GOBNILP 1.6.2 User/Developer Manual[1]

James Cussens, Mark Bartlett
University of York

July 2, 2015

[1]GOBNILP version 1.3-1.6 was supported by MRC Project Grant G1002312

Contents

Chapter 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 1.6.2 has only been tested using SCIP 3.2.0 (the current SCIP version at time of writing) so please use that version. We expect that it will work with future versions of SCIP.

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. Ensure `$(GOBNILPDIR)` is your current working directory and run `./configure.sh $(SCIPDIR)` where `$(SCIPDIR)` is the directory where you installed SCIP. For example, `./configure.sh /opt/scipoptsuite-3.2.0/scip-3.2.0` . This will create a soft link `scip` in `$(GOBNILPDIR)` that links to `$(SCIPDIR)`, which is needed by the GOBNILP make file to find the SCIP files. This script will also copy `$(SCIPDIR)/examples/LOP/src/cons_linearordering.c` and `$(SCIPDIR)/examples/LOP/src/cons_linearordering.h` into `$(GOBNILPDIR)/src`.

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

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

7. If you have doxygen installed you can use it to create HTML documentation for the GOBNILP source code. To do this do `make docs` and then point your web browser at `$(GOBNILPDIR)/docs/html/index.html`. You can edit the file `Doxyfile` in the GOBNILP distribution to alter the sort of HTML documentation you get.

8. If you have any problems installing you may need to run `make clean` which will remove everything the other make targets create.

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.

Chapter 2

Quick start

GOBNILP can learn Bayesian networks from complete discrete data. The simplest way to do this is to give the name of the file containing the data as a command line argument:

```
bin/gobnilp data/asia_100.dat
```

To run GOBNILP in this way it is essential that the filename has a `.dat` extension and that the data is in the right format. An example data file `asia_100.dat` is provided with the GOBNILP distribution (in the `data` directory). Note that it is essential that there is no trailing whitespace at the end of lines in the data file. Details on the correct data file format are given in Section 3.4.1.

By default GOBNILP searches for the BN with the highest BDeu score (log marginal likelihood) **subject to no node having more than 3 parents**. This limit on the number of parents can be raised (or lowered); see Section 3.5.1 for details. By default the BDeu score is computed using an *effective sample size* of 1. This too can be altered.

Alternatively, GOBNILP can learn BNs from cached (typically pruned) 'local scores'. Local score input is the default so unless you indicate otherwise (by, for example, using a `.dat` filename extension) GOBNILP will assume the input is of this sort. So, for example, if the filename extension is `.scores` GOBNILP will assume it is a file of local scores in the right format. So you can do:

```
bin/gobnilp data/asia_100_1_3.scores
```

An example local score file `asia_100_1_3.scores` is provided with the GOBNILP distribution (in the `data` directory). Details on the correct local scores file format are given in Section 3.4.2.

By default, GOBNILP prints out the learned BN structure with one line for each node specifying its parents and the local score for that choice of parents. Other output formats are available as described in Section 3.3.

Chapter 3

Basic usage

3.1 Running GOBNILP

GOBNILP is run from the command line and takes a number of optional arguments followed by the name of a file containing the information about the variables for which the Bayesian network is to be found. For example:

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

runs GOBNILP using the local scores found in `alarm100_1_3.scores`. If the filename is given as `-` then GOBNILP will from standard input.

GOBNILP currently recognises five command line arguments.

`-g=`*filename*	Sets the file from which parameters are read (see Section 3.2).
`-f=`*format*	Sets the input file format (see Section 3.4).
`-x`	Create the model and exit before solving it (see Chapter 6).
`-q=`*filename*	Sets the frequency file for pedigree learning (see Chapter 7).
`-v=`*level*	Set the verbosity level of the output.

Verbosity can be set to any value between 0 and 5. Levels 0 and 1 just display error and warning messages. Level 2 gives additional information about the system being run and file output. Level 3 is the default and gives basic information about the progress of the search. Level 4 gives more detailed information about the progress of the search, while level 5 also displays detailed statistics about the search once the BN has been found.

3.2 Settings

Changes to GOBNILP's behaviour are affected using a settings file. A settings file consists of a set of parameters and values, each on a line of its own. Lines beginning with a `#` are treated as comments and blank lines are also permitted.

Each parameter setting is of the form `parameter = value` where `parameter` is the name of the parameter and `value` is a permitted value for that parameter. Parameters which take a string value should have their value given in quotation marks. For example, the following sets the `gobnilp/outputfile/solution` parameter to `output.txt`.

```
gobnilp/outputfile/solution = "output.txt"
```

A full list of GOBNILP parameters is given in Chapter 9. The settings files can also be used to alter any of SCIP's parameters.

By default, GOBNILP tries to load parameters from the file `gobnilp.set` in the current working directory. If it cannot find this file, it will run using default settings. An alternative setting file can be specified using the `-g` command line argument. For example, to run GOBNILP on `data/asia_100.dat` with settings stored in the file `mysetttings.txt`, the following command should be given

```
bin/gobnilp -g=mysettings.txt data/asia_100.dat
```

3.3 GOBNILP output formats

GOBNILP can output the learned BN structure 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. This parameter, like all other GOBNILP parameters is set by adding a line to GOBNILP's parameter file which, by default, is called `gobnilp.set` and is in the current working directory. So for example, adding the following line (anywhere) to `gobnilp.set` will put the learned BN in the file `foo.bn`:

```
gobnilp/outputfile/solution = "foo.bn"
```

There now follows a list of parameters which can be used to control what GOBNILP outputs and where. As always they are set by adding a suitable line to the file `gobnilp.set`.

- `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 adjacency 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 [10]. 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`.

- `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/mec` is used to print 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 info printed out.
- `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 Chapter 7.

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. Any occurrence of `<probname>` will be replaced by the name of the current problem.

In the case that more than one BN is being sought, (i.e. `gobnilp/nbns` > 1, see Chapter 9), 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.bn"`, GOBNILP will write the first optimal network found to `output_1.bn`, the next to `output_2.bn` 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 = ""
gobnilp/outputfile/mec = "mec/<probname>.out"
```

When run, GOBNILP will find the 10 best networks in order. After the n^{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. The Markov equivalence class will be output to a file that depends on the file used as input; if the input file was `scores.txt`, the output will be to `mec/scores_n.out`.

3.4 GOBNILP input formats

In this section data and local score input formats are described. GOBNILP can also accept input in SCIP's CIP format. See Section 4.5 for details on that. Input for pedigree learning is discussed in Section 7.

3.4.1 Data files

GOBNILP can read in a set of observations of variables and search for the BN with the highest BDeu score. To specify that this type of input is being given, GOBNILP should either be run using `-f=dat` as a command line argument or the file containing the data should be given a `.dat` filename extension. For example:

```
gobnilp -f=dat observations.txt
```

or

```
gobnilp observations.dat
```

Data file format

The data file consists of a set of observations of the variables. Let the number of variables be p. Each line of the file representing data should contain p values with a separator between them. Each such line represents a single datapoint. The ith item on each line is the value of variable i for that datapoint. Values can take any type, e.g. integers, strings, Booleans. Any line starting with `#` is viewed as a comment and ignored.

In addition to the observations, variable names may also be given to make the output more intelligible. Variable names are given on the first non-comment line of the file. These should be given in the same format as the observations: on a single line with separators between them. If no variable names are given, names will be automatically generated according to the column number with the variable name for the first column being '0' and the variable name for the last column being '$p-1$'.

Finally, the arities of the variables may also be given. This should be done on the first non-comment line after the variable names, or the first non-comment line in the file if no variable names are given. The arities should state how

many values each variable can take and follow the same format as the names and observations, except that values must be integers. If arities are not given, GOBNILP assumes that the arity of a variable is just the number of distinct observed values for that variable. **Note that inferred arities may lead to incorrect BDeu scores if some possible values are never observed in the dataset.** We recommend explicitly stating arities.

Since variable names and arities are optional in data files (and it is not always possible for GOBNILP to distinguish between names, arities and datapoints) the parameters `gobnilp/scoring/names` and `gobnilp/scoring/arities` are used to specify whether these are provided in the file (the default is that both names and arities are provided). If you are getting errors when reading data into GOBNILP check that these parameters are set correctly.

An example of the first few lines of a file for a dataset with 8 variables might look as follows:

