# GOBNILP 1.3 User/Developer Manual*

James Cussens, Mark Bartlett
University of York

February 20, 2013

## Contents



*GOBNILP version 1.3 and higher is supported by MRC Project Grant G1002312

# 1 Downloading and installing

GOBNILP is distributed under the GNU Public Licence (version 3) and is available for download via `http://www.cs.york.ac.uk/aig/sw/gobnilp`. The following installation instructions assume that you are using Linux. (We suspect it would not be too hard to install under Windows or Mac as long as you have a C compiler and the 'make' utility.)

1. Assuming you meet the relevant licence conditions ( `http://scip.zib.de/licence.shtml` ), download SCIP ( `http://scip.zib.de`) if you do not already have it installed. GOBNILP works with both SCIP 2.1.1 and SCIP 3.0.0. Performance is sometimes better with 2.1.1 sometimes better with 3.0.0. SCIP 3.0.0 was the current SCIP version at time of writing.

2. Install SCIP following the instructions that come with it. If you have CPLEX installed be sure to make a CPLEX-linked version of SCIP.

3. Let `<version>` denote the GOBNILP version that you have downloaded. Do `tar zxf gobnilp<version>.tar.gz` (if you downloaded the `.tgz` archive) or `unzip gobnilp<version>.zip` (if you downloaded the `.zip` file) to put the GOBNILP distribution in a directory of your choosing. This directory will be called `$(GOBNILPDIR)` in what follows.

4. Edit the file `$(GOBNILPDIR)/Makefile` so that `SCIPDIR` is the directory where you have chosen to install SCIP. (This will be the directory you were in when you typed `make` (or `make LPS=cpx`) to compile SCIP.)

5. Ensure `$(GOBNILPDIR)` is your current working directory. Type `make` or, if you have CPLEX installed, `make LPS=cpx`.

6. If you are using SCIP 3.0.0 will get an error message warning that `SCIPcreateEmptyRow` is deprecated. You can ignore this.

7. An executable `$(GOBNILPDIR)/bin/gobnilp` will be created. This will be a symbolic link to a file with a much longer name.

If you are using GOBNILP we would appreciate it if you let us know. Please do so by email to `james.cussens@york.ac.uk` and put gobnilp somewhere in the subject header. We are particularly keen to hear about any problems you may have with GOBNILP.

## 2 Input format

### 2.1 Using local scores

GOBNILP accepts input in the form of local scores (e.g. local BDeu scores). The format of the data is as follows:

- 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 Jaakkola et al [5]. Example input files ( `alarm_10000_1_3.scores` and `alarm_100_1_3.scores` ) are included in the GOBNILP distribution. Further gzipped example input files in the right format are available at `http://www-users.cs.york.ac.uk/~jc/research/uai11/ua11_scores.tgz` and at `http://www.cs.york.ac.uk/aig/sw/gobnilp/data`. There is some overlap between the scores files at these two URLs, but since there has been some renaming (i.e. renumbering) of the variables they are not directly comparable.

### 2.2 Using data

At present, the GOBNILP executable only accepts local scores input, so an external program is required to generate these from data. Included in the GOBNILP distribution is a C program `scoring.c` which generates local BDeu scores from data. Compiling it as follows will produce an executable called `scoring`:

```
gcc -o scoring -O3 scoring.c -lm
```

`scoring` accepts data in the following format.

- Any line starting with `#` is seen as a comment and ignored.

- The first non-comment line contains a single positive integer which is the number of variables. Call this value $p$.
- The next (second) non-comment line contains $p$ positive integers separated by spaces. The $i$th integer in this line is the arity of the $i$th variable.
- The next (third) non-comment line contains a single positive integer which is the number of datapoints.
- Each remaining non-comment line contains $p$ non-negative integers separated by spaces. Each such line represents a single datapoint. The $i$th integer is the value of variable $i$ for that datapoint. Values are integers ranging from 0 up to, but not including, the arity of the variable. See Section 2.2.1 for how to deal with values starting at 1 rather than 0.

To generate local BDeu scores from data in this format do:

```
scoring <datafile> <ESS> <palim>
```

`datafile` is the name of the data file. If `-` is given as the datafile, scoring reads from standard input. `ESS` is the *effective sample size* used when computing BDeu scores. `palim` is the maximum number of parents allowed in any local score (and hence the maximum number of parents in any BN learnt from these local scores). If you do not wish to effect a limit on parent set sizes just set this number to be suitably high. `scoring` writes the scores to standard output, so typically you would redirect this to a (suitably named) file.

The GOBNILP distribution contains a data file `alarm_10000.data` and a score file `alarm_10000_1_3.scores` generated from the data using the `scoring` executable as follows:

```
scoring alarm_10000.data 1 3 > alarm_10000_1_3.scores
```

You may wish to re-generate `alarm_10000_1_3.scores` just to check all is well. To create scores from a gzipped data set you can pipe into `scoring`. For example:

```
gzip -dc alarm_10000.data.gz | scoring - 1 3 > alarm_10000_1_3.scores
```

### 2.2.1 Variable values starting at 1

If you have data where the values of variables run from 1 up to and including the arity of the variable. Just change this line in `scoring.c`

```
#define OFFSET 0
```

to

```
#define OFFSET 1
```

and recompile.

#### 2.2.2 Limitations of scoring code

The speed of scoring depends on the size of the input, where this size is determined by the number of variables and datapoints. But the limit on parent set size is the most important parameter. The scoring code can deal with datasets with many variables (in the hundreds) if the limit on parent sets is set low. To get an idea of how quickly it can deal (or not!) with various sorts of inputs please refer to the benchmarks on `http://www.cs.york.ac.uk/aig/sw/gobnilp`. (Or just experiment yourself.)

