
- The rest of the file has a section for each variable.
  - The section for a variable starts with a single line with the name of the variable and the number of parent sets recorded for it. Variable names can be any string of characters not containing white space. For example, `1`, `var1`, or `variable_one` are all permissible variable names; `variable one`, or `"var one"` are not. For example

    ```
    0 81
    ```

    states that variable 0 has 81 candidate parent sets.
  - The remaining lines in the section for a variable are local ('family') scores. Each such line starts with the score itself, the number of parents in the parent set and then the parents themselves, if any. So, for example,

    ```
    -106.565548505 3 13 15 11
    ```

    states that parent set $\{13, 15, 11\}$ has score -106.565548505 (and contains 3 members).

This format originated with the work done by Jaakkola et al. [2010].

3.4 Output of learned BNs and CPDAGs

By default pygobnilp prints out a description of learned BNs and CPDAGs to the terminal window and produces a plot. To generate a PDF of each learned BN and CPDAG just set the flag --output_stem to some value. For example doing this:

```
python rungobnilp.py discrete.dat --output_stem foo
```

will create two files `foo.pdf` and `foo_cpdag.pdf` showing the learned BN and learned CPDAG, respectively. To suppress printing of the CPDAG do:

```
python rungobnilp.py discrete.dat --output_stem foo --nooutput_cpdag
```

To suppress printing of the BN do:

```
python rungobnilp.py discrete.dat --output_stem foo --nooutput_dag
```

If multiple DAGs/CPDAGs are learned each will be printed to a separate PDF file, the name of which will include a suitable index. By default only PDF output is generated, but one can supply a list of comma separated formats to the --output_ext parameter to get other formats. For example,

```
python rungobnilp.py discrete.dat --output_stem foo --output_ext pdf,svg
```

will generate both PDF and SVG files. pygobnilp uses the draw method of the AGraph class from pygraphviz to generate output. Check the relevant documentation: `https://pygraphviz.github.io/documentation/latest/reference/agraph.html` for a full list of possible formats.

It is sometimes convenient to suppress the display of a plot of learned BNs (particularly if the learned DAG will be written to a file and/or if very many are being learned!), this can be done with the --noplot flag.

3.5 Using a settings file

The script `rungobnilp.py` is basically a way of collecting values to send to the `learn` method of the Gobnilp class (which is in the module `pygobnilp.gobnilp`). It is sometimes more convenient to put these values in a Python module and ask `rungobnilp.py` to consult this file using the --settingsfile parameter. For example suppose the file `setex.py` contained these two lines (and was thus a valid Python module):

```
palim = None
output_stem = 'bar'
```

then doing the following:

```
python rungobnilp.py discrete.dat --settingsfile setex.py
```

would learn a BN from the given dataset with no limit on parent set size and with output going to files with file stem bar. To use this approach you should consult the documentation on the `learn` method, which you can find here `https://pygobnilp.readthedocs.io/en/latest/#pygobnilp.gobnilp.Gobnilp.learn`. You can supply some values in the settings file and others on the command line if you wish. If you supply a value for some parameter (e.g. palim) both in the settings file and on the command line, the value in the settings file is used.

3.6 Using a constraints file

It is possible to put constraints on which BNs can be learned. For example, particular arrows between variables can be either made obligatory or forbidden

and particular conditional independence relations can be required. It is also possible to add arbitrary Gurobi constraints (perhaps using additional MIP variables). When using `rungobnilp.py` these constraints should be included in a Python module and the file for that module should be supplied using the --consfile parameter.

For example, if the file `cons1.py` contained the following definition:

```
def forbidden_ancestors(gobnilp):
    return [('A','E'),('B','E')]
```

and we ran `rungobnilp.py` like this:

```
python rungobnilp.py discrete.dat --consfile cons1.py
```

then a BN would be learned where node A was not an ancestor of E and node B was not an ancestor of E. For more information on how to use constraints files please consult these two Jupyter notebook examples:

`https://nbviewer.jupyter.org/urls/bitbucket.org/jamescussens/pygobnilp/raw/master/notebooks/Using_a_constraints_file.ipynb`

and

`https://nbviewer.jupyter.org/urls/bitbucket.org/jamescussens/pygobnilp/raw/master/notebooks/Defining_general_MIP_constraints_in_a_constraints_file.ipynb`.

3.7 Getting information on the learning process

It is possible to get information on what pygobnilp is doing when learning. This is useful particularly on big examples (to check that some progress is being made!). To see the output from Gurobi (which is normally suppressed) use the --gorubi_output flag. This will show, among other things the 'gap' between the score of the best BN found so far and an upper bound on the best possible score. To see information about how pygobnilp constructs a MIP problem to give to Gurobi to solve, set --verbose to a non-zero value. Higher values give more verbose output.

4 Citing pygobnilp

If you wish to cite pygobnilp Cussens [2011] is the most appropriate option.

References

James Cussens. Bayesian network learning with cutting planes. In Fabio G. Cozman and Avi Pfeffer, editors, *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI 2011)*, pages 153–160, Barcelona, 2011. AUAI Press. URL `http://uai.sis.pitt.edu/papers/11/p153-cussens.pdf`.

Tommi Jaakkola, David Sontag, Amir Globerson, and Marina Meila. Learning Bayesian network structure using LP relaxations. In *Proceedings of 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 358–365, 2010. Journal of Machine Learning Research Workshop and Conference Proceedings.

Zhenyu A. Liao, Charupriya Sharma, James Cussens, and Peter van Beek. Finding all Bayesian network structures within a factor of optimal. In *Proc. AAAI-19*, 2019.

Sara van de Geer and Peter Bühlmann. ℓ_0-penalized maximum likelihood for sparse directed acyclic graphs. *The Annals of Statistics*, 41(2):536–567, 2013.