```
# Data from experiment 1
AAA T2  S       L B   X   D  E
2   2   2       7 12  2   26 26
yes no  absent  3 foo bar x  y
yes yes present 4 bar foo z  x
no  no  absent  0 bar bar d  y
```

If the input given to GOBNILP is in the form of data, then a scorer is used to compute the local BDeu scores. There are a number of parameters that can be set to affect the scorer, primarily its input parser.

The basic format of the data input file can be set using the `gobnilp/delimiter` parameter, which specifies the separator between values in the file. This can be any character, `tab` or `whitespace` (which matches both the tab character and the space character). `gobnilp/mergedelimiters` can be used to say whether consecutive delimiters should be merged into a single separator. For example, the settings for white space delimited files would be:

```
gobnilp/delimiter = "whitespace"
gobnilp/mergedelimiters = TRUE
```

whereas CSV files would require these parameters to be set to:

```
gobnilp/delimiter = ","
gobnilp/mergedelimiters = FALSE
```

3.4.2 Score files

GOBNILP can read in a set of local scores and search for the BN with the highest overall score. Note that GOBNILP does not know or care what sort of local scores these are; they could be BIC, BDeu or some other sort of local score. To specify that this type of input is being given, GOBNILP should either be run using `-f=jkl` as a command line argument or the file containing the data should be given a `.jkl` filename extension. For example:

```
gobnilp -f=jkl scores.txt
```

or

```
gobnilp scores.jkl
```

Although this is the cleanest way of specifying local score input, this sort of input is the GOBNILP default, so if GOBNILP cannot deduce that the input is not local scores it will just assume that it is.

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 Jaakkola et al.[6]. 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.

3.4.3 How the expected input format is determined

The expected format of the input file it is determined using the following rules.

1. If the `-f` command line argument is given, the stated format will be used. For example:

   ```
   gobnilp -f=dat data.txt
   ```

will cause GOBNILP to attempt to read the file `data.txt` as a data file. The recognised values for `-f` are: `jkl` for local scores, `dat` for data and `cip` for CIP format.

2. If `-f` is not given, then the extension of the file will be used. For example:

   ```
   gobnilp data.cip
   ```

   will cause GOBNILP to attempt to read the file `data.cip` as a SCIP CIP file (see Section 4.5). The recognised extensions are: `jkl` for local scores, `dat` for data and `cip` for CIP format.

3. If the `-f` flag is not present and the file extension does not match any known type, GOBNILP assumes that it is being given local scores and attempts to read the file as such. For example:

   ```
   gobnilp data.txt
   ```

   will cause GOBNILP to attempt to read the file `data.txt` as a local score file.

3.4.4 Sensitivity to precision

GOBNILP's performance is sensitive to the precision for local scores. Removing a few decimal places can cause large variations in solving time (and of course the score of the optimal BN!). For this reason learning directly from local scores can give different behaviour to learning from data, even when the 'same' local scores are being produced internally by the scoring algorithm.

3.5 Effecting structural constraints

3.5.1 Simple global structural constraints

The most important constraint is the limit on the size of parent sets. The default value of 3 can be altered by setting the `gobnilp/scoring/palim` parameter to the desired limit. If set to `-1`, parent sets of all sizes will be considered. Setting `gobnilp/scoring/palim` to a high value will typically lead to very slow solving and/or a crash due to the generation of many candidate parent sets unless there are few (e.g. ≈ 12) variables in the data. However, this is not always the case, so it is best to experiment: setting `gobnilp/scoring/palim` to your preferred value to begin with and then compromising with a lower value if that proves to be necessary. (When learning from a file of local scores `gobnilp/scoring/palim` has no effect, since the set of all potential parent sets has already been decided.)

One of the advantages of the 'declarative' approach to structure learning is that many constraints on DAG structure are easy to implement. GOBNILP provides a number of simple 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, however if you want to learn a decomposable model, and your score function is likelihood-equivalent, then it is better to set `gobnilp/decomposable` to TRUE. Doing so rules out immoralities and also (admissibly) fixes an arbitrary variable to be an orphan, thus providing a modest speed-up.

A related global constraint is `gobnilp/orderedcoveredarcs`: setting this to TRUE will rule out *covered* directed edges from a lower indexed to a higher indexed vertex (when learning from data a vertex's index is given by its column in the data, not its name). 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, [7] for further details on covered edges.

It is also possible to put bounds on the number of parent vertices in the BN. A vertex is a parent if it has at least one child. Use `gobnilp/minparents` to set a lower bound and `gobnilp/maxparents` for an upper bound, where as usual, a value of -1 declares that there is no upper bound (the default). Unlike other simple global constraints setting either `gobnilp/minparents` or `gobnilp/maxparents` to non-default values creates auxiliary variables internally: there is an indicator variable for each BN vertex being a parent. This means that the 'sink' heuristic will not work unless it is allowed to use 'probing'. See Section 4.2.2 for further details.

3.5.2 Simple local structural constraints

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 these local constraints. The file `example.constraints` will work with the datafile `data/asia_100.dat` (note that some of constraints have been commented out, they are just there so you can see the format). 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 `~`.

3.5.3 Arbitrary conditional independence constraints

GOBNILP allows the user to put arbitrary conditional independence constraints on DAGs. This is done by adding such constraints to the file specified by `gobnilp/dagconstraintsfile`. The file `example.constraints` in the GOBNILP distribution demonstrates the correct format for declaring these constraints. This format is either:

```
<<A>> _|_ <<B>>
```

for independence constraints or

```
<<A>> _|_ <<B>> | <<S>>
```

for conditional independence constraints, where `<<A>>`, `<<B>>` and `<<S>>` are comma-separated lists of variable names (or variable numbers if variable names are not being used). Note that (conditional) independence constraints cannot currently be negated.

Unlike earlier versions, GOBNILP1.6 has a separator (cutting plane generator) for conditional independence constraints. Let V be the set of all BN variables. If $A \perp B|S$, $a \in A$, $b \in B$ and $\{a, b\} \subseteq C \subseteq V \setminus S$ then there must be at least two elements of C with no parents in C, since otherwise a and b are d-connected. GOBNILP searches for (useful) linear inequalities representing this observation. It does so by adapting the separator for ruling out cycles.

3.5.4 Arbitrary SCIP constraints

As described in Section 4.5 GOBNILP can read a description of a BN learning problem using SCIP's CIP format. Typically you would use GOBNILP to generate such a file from data or local scores. It is possible to edit such a file to add in additional constraints. Any constraint understood by SCIP is OK including, of course, linear constraints. See the SCIP documentation for a list of available constraints and how they are represented in the CIP format.

Note that it is also possible to use the CIP format to define extra variables with whatever objective coefficient you like. This would be an easy way to add prior terms to the objective function.

3.6 Vanilla BN learning

Since the user may choose to put all sorts of constraints on the learned Bayesian network, GOBNILP (unlike some other BN learning systems) cannot normally take advantage of certain properties of unconstrained optimal Bayesian networks. For example, with no constraints it is the case that in an optimal BN,

for each node the highest scoring parent set will be selected which is consistent with a topological ordering of the nodes.

To allow GOBNILP to take advantage of such facts, the user can choose to set the parameter `gobnilp/vanilla` to `TRUE`. (Doing this when there *are* constraints is a mistake, you may not get an optimal network.) If `gobnilp/vanilla = TRUE` the following happens:

1. For each node, if none of the parents in its 'best' available parent set can be descendants of the node (all are *allowed*), then this parent set is selected.

2. For each node, if a parent set differs from a 'better' one only by having *allowed* parents removed, then that parent set is ruled out.

Although `gobnilp/vanilla = TRUE` allows propagations that are normally not allowed, we have not observed any performance benefit on our test instances—but feel free to see if this setting helps on your problem.

Chapter 4

Advanced usage

In this chapter more advanced usage of GOBNILP is considered. This is done by setting GOBNILP or SCIP parameters to non-default values.

4.1 Controlling the AD tree and caching

In GOBNILP1.6 an input dataset is internally stored in an AD tree using almost exactly the approach given by Moore and Lee [9]. GOBNILP's default parameter settings for the AD tree are an attempt to guard against running out of memory (by building too big a tree) and delivering fast production of BDeu 'local scores'. GOBNILP also caches marginal likelihood values for subsets of variables. Here are the parameters related to the AD tree and to caching:

```
# minimum number of datapoints to create a branch in the AD tree
# [type: int, range: [0,2147483647], default: 2000]
gobnilp/scoring/rmin = 2000

# limit on the depth of the AD tree
# [type: int, range: [0,2147483647], default: 1000]
gobnilp/scoring/adtreedepthlim = 1000

# limit on the number of nodes in  the AD tree
# [type: int, range: [0,2147483647], default: 10000]
gobnilp/scoring/adtreenodeslim = 10000

# limit on number of scores that are cached
# [type: int, range: [0,2147483647], default: 10000000]
gobnilp/scoring/cachesizelimit = 10000000

# how much to increase the cache when needed and allowed
# [type: int, range: [0,2147483647], default: 10000]
gobnilp/scoring/cacheblocksize = 10000
```

```
# subsets must have size below this to be cached
# [type: int, range: [0,2147483647], default: 50]
gobnilp/scoring/nvarscachelimit = 50
```

If you have a machine with lots of memory then you may wish to alter the values of some of these parameters—in particular reducing `gobnilp/scoring/rmin` will typically improve performance. If you are running out of memory then moving values in the other direction will help you.

4.2 Altering the search strategy

4.2.1 Altering SCIP parameter values

Since GOBNILP is a SCIP [1] project SCIP parameters can be altered to affect the way GOBNILP behaves. By default, GOBNILP sets a number of SCIP parameters to non-default values, though these can be over-ridden by use of a settings file. An exhaustive search for the best possible setting of SCIP parameters has *not* been conducted; the SCIP parameter settings used by default by GOBNILP effect a general strategy of turning off all but the fastest SCIP heuristics and most SCIP cutting plane algorithms. If you find better SCIP parameter settings let us know!

SCIP gives you much control over the search: you can switch from SCIP's default search strategy to best-first search or depth-first for example. You can decide how many cuts to add, how often to look for cuts and how efficacious a cut must be to be added. You can choose to run any number of SCIP's builtin *primal heuristics* to look for solutions. Read the SCIP documentation for further details.

4.2.2 Controlling the sink-finding primal heuristic

The sink-finding primal heuristic uses the current LP solution to search for a high-scoring DAG. Finding such DAGs should speed up solving times and provides better anytime behaviour. This heuristic (described in [2]) has all the parameters that any other SCIP heuristic has. For example, setting

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

turns off this heuristic. The heuristic has two modes: non-probing (the default) and probing. Probing mode does a bit more work to set any auxiliary variables to (hopefully) feasible values. So if you are using any auxiliary variables, either explicitly or implicitly since you are putting bounds on the number of parents (see Section 3.5.1), you should turn probing mode on. Do this by adding the following line to your parameter file (usually `gobnilp.set`)

```
heuristics/sinks/probing = TRUE
```

By default probing mode does a search of maximal depth 100. This can be altered using the parameter `heuristics/sinks/maxdivedepth`.

The heuristic typically makes a number of tie-breaking random choices. You can set the random seed for such choices by setting `heuristics/sinks/seed`. Choosing different random seeds is an easy way to measure how sensitive the performance of GOBNILP is to such arbitrary decisions. The parameter `heuristics/sinks/nruns` controls how often the heuristic runs for each LP solution. Due to the just mentioned random tie-breaking choices, one may find better heuristic solutions by raising this parameter from its default value of 1.

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 = ""
```

4.2.3 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 [6]. 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* [3] 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
```

The dagcluster constraint has very many other parameters. These will be described in the next version of this manual.

4.3 Choosing a BDeu score

The BDeu score depends on a single parameter, the *effective sample size*. By default, this is set to 1, though it can be changed with a parameter as explained below. 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.)

By default GOBNILP uses C's `lgamma` function to compute BDeu scores. Values such as $\log\Gamma(\alpha+n)-\log\Gamma(\alpha)$ are computed, where α is the effective sample size and n is some count coming from the data. For very large values of α the `lgamma` function has numerical problems. For example, `lgamma(1e20+1) - lgamma(1e20)` will be computed as 0. To avoid this problem GOBNILP can compute $\log\Gamma(\alpha+n)-\log\Gamma(\alpha)$ as the sum $\sum_{i=0}^{n-1}\log(\alpha+i)$ which is more numerically stable, but slower to compute. To use this safer option set `gobnilp/scoring/fast` to `FALSE`. This is only necessary for very large values of α.

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 saved. **This means that parentset-child combinations for a second-best BN may be missing.** The algorithm uses a slightly modified version of the pruning approach devised by de Campos and Ji [5]. See Appendix A for a proof of the correctness of the implemented pruning method. If you need scores for parentset-child combinations which will never appear in the most likely BN, (for example, if you are computing the k best BNs) then pruning can be turned off using the appropriate parameter. Note that if you are learning a decomposable model from data then pruning should be turned off since such a constraint makes GOBNILP's pruning algorithm inadmissible. (All optimal decomposable models may be pruned away.)

There now follows a list of parameters affecting scoring. We have added parameters which concern the format of the data file from which scores would be generated.

gobnilp/mergedelimiters

```
# whether to treat consecutive delimitors in the input file as one
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/mergedelimiters = TRUE
```

gobnilp/delimiter

```
# the delimiter for fields in the input file (special values - whitespace, tab)
# [type: string, default: "whitespace"]
gobnilp/delimiter = "whitespace"
```

gobnilp/scoring/alpha

```
# alpha value for use in BDeu scoring
# [type: real, range: [0,3.4E38], default: 1]
gobnilp/scoring/alpha = 1
```

gobnilp/scoring/arities

```
# whether variable arities are given in the data file
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/scoring/arities = TRUE
```

gobnilp/scoring/names

```
# whether variable names are given in the data file
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/scoring/names = TRUE
```

gobnilp/scoring/palim

```
# maximum number of parents for each node (-1 for no limit)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/scoring/palim = -1
```

gobnilp/scoring/prune

```
# whether to prune during scoring
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/scoring/prune = TRUE
```

4.4 Collecting and analysing information on the search

By default, GOBNILP produces little information about how the search is progressing beyond the score of the current best solution and the upper bound on the best possible solution that may exist. Additional information can be obtained by changing the verbosity level of the program, either through the `-v` commandline argument or the SCIP parameter `display/verblevel`.

Setting verbosity to 4 will produce significantly more information about the progress of the search, such as the number of variables and rows active at the current stage in the solving. Please refer to the SCIP documentation for full information on the output produced.

Even greater information can be obtained by setting verbosity to 5. With this setting, GOBNILP will produce an initial listing of parameters not set to their default value. If you're playing with different parameter settings this is a good way to keep track of how different parameter settings perform. In addition, after the search finishes, SCIP will 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). Finally, at verbosity level 5, you will get information on the search tree, in particular information on variables that were branched upon.

Columns in the output which display information such as the number of cuts are by default suppressed if verbosity is below 4. To change this behaviour set the parameter `gobnilp/verblevelsetscols` to `FALSE`.

4.4.1 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
```

4.4.2 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`

4.5 Defining a BN learning problem using the CIP format

GOBNILP is capable of reading in and solving problems stored in SCIP's CIP format. This is specified through the command line argument of `-f=cip` or giving the input file a `.cip` filename extension.