#### 2.2.3 Scoring algorithm

The scoring algorithm works by computing parent set scores for each child variable independently. (Presumably one could thus parallelise it without too much work.) Parent sets are generated in layers, where each parent set in a layer has the same cardinality. Layers are generated in increasing cardinality. Parent sets (for a given child) which cannot exist in an optimal BN are not output. **This means that parentset-child combinations for a second-best BN may be missing from the output.** The algorithm uses a slightly modified version of the pruning approach devised by de Campos and Ji [4]. See Appendix A for a proof of the correctness of the implemented pruning method.

# 3 Running GOBNILP

GOBNILP expects the name of the file containing the local scores as the last command line argument. For example:

```
bin/gobnilp alarm100_1_3.scores
```

If the filename is given as `-` then GOBNILP will read scores from standard input. You can use this to pipe scores from the scoring algorithm:

```
scoring data/alarm_10000.data 1 3 | bin/gobnilp -
```

When run in this way GOBNILP will read parameter settings from the file `gobnilp.set` in the current working directory. If you are using SCIP 2.1.1 you may get the following warning:

```
[src/scip/paramset.c:2300] Warning: unknown parameter <heuristics/nlpdiving/freq>
```

which you can safely ignore. If you want GOBNILP to use a different parameter setting file called, say, `experimental.set`, you should supply this on the command line prefixed by `-g` as follows:

```
bin/gobnilp -gexperimental.set alarm100_1_3.scores
```

Changing the parameter setting file is the only command line option. All other changes to default behaviour are effected by altering parameters in the parameter setting file. If GOBNILP cannot find the parameter setting file (either the default file `gobnilp.set` or one you have selected using `-g`) then it will abort with a suitable error message.

# 4 Parameters

The behaviour of GOBNILP is affected by GOBNILP-specific parameter settings and also by SCIP parameter settings.

## 4.1 SCIP parameters

Since GOBNILP is a SCIP [1] project SCIP parameters can be altered to affect the way GOBNILP behaves. The file `gobnilp.set` in the GOBNILP distribution sets a number of SCIP parameters to non-default values. An exhaustive search for the best possible setting of SCIP parameters has *not* been conducted; the SCIP parameter settings in the supplied `gobnilp.set` file effect a general strategy of turning off all but the fastest SCIP heuristics and all SCIP cutting planes apart from Gomory cuts. Weak cutting planes (either Gomory ones or those described in [2]) are also allowed by set non-default values to certain SCIP parameters. If you find better SCIP parameter settings let us know!

By default, GOBNILP produces a lot of information about how the search is progressing due to these two parameter settings in `gobnilp.set`:

```
display/freq = 1
display/verblevel = 4
```

Reducing `display/verblevel` reduces how much information is produced, increasing `display/freq` will lead to information being printed out less often.

## 4.2 Global GOBNILP parameters

GOBNILP parameters come in three flavours: global parameters, those effecting 'plugins' in GOBNILP and those specifically related to learning pedigrees. Global parameters are listed in this section. Each parameter has three lines (description, type and setting) in `gobnilp.set` which are shown here. Apart from gobnilp/implicitfounders and gobnilp/nbns these parameters either affect what is output or allow the user to impose constraints on the structure of the BN DAG. For such parameters the following list just points you to a fuller explanation in either Section 5 or 6.

**gobnilp/dagconstraintsfile** See Section 6.

```
# file containing constraints on dag structure
# [type: string, default: ""]
gobnilp/dagconstraintsfile = ""
```

**gobnilp/implicitfounders** Let $I(W \rightarrow v)$ be the binary variable determining whether $W$ is the parent set for $v$. Normally GOBNILP uses *set partitioning* constraints $\forall v : \sum_W I(W \rightarrow v) = 1$, stating that each variable has exactly one parent set. If `gobnilp/implicitfounders` is `TRUE`, GOBNILP (effectively) replaces each variable $I(\emptyset \rightarrow v)$ with the linear expression

$1 - \sum_{W:W\neq\emptyset} I(W \to v)$ and weakens the set partitioning constraints to the *set packing* constraints: $\forall v : \sum_{W:W\neq\emptyset} I(W \to v) \leq 1$. An advantage of using set packing constraints is that SCIP knows that setting any of the variables in such a constraint to zero will never break the constraint. Note that if `gobnilp/implicitfounders` is `TRUE` the objective coefficients for the remaining $I(W \to v)$ become positive—each measuring how much better each non-empty $W$ is than the empty set as a parent set for $v$.

```
# whether to represent empty parent sets implicitly
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/implicitfounders = FALSE
```

**gobnilp/maxedges** See Section 6.

```
# maximum number of edges (-1 for no upper bound )
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/maxedges = -1
```

**gobnilp/maxfounders** See Section 6.

```
# maximum number of founders (-1 for no upper bound )
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/maxfounders = -1
```

**gobnilp/minedges** See Section 6.

```
# minimum number of edges
# [type: int, range: [0,2147483647], default: 0]
gobnilp/minedges = 0
```

**gobnilp/minfounders** See Section 6.

```
# minimum number of founders
# [type: int, range: [0,2147483647], default: 0]
gobnilp/minfounders = 0
```

