

# CaMML Manual

Charles R. Twardy & Rodney O'Donnell  
Covers rodo version

`www.datamining.monash.edu.au/software/camml`  
`camml@csse.monash.edu.au`

*Revision* : 1.10

*Date* : 2004/03/05 09 : 45 : 52

July 23, 2004

# Contents

<b>1</b>	<b>Introduction to CaMML</b>	<b>2</b>
1.1	A word about versions . . . . .	2
1.2	Getting CaMML . . . . .	3
<b>2</b>	<b>Quick Start</b>	<b>4</b>
2.1	Using CaMML . . . . .	4
2.1.1	Initialize the instance . . . . .	5
2.1.2	Add priors (soft constraints) . . . . .	5
2.1.3	Find a model: <b>g</b> . . . . .	6
2.1.4	View (“Print”) the model . . . . .	6
2.1.5	Export as Netica network: <b>x</b> . . . . .	7
2.1.6	Quit: <b>q</b> . . . . .	8
2.2	Other options and concepts . . . . .	8
2.2.1	Instances . . . . .	8
2.2.2	Miscellaneous . . . . .	8
2.2.3	View recalculated prior matrix: <b>b</b> . . . . .	8
2.2.4	Run inner CaMML: <b>r</b> . . . . .	9
2.3	Things you should know . . . . .	9
2.3.1	Data Format . . . . .	9
2.3.2	Limits . . . . .	10
<b>3</b>	<b>The Inner CaMML</b>	<b>11</b>
3.1	A normal session with old CaMML . . . . .	11
3.2	Constraints . . . . .	11
3.3	Gibbs Sampling . . . . .	12
3.3.1	<b>g</b> : Do Gibbs sampling starting from current . . . . .	12
3.3.2	<b>T</b> : Change search temperature, cap cost . . . . .	12
3.3.3	<b>w</b> <b>W</b> : Search for <b>W</b> times normal steps ( <b>W</b> is float) . . . . .	12
3.4	Model Input . . . . .	12
3.4.1	Model Format . . . . .	12
3.5	Working with Models . . . . .	13
3.5.1	Selecting models . . . . .	13
3.5.2	Printing models . . . . .	13
3.5.3	Modifying models . . . . .	15
3.5.4	Analyzing models . . . . .	15
3.6	Misc . . . . .	16
<b>4</b>	<b>How does CaMML Work?</b>	<b>17</b>
4.1	Gibbs sampling . . . . .	17
4.1.1	<b>k</b> <b>ba</b> : Cripple using bitmask <b>ba</b> : $a = \text{simplify}$ , $b = \text{P(arc)}$ . . . . .	17
4.2	What are the scores? . . . . .	18
4.3	Search Algorithm . . . . .	19
4.3.1	Linear Search Algorithm . . . . .	19
4.3.2	Discrete Search Algorithm . . . . .	19

# Chapter 1

## Introduction to CaMML

CaMML is a program which learns Bayesian networks from raw data. CaMML stands for Causal MML, because it searches for Bayesian networks according to causal structure, not just statistical equivalence classes, and uses the MML (Minimum Message Length) principle as its metric. It uses the Metropolis algorithm for stochastic sampling.

MML is the theory of Minimum Message Length inference developed by Chris Wallace. CaMML is a *fully* Bayesian way to infer Bayes nets. MML provides the correct information-theoretic penalty for model complexity, and thus the MML prior rewards simpler causal models. For example, among statistically-equivalent models, CaMML prefers common-cause structures to chain structures. Intuitively, the common-cause structure is more likely because it imposes far fewer time constraints than the chain model. For a more detailed explanation of the theory behind CaMML, see [9, 1]. For elaborations including alternative search methods and restricted (logistic) interactions, see [3, 8, 6, 7, 5].

CaMML has consistently outperformed Tetrad, K2, and other standard inference methods. In addition to previous papers, see [2, 3]. We should compare it against some newer alternatives like expand-and-prune learners that are supposed also to be good at finding minimal structure.

### 1.1 A word about versions

There are several versions of CaMML around.