Unlike the other input formats, a CIP file contains the ILP problem (objective function, variables and constraints) and not just the data (or scores) to construct it. It is not intended that users will create their own CIP files as input, instead GOBNILP can be used to create an initial CIP file which can then be edited. Details on how to get GOBNILP to produce a CIP file as output are given in Chapter 6. Please see the SCIP documentation for a full explanation of the CIP format.

4.6 Advanced output options

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.

Chapter 5

Troubleshooting / Howtos

1. *GOBNILP is very slow. How can I speed it up?* You may not be able to for your particular problem, but here are some things to try.

- If you have CPLEX installed be sure to make GOBNILP using `make LPS=cpx`. This will ensure that CPLEX is used to solve the many linear relaxations that are generated by GOBNILP
- You can, of course, compromise and solve an easier problem. For example, you can give up on getting an optimal solution by setting SCIP's `limits/time` and `limits/gap` appropriately. Alternatively, you can reduce the limit on the number of parents allowed in a parent set via the GOBNILP parameter `gobnilp/scoring/palim`. Setting `gobnilp/minfounders` and `gobnilp/edge_penalty` to large values also speeds up solving.
- You can try non-default parameter settings for GOBNILP. GOBNILP's most important decision is when to stop adding cutting planes in the root node and to start branch-and-bound. GOBNILP can guess wrong on this: either adding cuts when it should be branching on variables, or stopping cut generation too early and thus failing to generate a crucial cut. The following parameters (given with GOBNILP's default values) affect this decision:
  - `separating/maxstallrounds = 10`
  - `constraints/dagcluster/forcecuts = TRUE`
  - `constraints/dagcluster/rootgapsepalimit = 0.001`
  - `separating/minefficacyroot = 0.0005`
  - `separating/maxcutsroot = 300`

  You can also reduce the time spent on looking for 'cluster' cuts (whether in the root or elsewhere in the search tree) via the following paramters:
  - `constraints/dagcluster/sepafreq = 10`

- `constraints/dagcluster/consenfolpsubiptimelimit = 20`
- `constraints/dagcluster/conssepalpsubiptimelimit = 10`
- `constraints/dagcluster/conssepasolsubiptimelimit = 10`

GOBNILP uses SCIP's default approach to choosing which variable to branch on: *reliability branching on pseudo cost values* (`relpscost`) but we use non-default values:

- `branching/relpscost/inferenceweight = 0.1`
- `branching/relpscost/maxlookahead = 1`

You may get better performance by putting these back to SCIP's default values.

2. *The local BDeu scores that GOBNILP produces seem wrong—some are even positive!* This may be due to numerical problems, perhaps due to a very high value for the *effective sample size* (set via `gobnilp/scoring/alpha`). Try setting `gobnilp/scoring/fast` to `FALSE`.

3. *I want to bias GOBNILP towards sparse graphs. How do I do this?* Set `gobnilp/edge_penalty` to a positive value.

4. *GOBNILP is failing to find any solution—let alone an optimal one!* Try setting `heuristics/sinks/probing` to `TRUE`. You may also need to increase `heuristics/sinks/maxdivedepth`.

Chapter 6

Exiting without solving

The command line argument `-x` causes GOBNILP to exit just before attempting to find the best BN. The primary purpose of this is to enable the ILP model to be created and saved to file in CIP format without being solved. To do this, the parameter `gobnilp/outputfile/cip` should be set to something other than `""` and the program called with the `-x` argument. The model will be generated as usual, written to the specified location and then GOBNILP will exit without solving the model. If the parameter `gobnilp/presolvecip` is set to `TRUE` then the problem instance is presolved before being output. As GOBNILP can read CIP files, the output can subsequently be loaded back into GOBNILP without the `-x` argument and the solving can proceed.

Writing out the ILP model in this way has several purposes. First, it allows examination of exactly what constraints have been added to the model. Second, the model can be edited, perhaps adding additional constraints or to assess sensitivity to a value. Third, a basic model can be saved once and then solved under a variety of parameter settings to determine their effect.

Some care must be taken in this final case; the CIP model output will contain any additional constraints added to the model (such as those in Section 3.5). These will not be removed if the subsequent reloading and solving of the model takes place using parameter settings that say these constraints should not occur. Generally speaking, additional limitations on the problem added when creating the CIP file will never be removed when that CIP file is later used though further, tighter restrictions can be added with parameter settings.

Another use of the `-x` argument is to allow the local BDeu scores to be computed and written out, without then proceeding to the solving phase. If data is given as input, `gobnilp/outputfile/scores` can be used to write the local scores computed to file. As computation of local scores can be very time consuming, there can be considerable advantage in saving the local scores as a `.jkl` file and restarting any future BN learning from that stage.

Chapter 7

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 [8] and from this to state the problem of finding the most likely pedigree as finding a BN with the highest score. See [4] 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.

1. Sex consistency of the pedigree can be enforced.
2. Age information can be used to constrain the learning process.
3. Limitations on family sizes can be specified.
4. Output can be produced in a format convenient for interchange with other pedigree based programs.
5. Genetic profiles can be input and the likelihood scores can be calculated inside GOBNILP.

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.

If the ages of individuals are known, 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. How this can be done is explained when discussing input below.

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 3.3 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.

7.1 Input from genetic profiles

If log likelihood scores for parentage combinations are known, these can be input using the score format as explained in Section 3.4.2. However, GOBNILP can also compute these scores itself from the marker data for the individuals. The scores calculated are on the basis of strict Mendelian inheritance, please see [4] for a full explanation of the assumptions behind the calculated scores.

The basic format for the marker data file is one row for each individual. Each row should have an equal number of columns, with the separator between columns specified using the gobnilp/delimiter and gobnilp/mergedelimiters properties as described in Section 3.4.1. There may be an additional header row containing names of the columns, which is specified using the

gobnilp/pedigree/inputfile/columnheaders parameter.

Columns can contain any data and in any order, making it simple to import data from a spreadsheet for example. The only restriction is that the marker data is all in a contiguous block of columns and that the two alleles of each marker are in adjacent columns.

In order to tell GOBNILP which columns contain the marker data, the gobnilp/pedigree/inputfile/allelestartcolumn and gobnilp/pedigree/inputfile/alleleendcolumn parameters are used. For example, a file in which the first 10 columns contain marker data, one would set

```
gobnilp/pedigree/inputfile/allelestartcolumn = 0
gobnilp/pedigree/inputfile/alleleendcolumn = 9
```

Note that column numbers start from 0.