**gobnilp/nbns** If set to $k$ GOBNILP will find the $k$ highest scoring BNs in descending order of score, breaking ties arbitrarily.

```
# gobnilp to find the 'nbns' best BNs ( in decreasing order of score )
# [type: int, range: [1,2147483647], default: 1]
gobnilp/nbns = 1
```

**gobnilp/noimmoralities** See Section 6.

```
# whether to disallow immoralities
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/noimmoralities = FALSE
```

**gobnilp/orderedcoveredarcs** See Section 6.

```
# whether to only allow a covered arc i<-j if i<j
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/orderedcoveredarcs = FALSE
```

**gobnilp/outputfile/adjacencymatrix** See Section 5.

```
# where the adjacency matrix representation of the BN should be output
# [type: string, default: ""]
gobnilp/outputfile/adjacencymatrix = ""
```

**gobnilp/outputfile/dot** See Section 5.

```
# where the dot representation of the BN should be output
# [type: string, default: ""]
gobnilp/outputfile/dot = ""
```

**gobnilp/outputfile/pedigree** See Section 5.

```
# where the pedigree representation of the BN should be output
# [type: string, default: ""]
gobnilp/outputfile/pedigree = ""
```

**gobnilp/outputfile/scoreandtime** See Section 5.

```
# where the score of and time to find the BN should be output
# [type: string, default: ""]
gobnilp/outputfile/scoreandtime = ""
```

**gobnilp/outputfile/solution** See Section 5.

```
# where the resulting Bayesian network should be output
# [type: string, default: "stdout"]
gobnilp/outputfile/solution = "stdout"
```

**gobnilp/printbranchingstatistics** See Section 7

```
# whether to print variable branching statistics
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
```

**gobnilp/printmecinfo** See Section 5.

```
# whether to print edges in the undirected skeleton and any immoralities
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/printmecinfo = FALSE
```

**gobnilp/printparameters** See Section 5.

```
# whether to print parameters not at default values
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/printparameters = TRUE
```

**gobnilp/printscipsol** See Section 5.

```
# whether to (additionally) print BNs in SCIP solution format
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/printscipsol = FALSE
```

**gobnilp/printstatistics** See Section 7

```
# whether to print solving statistics
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/printstatistics = FALSE
```

**gobnilp/statisticsfile** See Section 7

```
# file for statistics
# [type: string, default: ""]
gobnilp/statisticsfile = ""
```

**gobnilp/outputfile/solution** See Section 5.

```
# where the resulting Bayesian network should be output
# [type: string, default: "stdout"]
gobnilp/outputfile/solution = "stdout"
```

**gobnilp/outputfile/dot** See Section 5.

```
# where the dot representation of the BN should be output
# [type: string, default: ""]
gobnilp/outputfile/dot = ""
```

**gobnilp/outputfile/pedigree** See Section 5.

```
# where the pedigree representation of the BN should be output
# [type: string, default: ""]
gobnilp/outputfile/pedigree = ""
```

**gobnilp/outputfile/scoreandtime** See Section 5.

```
# where the score of and time to find the BN should be output
# [type: string, default: ""]
gobnilp/outputfile/scoreandtime = ""
```

## 4.3 GOBNILP plugin parameters

In the current version of GOBNILP there are two plugins: `heur_sinks.c` which provides a heuristic for finding primal solutions (i.e. BNs) and `cons_dagcluster.c` which provides a constraint that BNs are acyclic. ( The key cutting plane algorithm [2] is implemented as the separator method for this constraint. )

### 4.3.1 Sink heuristic parameters

The sink heuristic has all the parameters that any other SCIP heuristic has. For example, setting

```
heuristics/sinks/freq = -1
```

turns off this heuristic. Additionally, you have the option of getting hold of *every* primal solution (i.e. BN) proposed by this heuristic. Since this heuristic is called each time a linear relaxation is solved there are very many of these, and the same BN may be proposed many times. However, this is a convenient way of collecting a large number of reasonably high-scoring BNs. To enable this behaviour set `heuristics/sinks/printsols` to TRUE. By default all the BNs will be sent to standard output. To have them put in a file instead set `heuristics/sinks/filesols` to the name of the file. Note that the solutions are output in SCIP's internal solution format.

**heuristics/sinks/printsols**

```
# whether to print *every* BN found by sink heuristic (in SCIP solution format)
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
heuristics/sinks/printsols = FALSE
```

**heuristics/sinks/filesols**

```
# where to print solutions found by sink heuristic
# [type: string, default: ""]
heuristics/sinks/filesols = ""
```

### 4.3.2 DAG cluster constraint parameters

The dagcluster constraint handler has all the parameters that any other SCIP constraint handler has. For example, setting

```
constraints/dagcluster/sepafreq = -1
```

turns off the constraint handler's separation routine. You can also effect which cutting planes GOBNILP looks for. By default GOBNILP only looks for *1-cluster-based constraints* as introduced in [5]. By increasing the value of `constraints/dagcluster/kmax` from its default value of 1 you can ask GOBNILP to additionally look for *k-cluster-based constraints* [2] for $1 < k <$ kmax. If you just want to effect such a change just in the root node of the search tree use `constraints/dagcluster/kmaxroot`.

**constraints/dagcluster/kmax**

```
# maximum k to try for k-cluster cutting planes
# [type: int, range: [1,2147483647], default: 1]
constraints/dagcluster/kmax = 1
```

**constraints/dagcluster/kmaxroot**

```
# maximum k to try for k-cluster cutting planes in the root
# [type: int, range: [1,2147483647], default: 1]
constraints/dagcluster/kmaxroot = 1
```

## 4.4 Pedigree parameters

Pedigrees, or family trees, can be represented as Bayesian networks. GOBNILP contains additional functionality specifically for learning these. If you are not interested in this application, make sure that `gobnilp/pedigreemode` is set to FALSE and all pedigree specific behaviour will be disabled. The use of the following parameters is documented in full in Section 8.