- OldCaMML: C-code originally written by Chris Wallace, and modified by others.
  - Discrete CaMML: handles Discrete Bayes nets. There are few minor variants.
    - \* `classic`: The original version from Chris Wallace. For historical and comparison purposes only.
    - \* `CPT`: A minor variant with CPT info and some cosmetic changes. Now merged into `rodo`, so only historical and comparison interest.
    - \* `rodo`: The object-oriented changes developed in 2001 by Rodney O'Donnell. In 2003, incorporated CPT changes too. Now the default for Discrete `oldcamml`.
    - \* `jules`: Julian Neil wrote an extension that codes CPTs with a logistic dependence model, and so does much better when the CPTs have independencies, as they usually do. Slow. (See [8].)
  - Linear CaMML: handles linear path models.
    - \* `camml1`: Original version. Also sometimes called `rn3(?)`.
    - \* `GA`: Julian Neil replaced the standard search with genetic algorithms.
- CaMML II: A re-implementation in Java, headed by Ph.D. candidate Rodney O'Donnell. Ideally it will incorporate all of the above in a nice modular design. As of the end of 2003, it does much of Discrete CaMML and adds the ability to code CPTs by decision trees. Hopefully available on the website soon. Further development may see the incorporation of missing values, experimental as well as observational data, and latent-variable discovery, and the ability to handle models with both continuous and discrete variables.

This manual covers oldCaMML, specifically *rodo*.<sup>1</sup> As of early 2004, *rodo* is the only version of oldCaMML still being actively maintained. The features of other versions will hopefully be incorporated into CaMML II. The *rodo* version has the following advantages over original CaMML:

**Priors:** Probabilistic constraints may be used to give prior information about the structure of a network, such as probabilities of Arc Presence and the ordering of the network, including Tetrad-style causal tiers.

**Better interface:** This version of Camml contains an easier to use interface than the original (but still far from perfect). The earlier menu options are still available.

**Refactoring:** The original CaMML code has been cleaned and given an object-oriented gloss. Although far from perfect it makes possible some tasks which were previously impossible.

**Netica Interface:** This version is loosely connected to Netica and can save models in Netica format. You may have to download the Netica API from [www.norsys.com](http://www.norsys.com). Please report success or failure to the CaMML list, [camml@csse.monash.edu.au](mailto:camml@csse.monash.edu.au).

**Multiple instances:** Several instances of CaMML may be run simultaneously. Although this is not making things parallel, it gives a convenient way to switch between models currently being worked on. It also gives the ability to tell all instances to search with one command instead of telling several to do the same thing. We shd decide if we want to keep this.

**Weka interface:** This version has been wrapped for Weka by Luke Hope. Some of its features have been modified to better support Weka.

## 1.2 Getting CaMML

CaMML is available under an Academic license, which means it cannot be used for profit or proprietary development. The source code is only available by special request. However, the binaries and documentation are available from:

[www.datamining.monash.edu.au/software/camml](http://www.datamining.monash.edu.au/software/camml)

Bug-tracking, Wiki pages, and further information is available from our CVSTrac server:

[www.datamining.monash.edu.au/cgi-bin/cgiwrap/mdmc/run-cvstrac.cgi/oldcamml](http://www.datamining.monash.edu.au/cgi-bin/cgiwrap/mdmc/run-cvstrac.cgi/oldcamml)

HTML documentation for the original code was once maintained by Sarah George at:

[barrymore.csse.monash.edu.au/~sarahg/camml/docs/index.html](http://barrymore.csse.monash.edu.au/~sarahg/camml/docs/index.html).

Maybe it's still there. Don't read it unless you're going to abide by the terms of the Academic License (available from the website).

We won't cover compiling and installation here. Read the README and INSTALL files, or obtain a precompiled binary from the website.

---

<sup>1</sup>Much of it applies to *CPT* as well, as that is what *rodo* runs as its "inner CaMML".

# Chapter 2

## Quick Start

### 2.1 Using CaMML

After starting CaMML, you will see a brief introductory message with version number and limits (see 2.3.2), and then the menu shown in Table 2.1.

```
-----  
CaMML main menu (ver. 2.35)  
-----  
i: Initialise CaMML                q: Quit CaMML  
1: Set prior on a link             2: Set ordering prior on link  
3: Set up causal tiers            (4: Toggle recalculation flag)  
5: Print link priors              z: Randomize (change seed)  
  
g: Perform Gibbs Sampling         r: Run old CaMML (kill models)  
p: Print final model              c: Print all CPTs  
a: Print default prior matrix     l: Print Link cost matrix  
b: Print recalculated prior       x: Export as Netica network  
  
s: Switch Instance (or F1-F10)    d: Delete Instance  
\: Run System Command            h: Extra Help  
-----  
Instance 1/10 Initialised. Capitalized commands affect all instances.  
-----
```

Table 2.1: CaMML main menu

The main menu is designed for very basic interaction. A typical session is:

1. i: Initialize the instance (specify a data file)
2. 1--3: Add any constraints
3. g: Gibbs sample to find models
4. p: Print the best model
5. c: Print the CPTs also
6. q: Quit

For more control you will want to drop into “old CaMML” before running the Gibbs sampling. More on that later. First let’s walk through a sample session.