In addition to the marker data for individuals, it is possible to give individuals' names, ages and sexes. This information will be used to limit which individuals can be parents of other individuals. Each of these forms of information is also given by specifying a column number, using the parameters gobnilp/pedigree/inputfile/namecolumn, gobnilp/pedigree/inputfile/agecolumn and gobnilp/pedigree/inputfile/sexcolumn. Names can be any value. Ages must be integers. Sexes can be any value, but any value beginning with a `F` will be interpreted as female, any value beginning with a `M` as male, and anything else as unknown.

If exact ages are not known, but it is known whether each individual is an adult or a juvenile, then the parameter gobnilp/pedigree/inputfile/adultcolumn can be used. Each individual which is an adult should be given a `TRUE` value, with a `FALSE` value otherwise.

If any of this information is not in the database, or you wish to exclude it, then simply set the appropriate parameter to -1 or omit it altogether.

As an example of this, consider the following first few lines of a CSV style file

```
Name,Age,Sex,City,CSF1PO_a,CSF1PO_b,FGA_a,FGA_b,TH01_a,TH01_b,TPOX_a,TPOX_b
John,35,Male,London,10,11,20,23,9,11,5,5
Peter,31,Male,New York,8,11,18,18,8,11,10,11
Mary,25,Female,Los Angeles,10,11,18,20,9,9,3,11,12
```

To read this information in, one would set

```
gobnilp/delimiter = ","
gobnilp/mergedelimiters = FALSE
gobnilp/pedigree/inputfile/columnheaders = TRUE
gobnilp/pedigree/inputfile/allelestartcolumn = 4
gobnilp/pedigree/inputfile/alleleendcolumn = 11
gobnilp/pedigree/inputfile/namecolumn = 0
gobnilp/pedigree/inputfile/agecolumn = 1
gobnilp/pedigree/inputfile/sexcolumn = 2
```

Once the information has been read in, it is used to compute log likelihoods of each possible parentage combination. In order to do this, it is necessary to know the population frequencies for each of the alleles of each marker. If these are not known, then the frequencies found in the data file can be used. However if they are known, they can be given using `-q=`*frequencyfile* from the command line. The frequency file should give the markers in the same order as they appear in the main data file. Each marker should have two rows, the first giving the names of the alleles and the second giving the frequencies of the alleles. The separator used should match that given for the main file.

An example of the first few column and lines of a frequency file is given below.

```
308,380,339,...
0.147058823529,0.117647058824,0.0294117647059, ...
458,468,454,...
0.229411764706,0.0294117647059,0.158823529412,..
```

Finally, it is possible to control which candidate parent sets are considered. The parameter `gobnilp/pedigree/singleparents` can be used to tell GOBNILP whether to consider parent sets of size 1, i.e. where an individual can have one parent who is known and the other who is unobserved.

7.2 Pedigree parameters

gobnilp/pedigreemode

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

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

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

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

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

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

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

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

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

# whether the pedigree can feature individuals with only one parent
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/pedigree/singleparents = TRUE

# whether the pedigree input file has column headers
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/pedigree/inputfile/columnheaders = TRUE

# the column with individuals' names (-1 if no such column)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/pedigree/inputfile/namecolumn = -1

# the column with individuals' sexes (-1 if no such column)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/pedigree/inputfile/sexcolumn = -1

# the column with individuals' ages (-1 if no such column)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/pedigree/inputfile/agecolumn = -1

# the first column with individuals' alleles
# [type: int, range: [0,2147483647], default: 0]
gobnilp/pedigree/inputfile/allelestartcolumn = 0

# the last column with individuals' alleles
# [type: int, range: [1,2147483647], default: 1]
gobnilp/pedigree/inputfile/alleleendcolumn = 1

# the column stating which individuals are adults (-1 if no such column)
# [type: int, range: [-1,2147483647], default: -1]
```

```
gobnilp/pedigree/inputfile/adultcolumn = -1

# the symbol used for unknown genotypes
# [type: string, default: "NA"]
gobnilp/pedigree/realdata/unknown = "NA"

# the most genotype mismatches that are allowed between a child and parent
# [type: int, range: [0,2147483647], default: 0]
gobnilp/pedigree/realdata/maxmismatches = 0

# the value to use for mutation probability
# [type: real, range: [0,1], default: 0]
gobnilp/pedigree/realdata/mutationprob = 0
```

Chapter 8

Debugging

8.1 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.

8.2 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.

Chapter 9

GOBNILP parameter listing

In this chapter all current GOBNILP parameters are listed. Quite a few of these are not described in the main text of this manual. Section 9.7 lists the changes in parameters between versions 1.4.1 and 1.6.

9.1 Dagcluster constraint parameters

```
# frequency for separating cuts (-1: never, 0: only in root node)
# [type: int, range: [-1,2147483647], default: 10]
constraints/dagcluster/sepafreq = 10

# frequency for propagating domains (-1: never, 0: only in root node)
# [type: int, range: [-1,2147483647], default: 1]
constraints/dagcluster/propfreq = 1

# timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTE
# [type: int, range: [1,15], default: 1]
constraints/dagcluster/timingmask = 1

# frequency for using all instead of only the useful constraints in separation, propagati
# [type: int, range: [-1,2147483647], default: 100]
constraints/dagcluster/eagerfreq = 100

# maximal number of presolving rounds the constraint handler participates in (-1: no limi
# [type: int, range: [-1,2147483647], default: -1]
constraints/dagcluster/maxprerounds = -1

# should separation method be delayed, if other separators found cuts?
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/dagcluster/delaysepa = FALSE
```

```
# should propagation method be delayed, if other propagators found reductions?
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/dagcluster/delayprop = FALSE

# should presolving method be delayed, if other presolvers found reductions?
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/dagcluster/delaypresol = FALSE

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

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

# whether to add cluster cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/dagcluster/clustercuts_addtopool = TRUE

# whether to propagate added set packing constraints
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/dagcluster/propagatespc = TRUE

# only write out pre-separation MIP relaxation after reaching this number of calls to sep
# [type: int, range: [1,2147483647], default: 1]
constraints/dagcluster/writemip_minsepalp_calls = 1

# only write out pre-separation MIP relaxation if not beyond this number of calls to sepa
# [type: int, range: [0,2147483647], default: 0]
constraints/dagcluster/writemip_maxsepalp_calls = 0

# do not use the subIP to search for cluster cuts if deeper than this (-1 for no limit)
# [type: int, range: [-1,2147483647], default: -1]
constraints/dagcluster/subipdepthlimit = -1

# whether to look for cluster cuts which are tight for the current incumbent
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/dagcluster/useincumbentcons = FALSE

# whether to always use the subIP to look for cluster cuts
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/dagcluster/subipalways = TRUE

# whether to ever use the subIP to look for cluster cuts
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
```

```
constraints/dagcluster/subipever = TRUE

# whether to always use fractional cycles to look for cluster cuts
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/dagcluster/fractionalalways = FALSE

# whether to ever use fractional cycles to look for cluster cuts
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/dagcluster/fractionalever = FALSE

# whether to search for convex4 cuts when separating an arbitrary primal solution
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/dagcluster/sepasol_useconvex4 = FALSE

# time limit on sub IP for finding cluster cuts in LP separation
# [type: real, range: [0,1e+20], default: 10]
constraints/dagcluster/conssepalpsubiptimelimit = 10

# time limit on sub IP for finding cluster cuts in arbitrary primal solution separation
# [type: real, range: [0,1e+20], default: 10]
constraints/dagcluster/conssepasolsubiptimelimit = 10

# time limit on sub IP for finding cluster cuts in LP enforcement
# [type: real, range: [0,1e+20], default: 20]
constraints/dagcluster/consenfolpsubiptimelimit = 20

# gap limit on sub IP for finding cluster cuts in LP separation
# [type: real, range: [0,1.7e+308], default: 0]
constraints/dagcluster/conssepalpsubipgaplimit = 0