**gobnilp/pedigreemode**

```
# whether to use GOBNILP for pedigrees
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/pedigreemode = FALSE
```

**gobnilp/sexconsistent**

> **This parameter is deprecated.**
> **Please use** `gobnilp/pedigree/sexconsistent` **instead.**

**gobnilp/pedigree/sexconsistent**

```
# whether to enforce sexual consistency in the dag
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/pedigree/sexconsistent = FALSE
```

**gobnilp/pedigree/maxsibagegap**

```
# maximum age gap permitted between full siblings (-1 for no restriction)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/pedigree/maxsibagegap = -1
```

**gobnilp/pedigree/maxhalfsibagegap**

```
# maximum age gap permitted between half siblings (-1 for no restriction)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/pedigree/maxhalfsibagegap = -1
```

**gobnilp/pedigree/maxsibsetsize**

```
# maximum number of children a pair of parents can have (-1 for no restriction)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/pedigree/maxsibsetsize = -1
```

**gobnilp/pedigree/maxchildren**

```
# maximum number of children any individual can have (-1 for no restriction)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/pedigree/maxchildren = -1
```

**gobnilp/pedigree/maxchildrenmother**

```
# maximum number of children a mother can have (-1 for no restriction)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/pedigree/maxchildrenmother = -1
```

**gobnilp/pedigree/maxchildrenfather**

```
# maximum number of children a father can have (-1 for no restriction)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/pedigree/maxchildrenfather = -1
```

**gobnilp/pedigree/agedatafile**

```
# file containing individual's age data
# [type: string, default: ""]
gobnilp/pedigree/agedatafile = ""
```

# 5 Controlling what GOBNILP outputs

In addition to allowing additional output from the sinks heuristic, increasing or reducing output from the main GOBNILP process is possible. `gobnilp/printparameters` is set to TRUE by default. Doing so ensures that any parameters set at non-default values are printed out before solving starts. If you're playing with different parameter settings this is a good way to keep track of how different parameter settings perform.

When an optimal BN is found you have the option of additionally printing out the solution in SCIP's internal format by setting `gobnilp/printscipsol` to TRUE.

Setting `gobnilp/printmecinfo` to TRUE prints out the undirected skeleton of the found BN together with any immoralities (v-structures). This determines the Markov equivalence class of the BN. Since this information is printed out in a fixed format, two Markov equivalent BNs will have exactly the same 'mecinfo' printed out. This is convenient for keeping track of which BNs are Markov equivalent to previously found BNs when `nbns` is set above 1.

GOBNILP also offers the ability to output the results of the search to screen or file in several formats. By default, GOBNILP displays the resulting BN and some score information on screen. This information can be instead redirected to a file by using the `gobnilp/outputfile/solution` parameter. This parameter's default value of `"stdout"` tells GOBNILP to show the information on the screen. If changed to `""`, it will not be shown at all and if given any other value, the information will be written to a file with that name.

There are another three similar parameters which can also be used to control what GOBNILP outputs and where.

- `gobnilp/outputfile/adjacencymatrix` is used to control whether and where to output an adjacency matrix representation of the found BN. The adjacency matrix $A$ of a directed graph has $A_{ij} = 1$ if there is an arrow from $i$ to $j$ and $A_{ij} = 0$ if there is no such arrow. GOBNILP outputs each row of the adjancecy matrix on a separate line with columns separated by a single space character. The first (top) line is the row for $i = 0$. This format has been chosen to make it convenient to read the matrices into R [8]. If `bn.mat` is the name of the file containing an adjacency matrix then

  ```
  bn <- as.matrix(read.table("bn.mat",header=FALSE))
  ```

  will create an R (adjacency) matrix `bn`. Note that R numbering starts from 1 so that `bn[1,2]` will tell you whether there is an arrow from GOBNILP variable 0 to GOBNILP variable 1.

- `gobnilp/outputfile/dot` is used to control whether and where to output a dot file that can be used by Graphviz to visualise the found BN. See `http://www.graphviz.org/` for more information on the file format and options for rendering this file.

- `gobnilp/outputfile/scoreandtime` is used to just output the score of the found BN and the time taken to find it. This can be useful when the actual BN found is not of interest, for example running experiments to compare GOBNILP to other systems or in testing during development.

- `gobnilp/outputfile/pedigree` is used to output the found BN as a pedigree. This is only of interest when using GOBNILP for finding pedigrees. See Section 8.

All the `gobnilp/outputfile/*` parameters function in the same way. If set to the empty string (`""`), they will suppress that output altogether. If set to `"stdout"`, they display the appropriate information on the screen after each optimal BN is found. If given any other value, they will attempt to write their output to a file with that name.

In the case that more than one BN is being sought, (i.e. `gobnilp/nbns` $> 1$), the filename that the `gobnilp/outputfile/` parameters try to write to will be modified to include the number of the current network. For example, if gobnilp/outputfile/solution = "output.dat", GOBNILP will write the first optimal network found to output_1.dat, the next to output_2.dat and so on.