## 2.1.1 Initialize the instance

```
Camml> : i
Enter File Name : AsiaCases.100.cas
New Instance 1
```

## 2.1.2 Add priors (soft constraints)

You can add priors on the existence of links between any two nodes, and the ordering between nodes, given that there is a link. You can also set up causal tiers, such that all variables on tier 1 occur before all variables on tier 2, etc. (Causal tiers is an idea borrowed from Tetrad II.)

### Set link prior: 1

Here we specify a likely connection between `VisitAsia` (node 1) and `TB` (node 2), and we know it is not bidirectional.

```
Camml> : 1
Enter Link Numbers 1 : 1
Enter Link Numbers 2 : 2
Enter Probability : .4
BiDirectional link(y/n) : n
Added Prior P( Connection:1,2) = 0.4
```

### Set ordering prior: 2

Any time a model finds a link between nodes  $a$  and  $b$ , it applies the ordering prior to help determine the direction of the arc between them.

So another way to specify that `VisitAsia` should precede `TB` is to put a high prior on that ordering: ??

```
Camml> : 2
Adding prior on Ordering
Enter Link Numbers 1 : 1
Enter Link Numbers 2 : 2
Enter Probability : .9
Added Prior P( 1 ->2 | Connection ) = 0.9
```

I'm not at all sure how this interacts with arc priors, esp. bidirectional ones, or whether it gets applied when there is a directed path between  $a$  and  $b$  but no direct connection.

Bug?

### Set up causal tiers: 3

Causal tiers are an idea borrowed from Tetrad II. They involve setting up a list of tiers such that all values on tier 1 occur before all on tier 2, and all on tier 3 occur after all on tier 1 and 2. They're really just a "script" for batch-setting link orderings. For example, we might believe that `VisitAsia`, `Smoking`, and `TB` all precede `XRay` and `Dysphenia`, but have no other preferences. We would specify two tiers. Each tier gets one line, and we can use either names or numbers.<sup>1</sup> Here, `Smoking` is Variable 3, and `Dysphenia` is Variable 8. (Use human numbers. CaMML will translate for you.) For example:

New format  
(2.35)!

```
Camml> : 3
Number of Tiers : 2
Enter items on tier 0 (separate by space) : VisitAsia 3 TB
Enter items on tier 1 (separate by space) : XRay 8

0 2 1
5 7
Adding prior on Ordering
Added Prior P( 1 ->6 | Connection ) = 0.999
Adding prior on Ordering
Added Prior P( 1 ->8 | Connection ) = 0.999
```

<sup>1</sup>Versions prior to 2.35 required numbers, and required you to terminate with '-1'.

```

Adding prior on Ordering
Added Prior P( 3 ->6 | Connection ) = 0.999
Adding prior on Ordering
Added Prior P( 3 ->8 | Connection ) = 0.999
Adding prior on Ordering
Added Prior P( 2 ->6 | Connection ) = 0.999
Adding prior on Ordering
Added Prior P( 2 ->8 | Connection ) = 0.999

```

### 2.1.3 Find a model: g

Performs Gibbs sampling (a Metropolis search) to find the best model. This procedure starts up the old (Chris Wallace) CaMML, and runs Gibbs sampling from there. You can tell this by noting the Fun: prompts and the last line “Quelling the inner CaMML”, which reports that it is ending that run of the old CaMML. For more details on Gibbs sampling, see Chapter 4.

```

Camml> : g
For menu, type ?
Fun: ---- Starting Annealing ----
Anneal then sample
Cooking
  <T=1.00 P=0.250> (BEST=    281.46)  <T=1.00 P=0.164> (BEST=    281.46)
  .
  .
  .
Prob(arc) = 0.147  weight =    3243.7
Goto getkept
At getkept
printit(fullf)
:::::::::::::::::::: 1-th final model.
  7 arcs, -LogLike  229.57, RawCost  286.04
Same M 30  Becomes N  1
Rel Prior 15.400 Posteriors: SEQ:  0.66  MML:  6.70
Perms          496
Fun:
Quelling the inner CaMML
Camml> :

```

### 2.1.4 View (“Print”) the model

Two print commands show different details about the best model. (Note, you may want to look at alternative models, for which you will need to run the Inner CaMML.)