# gap limit on sub IP for finding cluster cuts in arbitrary primal solution separation
# [type: real, range: [0,1.7e+308], default: 0]
constraints/dagcluster/conssepasolsubipgaplimit = 0

# gap limit on sub IP for finding cluster cuts in LP enforcement
# [type: real, range: [0,1.7e+308], default: 0]
constraints/dagcluster/consenfolpsubipgaplimit = 0

# absgap limit on sub IP for finding cluster cuts in LP separation
# [type: real, range: [0,1.7e+308], default: 0]
constraints/dagcluster/conssepalpsubipabsgaplimit = 0

# absgap limit on sub IP for finding cluster cuts in arbitrary primal solution separatio
# [type: real, range: [0,1.7e+308], default: 0]
constraints/dagcluster/conssepasolsubipabsgaplimit = 0
```

```
# absgap limit on sub IP for finding cluster cuts in LP enforcement
# [type: real, range: [0,1.7e+308], default: 0]
constraints/dagcluster/consenfolpsubipabsgaplimit = 0

# whether to force all cuts to be added
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/dagcluster/forcecuts = TRUE

# whether to do extra (slow) propagations
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/dagcluster/extraprops = FALSE

# no cuts generated in root if gap is below this value
# [type: real, range: [0,1e+20], default: 0.001]
constraints/dagcluster/rootgapsepalimit = 0.001

# whether cluster cuts should be added based on cycles in the current solution
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/circuit_cuts/add_cluster_cuts = TRUE

# whether cluster cuts should be added to the pool based on cycles in the current soluti
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/circuit_cuts/add_cluster_cuts_to_pool = FALSE

# whether cycle cuts should be added based on cycles in the current solution
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/circuit_cuts/add_cycle_cuts = FALSE

# whether cycle cuts should be added to the pool based on cycles in the current solution
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/circuit_cuts/add_cycle_cuts_to_pool = FALSE

# whether cuts should be added based on fractional cycles in the current solution
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/circuit_cuts/use_fractional = TRUE

# the maximum length of cycle to search for
# [type: int, range: [0,2147483647], default: 6]
gobnilp/circuit_cuts/max_length = 6

# whether cluster cuts should be added based on fractional cycles in the current solutio
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/fractional_circuit_cuts/add_cluster_cuts = TRUE

# whether cluster cuts should be added to the pool based on fractional cycles in the cur
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
```

```
gobnilp/fractional_circuit_cuts/add_cluster_cuts_to_pool = FALSE
```

9.2 Convex4 cuts

```
# maximum number of 'convexhull4b' cuts to add initially (-1 for no upper bound)
# [type: int, range: [-1,2147483647], default: 0]
constraints/convex4_cuts/b_init_limit = 0

# maximum number of 'convexhull4c' cuts to add initially (-1 for no upper bound)
# [type: int, range: [-1,2147483647], default: 0]
constraints/convex4_cuts/c_init_limit = 0

# maximum number of 'convexhull4d' cuts to add initially (-1 for no upper bound)
# [type: int, range: [-1,2147483647], default: 0]
constraints/convex4_cuts/d_init_limit = 0

# maximum number of 'convexhull4e' cuts to add initially (-1 for no upper bound)
# [type: int, range: [-1,2147483647], default: 0]
constraints/convex4_cuts/e_init_limit = 0

# maximum number of 'convexhull4g' cuts to add initially (-1 for no upper bound)
# [type: int, range: [-1,2147483647], default: 0]
constraints/convex4_cuts/g_init_limit = 0

# maximum number of 'convexhull4h' cuts to add initially (-1 for no upper bound)
# [type: int, range: [-1,2147483647], default: 0]
constraints/convex4_cuts/h_init_limit = 0

# maximum number of 'convexhull4i' cuts to add initially (-1 for no upper bound)
# [type: int, range: [-1,2147483647], default: 0]
constraints/convex4_cuts/i_init_limit = 0

# whether to add initial 'convexhull4b' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/b_init_addtopool = FALSE

# whether to add initial 'convexhull4c' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/c_init_addtopool = FALSE

# whether to add initial 'convexhull4d' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/d_init_addtopool = FALSE
```

```
# whether to add initial 'convexhull4e' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/e_init_addtopool = FALSE

# whether to add initial 'convexhull4g' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/g_init_addtopool = FALSE

# whether to add initial 'convexhull4h' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/h_init_addtopool = FALSE

# whether to add initial 'convexhull4i' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/i_init_addtopool = FALSE

# whether to add initial 'convexhull4b' cuts to the initial LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/b_init_addcut = TRUE

# whether to add initial 'convexhull4c' cuts to the initial LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/c_init_addcut = TRUE

# whether to add initial 'convexhull4d' cuts to the initial LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/d_init_addcut = TRUE

# whether to add initial 'convexhull4e' cuts to the initial LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/e_init_addcut = TRUE

# whether to add initial 'convexhull4g' cuts to the initial LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/g_init_addcut = TRUE

# whether to add initial 'convexhull4h' cuts to the initial LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/h_init_addcut = TRUE

# whether to add initial 'convexhull4i' cuts to the initial LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/i_init_addcut = TRUE

# whether to use 'convexhull4b' cuts
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
```

```
constraints/convex4_cuts/b_enable = TRUE

# whether to use 'convexhull4c' cuts
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/c_enable = FALSE

# whether to use 'convexhull4d' cuts
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/d_enable = FALSE

# whether to use 'convexhull4e' cuts
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/e_enable = FALSE

# whether to use 'convexhull4g' cuts
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/g_enable = FALSE

# whether to use 'convexhull4h' cuts
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/h_enable = FALSE

# whether to use 'convexhull4i' cuts
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/i_enable = FALSE

# whether to add separating 'convexhull4b' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/b_addtopool = TRUE

# whether to add separating 'convexhull4c' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/c_addtopool = TRUE

# whether to add separating 'convexhull4d' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/d_addtopool = TRUE

# whether to add separating 'convexhull4e' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/e_addtopool = TRUE

# whether to add separating 'convexhull4g' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/g_addtopool = TRUE
```

```
# whether to add separating 'convexhull4h' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/h_addtopool = TRUE

# whether to add separating 'convexhull4i' cuts to the global cut pool
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/i_addtopool = TRUE

# whether to add separating 'convexhull4b' cuts to the LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/b_addcut = TRUE

# whether to add separating 'convexhull4c' cuts to the LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/c_addcut = TRUE

# whether to add separating 'convexhull4d' cuts to the LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/d_addcut = TRUE

# whether to add separating 'convexhull4e' cuts to the LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/e_addcut = TRUE

# whether to add separating 'convexhull4g' cuts to the LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/g_addcut = TRUE

# whether to add separating 'convexhull4h' cuts to the LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/h_addcut = TRUE

# whether to add separating 'convexhull4i' cuts to the LP
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/i_addcut = TRUE

# whether to only add 'convexhull4b' when no cluster cuts can be found
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/convex4_cuts/b_delay = FALSE

# whether to only add 'convexhull4b' when no cluster cuts can be found
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/c_delay = TRUE

# whether to only add 'convexhull4d' when no cluster cuts can be found
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
```

```
constraints/convex4_cuts/d_delay = TRUE

# whether to only add 'convexhull4e' when no cluster cuts can be found
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/e_delay = TRUE

# whether to only add 'convexhull4g' when no cluster cuts can be found
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/g_delay = TRUE

# whether to only add 'convexhull4h' when no cluster cuts can be found
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/h_delay = TRUE

# whether to only add 'convexhull4i' when no cluster cuts can be found
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/convex4_cuts/i_delay = TRUE
```

9.3 Conditional independence constraint parameters