As a full example of the use of these parameters, consider a settings file containing the following lines.

```
gobnilp/nbns = 10
gobnilp/outputfile/solution = "stdout"
gobnilp/outputfile/dot = "bn.dot"
gobnilp/outputfile/adjacencymatrix = "bn.mat"
gobnilp/outputfile/scoreandtime = "times/data"
gobnilp/outputfile/pedigree = ""
```

When run, GOBNILP will find the 10 best networks in order. After the $\mathrm{n}^{\mathrm{th}}$ BN is found, the network will be output to screen, a representation of it as a dot file will be saved to bn_n.dot for later rendering with Graphviz, an adjacency matrix representation will be save to bn_n.mat and the network's score and the time to find it will be saved to times/data_n. No pedigree information will be output to screen or file.

# 6 Effecting structural constraints

One of the advantages of the 'declarative' approach to structure learning is that some constraints on DAG structure are easy to implement. GOBNILP provides a number of global constraints on DAG structure. gobnilp/minedges and gobnilp/maxedges allow the user to express lower and upper bounds (respectively) on the number of edges in permissible BNs. Set gobnilp/maxedges to -1 to have no upper bound. Similarly, gobnilp/minfounders and gobnilp/maxfounders provide lower and upper bounds on the number of founders (vertices with no parents) in a permissible BN. Again, set gobnilp/maxfounders to -1 to have no upper bound. **Setting gobnilp/minfounders to a high value speeds up solving considerably.** So if you know (or are prepared to assume) a lower bound on the number of founders be sure to use this parameter.

Setting gobnilp/noimmoralities to TRUE rules out any BN with an immorality. Doing so will thus ensure that any learnt BN is equivalent to a decomposable undirected model. A final global constraint is gobnilp/orderedcoveredarcs: setting this to TRUE will rule out *covered* directed edges from a lower to a higher vertex. This will reduce the number of Markov equivalent BNs, which may be useful when doing repeated searches, but does not affect the score of an optimal BN (since each Markov equivalent class still has at least one representative with this constraint imposed). See, for example, [6] for further details on covered edges.

The user can also declare specific constraints on edges and immoralities by placing them in a file and setting the parameter gobnilp/dagconstraintsfile to a string which is the file's name. (If the string is empty, GOBNILP does not look for such a file.)

The file `example.constraints` in the GOBNILP distribution demonstrates the correct format for declaring constraints. Each line in the file is either a comment starting with # or a constraint. Blank lines are not permitted. `x-y` states that there must be an edge between $x$ and $y$ in the undirected skeleton. `x<-y` states that $y$ must be a parent of $x$. `x->z<-y` states that $x$ and $y$ must be unconnected parents of $z$ (an immorality). The negations of any of these constraints are possible by prefixing them with the character `~`.

# 7 Analysing the search

## 7.1 Using SCIP

The easiest way to gain insight into the search is to set `gobnilp/printstatistics` to TRUE. This will cause SCIP to output information on how often various routines are called and how long they took. You will see that, typically, most of the time is spent in the separation routine of the `dagcluster` constraint (i.e. most of the time is spent looking for cutting planes). By default, this information is sent to standard output. To send it to a file set `gobnilp/statisticsfile` appropriately.

If you set `gobnilp/printbranchingstatistics` you will get information on the search tree, in particular information on variables that were branched upon. This output by default goes to standard output, but if `gobnilp/statisticsfile` is set appropriately it goes there.

## 7.2 Profiling

For lower-level profiling of GOBNILP you need to make both SCIP and GOBNILP with `OPT=prf`. If you have CPLEX:

```
make LPS=cpx OPT=prf
```

otherwise

```
make OPT=prf
```

This will create a new executable in your bin directory and make a symbolic link to it called `gobnilp`. Running this executable on a BN learning instance will create a file called `gmon.out`. A profile of that run is then available with:

```
gprof bin/gobnilp
```

## 7.3 Using VBCTOOL

You are encouraged to download and install VBCTOOL, which is available from: `http://www.informatik.uni-koeln.de/ls_juenger/research/vbctool/`. This will allow you to see the search tree and extract useful information from nodes such as bounds and which variable was branched on. You may have to build VBCTOOL from source.

To have GOBNILP generate the required information, it is enough to set the SCIP parameter `vbc/filename` to a string specifying a filename. Just have a line like:

```
vbc/filename = "foo"
```

in your GOBNILP parameter file (which is typically `gobnilp.set`). Let's assume that you did use `"foo"` as your file name.

Once GOBNILP has stopped (and it's OK to stop it with a CTRL-C) start VBCTOOL (with no command line arguments). Use `File->Load` to load `foo`. Now do `Emulation->Start Emulation`. This will grow the search tree that GOBNILP used when running whichever BN learning problem generated `foo`. By default this will grow the tree at the same speed at which GOBNILP originally built it. If this is too slow for you, you can speed it up using the `Emulation->Setup...` menu. Setting `Seconds` to e.g. 10, will ensure that the entire simulation takes 10 seconds.

You will see that different nodes have different colours. For a full explanation of these colours consult `type_vbc.h` in the SCIP source directory. Red nodes are cut-off nodes—where GOBNILP pruned the search. Light grey (which look almost white) nodes are unsolved nodes. Basically red is good and light grey is bad.

By right-clicking on a node you get information about that node. (You may need to expand the information panel to see all of this information.) In particular you get to see the branching variable (and its branching value) that was branched on to create the node. You also get to see the dual bound (although the negative sign will be omitted).

You can print out the search tree using `File->Print`

# 8 Learning Pedigrees

Pedigrees (or family trees) are a way of recording relationships amongst a group of individuals. It is possible to represent a pedigree as a Bayesian network [7] and from this to state the problem of finding the most likely pedigree as finding a BN with the highest score. See [3] for a description of how this can be formulated as an ILP problem.