#### Print the model: p

Shows the number of arcs and some MML cost information, then the prior and posterior probabilities, the number of permutations represented by this model, and finally, a list of arcs using variable names (if provided). The score of interest is the MML posterior, given below as MML: 6.70, indicating that this model did not have a very high posterior probability. (Incidentally, the highest posterior I’ve found on AsiaCases.100.cas is about 11%.)

```

Camml> : p
NumInstances = 0
For menu, type ?
Fun: :::::::::::::: Current model.
  7 arcs, -LogLike  229.57, RawCost  286.04

```

```

Same M 30   Becomes N 1
Rel Prior 15.400 Posteriors: SEQ: 0.66 MML: 6.70
Perms      496
Child      <-      Parent(s)
V VisitAsia <-
V TB       <- Cancer   ~TBorCancer
V Smoking  <-
V Cancer   <-
V TBorCancer <- Cancer
V XRay     <- TBorCancer
V Bronchitis <- Smoking ~
V Dysphenia <- TBorCancer Bronchitis~
Fun:
Quelling the inner CaMML
Camml> :

```

### Print the CPTs: c

Prints out the CPTs for the best model. This was more useful when CaMML did not have the ability to save directly to Netica files. (Then one would log the output and run `camml2dne.pl` on it.) However, it is still good for a quick look at the numbers; Netica's GUI is not very good for that.

The parents of the node appear on the left, and the node's states appear on the right. Probabilities are given to 5 decimal places. If (as often happens), probabilities are still zero at 5 decimal places, they are replaced with .00001. [Check this](#)

```

Camml> : c
CPT of node 1 with 0 dads, Rawcost = 4.79
| 0 1
+-----+
| 0.00495 0.99505
CPT of node 2 with 2 dads, Rawcost = 10.17
4 5 | 0 1
-----+-----
0 0 | 0.06250 0.93750
0 1 | 0.50000 0.50000
1 0 | 0.83333 0.16667
1 1 | 0.00543 0.99457

.
.
.

```

### 2.1.5 Export as Netica network: x

Saves the model as a Netica (.dne) file. Will not do so if you have no active instances. Also, make sure your variable names agree with Netica's rules (no dots, etc) or it will give "Error in NewNode\_bn: Name string passed is bad...."

```

Camml> : x
Enter FileName : nv-null.dne
Netica (AB) 2.15 Linux, (C) 1990-2002 Norsys Software Corp.

```

The license being used is +Site/MonashU/310-1 (security part removed).

```

creating new casefile
Writing cases
reading cases
Saving Network
removing cases

```

### 2.1.6 Quit: q

Quits the program directly.

## 2.2 Other options and concepts

That completes the walk-through. At the top level, there are a few other concepts and options you may wish to explore.

### 2.2.1 Instances

*Rodo* introduced the concept of multiple instances. CaMML can run up to 10 instances, each of which can have its own data set. Alternatively, you could give them all the same data set, and different random seeds or priors, to more completely explore the search space. If you use CAPITAL letters for the commands, they apply to all instances. For example, you could set up several instances and then type “G” to start them all sampling at once.

**Switch Instance: s or F1 ...F10**

```
Camml> : s
Enter Instance Number : 2
```

When you switch instances, CaMML prints the menu and status line telling you whether that instance has been initialized or not.

**Delete instance: d**

Clear the current data, including models and data file.

### 2.2.2 Miscellaneous

**Toggle recalculation flag: 4**

Testing only. Should not be used.

**View arc priors: 5**

This will print a table showing any specified arc priors. It does not show ordering priors. It uses a dot (.) for unspecified priors. By default, unspecified priors between  $a$  and  $b$  are 0.25 in each direction, for an overall arc prior of 0.5. Usually, Bayesian networks are *sparser* than this, so you may want to specify arc priors!

**View default prior matrix: a**

Mostly for testing, but if you want to see the defaults as well as the specified priors, use this one.

### 2.2.3 View recalculated prior matrix: b

After doing Gibbs sampling, CaMML recalculates the overall arc frequencies, changing the default from 0.25 to the overall arc frequency, which is usually much lower. [Look into this code](#)

**View link cost matrix: 1**

Shows the MML cost to add a given link. This is equivalent to the arc priors, but using message length rather than probability. Only calculated after a sampling run.