```
# frequency for separating cuts (-1: never, 0: only in root node)
# [type: int, range: [-1,2147483647], default: 1]
constraints/ci/sepafreq = 1

# frequency for propagating domains (-1: never, 0: only in root node)
# [type: int, range: [-1,2147483647], default: -1]
constraints/ci/propfreq = -1

# timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTE
# [type: int, range: [1,15], default: 1]
constraints/ci/timingmask = 1

# frequency for using all instead of only the useful constraints in separation, propagati
# [type: int, range: [-1,2147483647], default: 100]
constraints/ci/eagerfreq = 100

# maximal number of presolving rounds the constraint handler participates in (-1: no limi
# [type: int, range: [-1,2147483647], default: -1]
constraints/ci/maxprerounds = -1

# should separation method be delayed, if other separators found cuts?
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
```

```
constraints/ci/delaysepa = FALSE

# should propagation method be delayed, if other propagators found reductions?
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/ci/delayprop = FALSE

# should presolving method be delayed, if other presolvers found reductions?
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
constraints/ci/delaypresol = FALSE

# whether to force all cuts to be added
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
constraints/ci/forcecuts = TRUE
```

9.4 Sink heuristic parameters

```
# priority of heuristic <sinks>
# [type: int, range: [-536870912,536870911], default: 10]
heuristics/sinks/priority = 10

# frequency for calling primal heuristic <sinks> (-1: never, 0: only at depth freqofs)
# [type: int, range: [-1,2147483647], default: 0]
heuristics/sinks/freq = 0

# frequency offset for calling primal heuristic <sinks>
# [type: int, range: [0,2147483647], default: 0]
heuristics/sinks/freqofs = 0

# maximal depth level to call primal heuristic <sinks> (-1: no limit)
# [type: int, range: [-1,2147483647], default: -1]
heuristics/sinks/maxdepth = -1

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

# whether to use probing
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
heuristics/sinks/probing = FALSE

# maximum dive depth when using probing
# [type: int, range: [0,2147483647], default: 100]
heuristics/sinks/maxdivedepth = 100
```

```
# initial value for random seed
# [type: int, range: [0,2147483647], default: 0]
heuristics/sinks/seed = 0

# how many times to run the heuristic on each LP solution
# [type: int, range: [0,2147483647], default: 1]
heuristics/sinks/nruns = 1

# whether to assume that no problem variable has a positive objective coefficient
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
heuristics/sinks/assumenoposobj = TRUE

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

9.5 Scoring parameters

```
# whether to prune during scoring
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/scoring/prune = TRUE

# minimum number of datapoints to create a branch in the AD tree
# [type: int, range: [0,2147483647], default: 1000]
gobnilp/scoring/rmin = 1000

# limit on the depth of the AD tree
# [type: int, range: [0,2147483647], default: 1000]
gobnilp/scoring/adtreedepthlim = 1000

# limit on the number of nodes in  the AD tree
# [type: int, range: [0,2147483647], default: 10000]
gobnilp/scoring/adtreenodeslim = 10000

# limit on number of scores that are cached
# [type: int, range: [0,2147483647], default: 10000000]
gobnilp/scoring/cachesizelimit = 10000000

# how much to increase the cache when needed and allowed
# [type: int, range: [0,2147483647], default: 10000]
gobnilp/scoring/cacheblocksize = 10000

# subsets must have size below this to be cached
# [type: int, range: [0,2147483647], default: 4]
```

```
gobnilp/scoring/nvarscachelimit = 4

# maximum number of parents for each node (-1 for no limit)
# [type: int, range: [-1,2147483647], default: 3]
gobnilp/scoring/palim = 3

# alpha value for use in BDeu scoring
# [type: real, range: [0,1e+20], default: 1]
gobnilp/scoring/alpha = 1

# whether variable names are given in the data file
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/scoring/names = TRUE

# whether variable arities are given in the data file
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/scoring/arities = TRUE

# If Pa1 and Pa2 are parent sets for some variable, Pa1 is a subset of Pa2 and local_sco
# [type: real, range: [0,1e+20], default: 0]
gobnilp/scoring/prunegap = 0

# whether to fast scoring (which is inaccurate for high values of ESS)
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/scoring/fast = TRUE
```

9.6 Other GOBNILP parameters

```
# whether model averaging should assume logarithmic objective function
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/logaverage = TRUE

# whether a verbosity level of at most 3 suppresses columns in the display
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/verblevelsetscols = TRUE

# whether the problem is 'vanilla' BN learning
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/vanilla = FALSE

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

```
# whether to assume a decomposable model is being learned with a
# likelihood-equivalent score
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/decomposable = FALSE

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

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

# whether to presolve the problem before printing out CIP version
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/presolvecip = FALSE

# the maximum number of children a node can have (-1 for no limit)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/maxnchildren = -1

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

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

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

# minimum number of parents
# [type: int, range: [0,2147483647], default: 0]
gobnilp/minparents = 0

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

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

# maximum number of edges (-1 for no upper bound )
```

```
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/maxedges = -1

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

# whether to treat consecutive delimitors in the input file as one
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/mergedelimiters = TRUE

# the delimiter for fields in the input file (special values - whitespace, tab)
# [type: string, default: "whitespace"]
gobnilp/delimiter = "whitespace"

# whether to create totalorder variables
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/totalordervars = FALSE

# whether to create variables counting the number of children a node has
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/nchildrenvars = FALSE

# whether to only allow best scoring parents for a given total order
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/bestparentsfororder = TRUE

# whether to use SCIP's linear ordering constraint
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/useconslinearordering = TRUE

# whether to use the dagcluster acyclicity constraint
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/useconsdagcluster = TRUE

# whether to add transitivity constraints on totalorder variables initially (rather than
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/inittransitivity = FALSE

# whether to create imset variables
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/imsetvars = FALSE

# whether to create position indicator variables
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/posindvars = FALSE
```

```
# whether to create position variables
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/posvars = FALSE

# maximum value of a position variable (-1 for no restriction)
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/maxpos = -1

# whether to create position variables indicate position in a total order of BN variables
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/postotal = FALSE

# whether to create parent set size variables
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/pasizevars = FALSE

# branching priority of parent set size variables
# [type: int, range: [0,2147483647], default: 0]
gobnilp/pasizepriority = 0

# branching priority of family variables
# [type: int, range: [-2147483647,2147483647], default: 0]
gobnilp/pavarspriority = 0

# branching priority of edge variables
# [type: int, range: [-2147483647,2147483647], default: 10]
gobnilp/edgevarpriority = 10

# branching priority of nchildrenvar variables
# [type: int, range: [-2147483647,2147483647], default: 0]
gobnilp/nchildrenvarpriority = 0

# branching priority of arrow variables
# [type: int, range: [-2147483647,2147483647], default: 10]
gobnilp/arrowvarpriority = 10

# branching priority of imset variables
# [type: int, range: [-2147483647,2147483647], default: 0]
gobnilp/imsetvarpriority = 0

# branching priority of imset variables
# [type: int, range: [-2147483647,2147483647], default: 0]
gobnilp/totalordervarpriority = 0

# branching priority of posind variables
```

```
# [type: int, range: [-2147483647,2147483647], default: 0]
gobnilp/posindvarpriority = 0

# whether to split dagcluster constraints into strongly connected components
# [type: bool, range: {TRUE,FALSE}, default: TRUE]
gobnilp/splitdags = TRUE

# edge_penalty*|edges| is subtracted from objective
# [type: real, range: [-1e+20,1e+20], default: 0]
gobnilp/edge_penalty = 0