GOBNILP provides specialised support for learning pedigrees from genetic data. In order to use any of these additional functions, `gobnilp/pedigreemode` should be set to TRUE.

GOBNILPcurrently offers four forms of additional support for learning pedigrees. First, sex consistency of the pedigree can be enforced. Second, age information can be used to constrain the learning process. Third, limitations on family sizes can be specified. Fourth, output can be produced in a format convenient for interchange with other pedigree based programs. Use of each of these will be now be explained in turn.

Setting the parameter `gobnilp/pedigree/sexconsistent` to TRUE will enforce sexual consistency in the pedigree, i.e. for all found networks, it will be possible to assign sexes to the individuals in some way such that all individuals can be labelled male or female and no two individuals of the same sex have a child together. There may be several ways of labelling of a pedigree such that it is sex consistent, but if GOBNILP is used to find multiple pedigrees, only one such pedigree will be returned: each pedigree found will have some structural differences from the others, not just a relabelling of sexes.

GOBNILP does not currently allow the sexes of individuals to be given if they are known, but these should be utilised when constructing the inputs for GOBNILP. If two individuals are known to have the same sex, then they should not feature as possible co-parents in the score file.

If the ages of individuals are know, GOBNILP allows this information to be utilised to restrict the pedigree learned. Specifically, sets of siblings and sets of half-siblings can both be limited to only those with a maximum age gap between them. If `gobnilp/pedigree/maxsibagegap` is set then it with ensure that for all generated sibsets, the age difference between the youngest and oldest sibling does not exceed the age gap given. `gobnilp/pedigree/maxhalfsibagegap` works similarly except for enforcing the constraint on sets of half-siblings who have a common mother. In order to use these functions, age information must first be supplied. This is done through the parameter `gobnilp/pedigree/agedatafile`, which gives the file from which age data should be read. This file should have age information for one individual per line, given as name and age separated by a space. Ages should be given as integers.

Other uses of age information, such as preventing children being older than parents or individuals having children before reaching sexual maturity, should be enforced outsiede GOBNILP by only providing parentage scores as input when that parentage is possible.

In addition to limiting sibsets by age difference, GOBNILP also allows them to be limited by size. `gobnilp/pedigree/maxsibsetsize` can be used to limit the maximum number of children that a pair of individuals have together, i.e. the maximum number of full siblings in a family. It is also possible to limit the number of children that any one individual can have with all its mates. This is specified using `gobnilp/pedigree/maxchildren`. As male and female mating reproductive behaviour may vary considerably, the maximum number of children allowed ca be specified independently for males and females. The parameters `gobnilp/pedigree/maxchildrenfather` and `gobnilp/pedigree/maxchildrenmother` allow these values to be given respectively. In the case that both a limit on the number of children for all individuals and a limit specific to a sex are given, the lower of the two numbers is enforced.

Output of the found BN from GOBNILP can be in a convenient pedigree format. This is controlled through the `gobnilp/outputfile/pedigree` parameter, see section 5 for a description of how to use this parameter. The output pedigree has one line per individual with each line having four tab separated fields. The first field is the individual that the line refers to. The second field is the sex of the individual. If `gobnilp/pedigree/sexconsistent = FALSE` in the settings file, then all individuals will have a sex of U for unknown, otherwise this will be either M (male) or F (female). Finally, the last two fields record the father and mother of the individual respectively. All nodes labelled as male can only appear in the father field and similarly all females can only appear as mothers. If a node has one or more parents who are not in the sample, these are shown as a "–" in the appropriate field. The pedigree is always arranged such that the row stating a node's parents always appears before all rows in which that node appears as another individual's parent.

# 9 Debugging

## 9.1 Compiling in debug mode

To create a GOBNILP executable that runs in debug mode, just 'make' SCIP and then GOBNILP with `OPT=dbg`:

```
make LPS=cpx OPT=dbg
```

or (if without CPLEX)

```
make OPT=dbg
```

After making GOBNILP in this way there will be a symbolic link from an executable whose name contains ".dbg." to `gobnilp`. Just run this GOBNILP as normal. Execution will terminate with a suitable error message if an assert error is generated.

## 9.2 Using gdb

You can use `gdb` to track down bugs. To do this you need to compile with the `-g` flag set. The simple way to do this is just to edit the Makefile. Replacing this:

```
$(OBJDIR)/%.o: $(SRCDIR)/%.c
      @echo "-> compiling $@"
      $(CC) $(FLAGS) $(OFLAGS) $(BINOFLAGS) $(CFLAGS) -c  $< $(CC_o)$@
```

with this

```
$(OBJDIR)/%.o: $(SRCDIR)/%.c
      @echo "-> compiling $@"
      $(CC) $(FLAGS) $(OFLAGS) $(BINOFLAGS) $(CFLAGS) -c -g $< $(CC_o)$@
```