# whether GOBNILP should return exemplars of the k best MECs instead of the k best BNs
# [type: bool, range: {TRUE,FALSE}, default: FALSE]
gobnilp/kbestMEC = FALSE

# file which solution should be printed to (stdout for standard out, empty string for no
# [type: string, default: "stdout"]
gobnilp/outputfile/solution = "stdout"

# file which dot output should be printed to (stdout for standard out, empty string for
# [type: string, default: ""]
gobnilp/outputfile/dot = ""

# file which pedigree output should be printed to (stdout for standard out, empty string
# [type: string, default: ""]
gobnilp/outputfile/pedigree = ""

# file which additional score and time data should be printed to (stdout for standard ou
# [type: string, default: ""]
gobnilp/outputfile/scoreandtime = ""

# file which adjacency matrix output should be printed to (stdout for standard out, empt
# [type: string, default: ""]
gobnilp/outputfile/adjacencymatrix = ""

# file which cip problem representation should be printed to (stdout for standard out, e
# [type: string, default: ""]
gobnilp/outputfile/cip = ""

# how many iterations to skip before first model averaging output (-1 to suppress always
# [type: int, range: [-1,2147483647], default: -1]
gobnilp/avgoutputoffset = -1

# how many iterations between outputting model averaging information
# [type: int, range: [1,2147483647], default: 1]
gobnilp/avgoutputstep = 1
```

```
# file which model averging solution should be printed to (stdout for standard out, empty
# [type: string, default: "stdout"]
gobnilp/outputfile/solutionavg = "stdout"

# file which model averging dot output should be printed to (stdout for standard out, emp
# [type: string, default: ""]
gobnilp/outputfile/dotavg = ""

# file which model averging pedigree output should be printed to (stdout for standard out
# [type: string, default: ""]
gobnilp/outputfile/pedigreeavg = ""

# file which model averging additional score and time data should be printed to (stdout f
# [type: string, default: ""]
gobnilp/outputfile/scoreandtimeavg = ""

# file which model averging adjacency matrix output should be printed to (stdout for star
# [type: string, default: ""]
gobnilp/outputfile/adjacencymatrixavg = ""

# file which parent set scores should be output (stdout for standard out, empty string fo
# [type: string, default: ""]
gobnilp/outputfile/scores = ""

# file which Markov equivalence class should be output (stdout for standard out, empty st
# [type: string, default: ""]
gobnilp/outputfile/mec = ""

# file to which parent set scores should be output (stdout for standard out, empty string
# [type: string, default: ""]
gobnilp/outputfile/pss = ""
```

9.7 Parameter changes between 1.4.1 and 1.6

9.7.1 New parameters for version 1.6

```
constraints/dagcluster/extraprops = FALSE
constraints/dagcluster/forcecuts = TRUE
constraints/dagcluster/fractionalalways = FALSE
constraints/dagcluster/fractionalever = FALSE
constraints/dagcluster/rootgapsepalimit = 0.001
constraints/dagcluster/subipever = TRUE
gobnilp/arrowvarpriority = 10
gobnilp/circuit_cuts/use_fractional = TRUE
```

```
gobnilp/decomposable = FALSE
gobnilp/edge_penalty = 0
gobnilp/edgevarpriority = 10
gobnilp/fractional_circuit_cuts/add_cluster_cuts = TRUE
gobnilp/fractional_circuit_cuts/add_cluster_cuts_to_pool = FALSE
gobnilp/kbestMEC = FALSE
gobnilp/logaverage = TRUE
gobnilp/maxnchildren = -1
gobnilp/nchildrenvarpriority = 0
gobnilp/nchildrenvars = FALSE
gobnilp/outputfile/pss = ""
gobnilp/pavarspriority = 0
gobnilp/posindvarpriority = 0
gobnilp/postotal = FALSE
gobnilp/presolvecip = FALSE
gobnilp/scoring/adtreedepthlim = 1000
gobnilp/scoring/adtreenodeslim = 10000
gobnilp/scoring/cacheblocksize = 10000
gobnilp/scoring/cachesizelimit = 10000000
gobnilp/scoring/fast = TRUE
gobnilp/scoring/nvarscachelimit = 4
gobnilp/scoring/prunegap = 0
gobnilp/scoring/rmin = 1000
gobnilp/splitdags = TRUE
gobnilp/useconsdagcluster = TRUE
gobnilp/vanilla = FALSE
heuristics/sinks/nruns = 1
heuristics/sinks/seed = 0
```

9.7.2 Deprecated parameters

```
#fix this to 1 - SCIP's epsilon
gobnilp/cycles/link_threshold = 0.99

#edge variables are now always present
gobnilp/edgevars = FALSE

#now we never remove variables indicating empty parent sets
gobnilp/implicitfounders = FALSE
```

9.7.3 Parameters with changed default values

In this section, GOBNILP and SCIP parameters are listed whose default values in 1.6 differ from those in 1.4.1.

```
branching/relpscost/inferenceweight = 0.1
```

```
branching/relpscost/maxlookahead = 1
constraints/ci/sepafreq = 1
constraints/dagcluster/consenfolpsubiptimelimit = 20
constraints/dagcluster/conssepalpsubiptimelimit = 10
constraints/dagcluster/conssepasolsubiptimelimit = 10
constraints/dagcluster/subipalways = TRUE
heuristics/sinks/freq = 0
separating/maxcutsroot = 300
separating/maxstallrounds = 10
separating/minefficacyroot = 0.0005
```

9.7.4 Parameters with changed names

```
constraints/dagcluster/convexhull4bcuts = TRUE
constraints/dagcluster/convexhull4bcuts_addcut = TRUE
constraints/dagcluster/convexhull4bcuts_addtopool = TRUE
constraints/dagcluster/convexhull4bcuts_delay = FALSE
```

is now

```
constraints/convex4_cuts/b_enable = TRUE
constraints/convex4_cuts/b_addcut = TRUE
constraints/convex4_cuts/b_addtopool = TRUE
constraints/convex4_cuts/b_delay = FALSE
```

and similarly for other 'convex4' parameters.

```
gobnilp/cycles/add_cluster_cuts = TRUE
```

is now

```
gobnilp/circuit_cuts/add_cluster_cuts = TRUE
```

and similarly for other 'cycles' parameters.

Chapter 10

Citing GOBNILP

When citing GOBNILP please use [3] and/or [2]. (GOBNILP is an improved version of the software used to create the results in the first of these papers.) 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 [6]. The use of ILP for pedigree learning is introduced in [4], which should also be cited if you use GOBNILP for this purpose.

Acknowledgements

- GOBNILP version 1.2-1.6 and higher was 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 [6]. Additional thanks to Tommi Jaakkola and David Sontag for providing data and encouragement-to-distribute, respectively.

Appendix A

Correctness of pruning

The GOBNILP BDeu scoring code uses a slightly modified version of the pruning approach devised by de Campos and Ji [5]. 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 α' be a positive real. Then*

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

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

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

Proof. With the sum $n_{ij} = \sum_{k=1}^{r_i} n_{ijk}$ fixed consider distributions of counts $\{n_{ijk}\}_{k=1,\ldots r_i}$ 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 (A.1) the increase in the LHS of (A.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 (A.2) is increasing. The sequence terminates when $n_{ijk^*} = n_{ij}$ for some k^*. The result follows. □

Theorem 1.

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

Proof.

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

□

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

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

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 α). 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 (A.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 (A.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. □

Bibliography

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

[2] Mark Bartlett and James Cussens. Advances in Bayesian network learning using integer programming. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI 2013).* AUAI Press, 2013. To appear.

[3] 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.

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

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

[6] 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.

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

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

[9] Andrew Moore and Mary Soon Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998.

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