Here is an abbreviated session using `gdb` to track down a bug. Text after `(gdb)` is user input. Some newlines have been added for readability.

```
pc435h ~/research/gobnilp gdb
GNU gdb (GDB) 7.2
...
```

```
(gdb) file bin/gobnilp
Reading symbols from /n/staff/jc/research/gobnilp/bin/gobnilp...done.
(gdb) run data/asia_100_pascores_3._filtered.pck.mon
Starting program:
/n/staff/jc/research/gobnilp/bin/gobnilp data/asia_100_pascores_3._filtered.pck.mon
[Thread debugging using libthread_db enabled]
Solving the BN structure learning problem using SCIP.
....
SCIP Status        : problem is solved [optimal solution found]
Solving Time (sec) : 1.43
Solving Nodes      : 1
Primal Bound       : -2.45644265388390e+02 (7 solutions)
Dual Bound         : -2.45644265388390e+02
Gap                : 0.00 %
0<-5,6, -21.675646
1<- -71.525263
2<-0,5, -2.247729
3<-2, -16.935375
4<- -12.354096
5<-4, -2.737050
6<-1, -65.918644
7<-1, -52.250461
BN score is -245.644265
gobnilp: src/scip/primal.c:143:
 SCIPprimalFree: Assertion `(*primal)->nexistingsols == 0' failed.

Program received signal SIGABRT, Aborted.
0x00007ffff679c035 in raise () from /lib64/libc.so.6
(gdb) backtrace
#0  0x00007ffff679c035 in raise () from /lib64/libc.so.6
#1  0x00007ffff679d9e6 in abort () from /lib64/libc.so.6
#2  0x00007ffff67948e5 in __assert_fail () from /lib64/libc.so.6
#3  0x000000000059dc81 in SCIPprimalFree (primal=0x140d080, blkmem=0x140f1b0) at src/scip/primal.c
#4  0x00000000005c43cf in freeTransform (scip=0x140d010) at src/scip/scip.c:7302
#5  0x00000000005c5ce7 in SCIPfreeTransform (scip=0x140d010) at src/scip/scip.c:7764
#6  0x0000000000407303 in main (argc=2, argv=0x7fffffffde98) at src/gobnilp.c:279
(gdb) q
```

Note that the GOBNILP executable here must have been compiled using `OPT=dbg` since an assert error has been raised. gdb can be used with any sort of GOBNILP executable.

# 10 Citing GOBNILP

When citing GOBNILP please use [2] (GOBNILP is an improved version of the software used to create the results in that paper.) You should also cite SCIP. See `http://scip.zib.de/licence.shtml` for how to do this. It would also be odd not to cite [5]. The use of ILP for pedigree learning is introduced in [3], which should also be cited if you use GOBNILP for this purpose.

## Acknowledgements

- GOBNILP version 1.2 and higher is supported by MRC Project Grant G1002312.
- Many thanks are due to the SCIP developers for writing and distributing SCIP and also helping with queries. Particular thanks are due to: Tobias Achterberg, Timo Berthold, Ambros Gleixner, Gerald Gamrath, Stefan Heinz, Michael Winkler and Kati Wolter.
- Marc Pfetsch wrote the `cons_linearordering.c` constraint handler which provided a useful 'template' which helped JC write `cons_dagcluster.c`.
- The most important part of the separation routine in `cons_dagcluster.c` uses the 'cluster constraint' introduced in [5]. Additional thanks to Tommi Jaakkola and David Sontag for providing data and encouragement-to-distribute, respectively.


# A Correctness of pruning in `scoring.c`

The GOBNILP BDeu scoring code `scoring.c` uses a slightly modified version of the pruning approach devised by de Campos and Ji [4]. Search for "de Campos and Ji" in the source. The slight modification is that there is no check on the value of the *effective sample size*. This section provides a justification for this modification (and an alternative BDeu-specific proof that the de Campos and Ji pruning is valid).

**Lemma 1.** *Let $n_{ij}$ be a positive integer and $\alpha'$ be a positive real. Then*

$$\log \frac{\Gamma(n_{ij}+\alpha')}{\Gamma(\alpha')} = \sum_{i=0}^{n_{ij}-1} \log(i+\alpha') \tag{1}$$

*Proof.* Consider the identity $\Gamma(x+1) \equiv x\Gamma(x)$. From this identity it follows that $\Gamma(n_{ij}+\alpha') = (n_{ij}+\alpha'-1)\Gamma(n_{ij}+\alpha'-1)$. The result follows by repeated application of this identity. □

**Lemma 2.** *Let $\{n_{ijk}\}_{k=1,\dots r_i}$ be non-negative integers with a positive sum $n_{ij} = \sum_{k=1}^{r_i} n_{ijk}$. Let $\alpha''$ be a positive real. Then*

$$\sum_{k=1}^{r_i} \log \frac{\Gamma(n_{ijk}+\alpha'')}{\Gamma(\alpha'')} \leq \log \frac{\Gamma(n_{ij}+\alpha'')}{\Gamma(\alpha'')} \tag{2}$$

*Proof.* With the sum $n_{ij} = \sum_{k=1}^{r_i} n_{ijk}$ fixed consider distributions of counts $\{n_{ijk}\}_{k=1,\dots r_i}$ over the $r_i$ 'cells'. If $n_{ijk^*} = n_{ij}$ for some $k^*$ then $n_{ijk} = 0$ for all other values of $k$ and it is clear that $\sum_{k=1}^{r_i} \log \frac{\Gamma(n_{ijk}+\alpha'')}{\Gamma(\alpha'')} = \log \frac{\Gamma(n_{ij}+\alpha'')}{\Gamma(\alpha'')}$. Suppose now that there are two indices $k_1$ and $k_2$ such that $n_{ijk_1} > 0$ and

$n_{ijk_2} > 0$, and suppose, wlog, that $n_{ijk_1} \geq n_{ijk_2}$. Consider moving one count from cell $k_1$ to cell $k_2$, that is: increasing $n_{ijk_1}$ by one and decreasing $n_{ijk_2}$ by one (so that $n_{ij}$ remains constant). By (1) the increase in the LHS of (2) is $\log(n_{ijk_1}+\alpha'')-\log(n_{ijk_2}-1+\alpha'')$. Since $n_{ijk_1} \geq n_{ijk_2}$ this is a positive increase. By increasing big counts at the expense of small counts in this way a sequence of distributions of the fixed sum $n_{ij}$ over the $r_i$ cells can be constructed for which the LHS of (2) is increasing. The sequence terminates when $n_{ijk^*} = n_{ij}$ for some $k^*$. The result follows. □

**Theorem 1.**

$$\sum_{j=1}^{q_i} \log \frac{\Gamma(\alpha')}{\Gamma(n_{ij}+\alpha')} + \sum_{k=1}^{r_i} \log \frac{\Gamma(n_{ijk}+\frac{\alpha'}{r_i})}{\Gamma(\frac{\alpha'}{r_i})} \leq \sum_{j:n_{ij}>0} \sum_{i=0}^{n_{ij}-1} \log \left( \frac{i+\alpha'/r_i}{i+\alpha'} \right) \tag{3}$$

*Proof.*

$$\begin{aligned} & \sum_{j=1}^{q_i} \log \frac{\Gamma(\alpha')}{\Gamma(n_{ij}+\alpha')} + \sum_{k=1}^{r_i} \log \frac{\Gamma(n_{ijk}+\frac{\alpha'}{r_i})}{\Gamma(\frac{\alpha'}{r_i})} \\ \leq & \sum_{j=1}^{q_i} \log \left( \frac{\Gamma(\alpha')}{\Gamma(n_{ij}+\alpha')} \frac{\Gamma(n_{ij}+\alpha'/r_i)}{\Gamma(\alpha'/r_i)} \right) \qquad \text{by (2)} \\ = & \sum_{j=1}^{q_i} \sum_{i=0}^{n_{ij}-1} \log \left( \frac{i+\alpha'/r_i}{i+\alpha'} \right) \qquad \text{by (1)} \\ = & \sum_{j:n_{ij}>0} \sum_{i=0}^{n_{ij}-1} \log \left( \frac{i+\alpha'/r_i}{i+\alpha'} \right) \end{aligned}$$

□

**Corollary 1.** *Let $r_i > 1$ and define $q^{(+)} \equiv |\{j : n_{ij} > 0\}|$ then*

$$\sum_{j=1}^{q_i} \log \frac{\Gamma(\alpha')}{\Gamma(n_{ij}+\alpha')} + \sum_{k=1}^{r_i} \log \frac{\Gamma(n_{ijk}+\frac{\alpha'}{r_i})}{\Gamma(\frac{\alpha'}{r_i})} \leq -q^{(+)} \log r_i \tag{4}$$

*Proof.* If $n_{ij} > 0$ then

$$\begin{aligned} \sum_{i=0}^{n_{ij}-1} \log \left( \frac{i+\alpha'/r_i}{i+\alpha'} \right) &= -\log r_i + \sum_{i=1}^{n_{ij}-1} \log \left( \frac{i+\alpha'/r_i}{i+\alpha'} \right) \\ &\leq -\log r_i \end{aligned}$$

The inequality holds because each term in the sum on the RHS is negative (since $r_i > 1$). The result then follows from Theorem 1. □

**Corollary 2.** *Consider a fixed child variable $i$ with $r_i$ values. Let $W$ and $W'$ be distinct candidate parent sets for $i$ such that $W \subset W'$. Let $c(i, W)$ the the BDeu local score for $W$ as the parent set of $i$ (for some fixed ESS $\alpha$). Let $q^{(+)}(W')$ be the number of positive counts in the contingency table for $W'$. If $c(i, W) > -q^{(+)}(W') \log r_i$ then neither $W'$ nor any of the supersets of $W'$ can be an optimal parent set for $i$.*

*Proof.* The LHS of (4) is the BDeu local score for a parent set $W'$ which has $q_i$ counts $n_{ij}$ in its contingency table and counts $n_{ijk}$ in the contingency table for $W' \cup \{i\}$ and where $\alpha' = \alpha/q_i$. If $-q^{(+)}(W') \log r_i < c(i, W)$ then $c(i, W)$ is certainly higher than the local BDeu score for $W'$ due to (4). If $W' \subset W''$ then $q^{(+)}(W'') \geq q^{(+)}(W')$ and so $-q^{(+)}(W'') \log r_i \leq -q^{(+)}(W') \log r_i$. From this it follows that the local score for $W''$ must also be less than $c(i, W)$. Consider a BN where $W'$ or one of its supersets is a parent set for $i$. A BN with a higher score can be obtained by replacing that parent set with $W$. The result follows. $\square$

# References


[1] Tobias Achterberg. *Constraint Integer Programming.* PhD thesis, TU Berlin, July 2007.

[2] 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. AUAI Press, 2011.

[3] James Cussens, Mark Bartlett, Elinor M. Jones, and Nuala A. Sheehan. Maximum likelihood pedigree reconstruction using integer linear programming. *Genetic Epidemiology*, 2012. To Appear.

[4] Cassio de Campos and Qiang Ji. Properties of Bayesian Dirichlet scores to learn Bayesian network structures. In *AAAI-10*, pages 431–436, 2010.

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

[6] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques.* MIT Press, 2009.

[7] Steffen L. Lauritzen and Nuala A. Sheehan. Graphical models for genetic analyses. *Statistical Science*, 18(4):489–514, 2003.

[8] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.