## Run system command:

Allows the user to run a system command. For example:

```
Camml> : \ls
AsiaCases.1000.cas  CHANGELOG      Netica.h
                  .
                  .
                  .
```

On \*nix systems, you can always suspend CaMML via `Ctrl-Z`, and run the command directly.

## Change random seed: z

Set the random seeds based on a small integer you supply. Without this, every search with the same data (for the same number of steps) will return the same model. Also useful for reproducing a search.

## Menu and Help: ? and h

Typing `?` shows the menu, and `h` shows additional, but largely inadequate, help.

### 2.2.4 Run inner CaMML: r

*Rodo* uses “inner” CaMML for its own work. This option gives you direct access to that inner CaMML, which is largely *CPT* CaMML. You have much more control here, but the menu options are different. They are described in detail in Chapter 3.

**Note:** the inner CaMML does not know about any sampling you have done, nor of any models you have already found. Sadly, it also seems to forget any constraints you have added. Fix?

## 2.3 Things you should know

### 2.3.1 Data Format

CaMML is more forgiving than it used to be. It now allows comments (using the `%` character). CaMML takes data in *case* files, not total-count matrices. Two sample data files are included from the famous Asia network.

- AsiaCases.100.cas has 100 cases
- AsiaCases.1000.cas has (you guessed it) 1000 cases

The format is pretty straightforward, and is given in Table 2.2.

File sample	Explanation
8 100	8 variables, 100 cases
Visit TB Smoke Cancer TBorCanc XRay Bronch Dysp	Variable names (shortened here)
2 2 2 2 2 2 2 2	Each variable has 2 states (0 and 1).
1 1 1 1 1 1 1 1	Case 1: all variables in state two
1 1 0 1 1 1 0 0	Case 2: variables 3,7,8 in state one
⋮	⋮
1 1 1 0 1 0 0 0	Case 100.

Table 2.2: Data file format

The first line lists the number of variables, followed by the number of cases (in fact, these can be separate lines). The next line is optional: it gives the variable names. The one after that lists the number of states for each variable. Following that are the cases, usually one per line although CaMML allows free format.

Older versions of CaMML did not allow for variable names (or comments).

### 2.3.2 Limits

CaMML has some unfortunate hard-coded limits. In the original version, it had a limit of 7 parents per variable, with a max of 5 states per variable, and a maximum number of cells in any variable's CPT of 40,000. The version described here reports the compiled-in limits on startup. For example, my startup screen shows:

```
CaMML: Causal MML for discrete Bayes nets.          Version : 2.20
-----
Original CaMML Code by Chris Wallace;
Modification made by Rodney O'Donnell rodo@csse.monash.edu.au
Limits: 40 states/var, 7 parents/var, 50000 entries/CPT
If CaMML hangs, segfaults, or dies, check these. You may increase the hard
limits by changing MAX_NUM_STATES, MAX_PARENTS, or Maxcell and recompiling.
-----
```

Mind you, CaMML doesn't necessarily *check* to see if you have respected those limits. It might just hang or crash if you disobey.

You should also note that the real constraint is the number of entries per CPT. A 31-state node can have two 40-state parents, and that's it. A 3-state node can have seven 4-state parents. In general you should try to keep the average number of states as low as possible.

Furthermore, CaMML does not currently handle missing values. However, you can circumvent this limitation manually by coding your missing values as a state (I suggest the first state (0) to be consistent). Then you can model missing data explicitly in your final network, and discover whether missing data is itself predictive.

Note: you can significantly speed up your search by reducing the maximum number of parents per node.

# Chapter 3

## The Inner CaMML

By typing `r` at the main menu, you drop down into the “inner” CaMML that *rodo* uses for its own work. Unfortunately, you lose all of your settings and findings except for the data file itself.

### 3.1 A normal session with old CaMML

Table 1 in the Appendix presents the extended menu of commands available when you run old CaMML (by typing ‘`r`’ from the main menu). You can see these commands by typing ‘?’ for the common commands or ‘??’ for the extended commands, each in a 25-line format. Each group of commands is explained in more detail in its own section later.

A typical session with inner CaMML goes like this:

1. `r`: Run the inner CaMML (prompt changes to `Fun:`)
2. `a`: Add any constraints
3. `g`: Gibbs sample to find models
4. Examine the models
  - `pf`: to fully print the best model
  - `pnf`: to fully print all the final models
  - `pmf`: to fully print all the kept models
5. `n` or `m`: Pick a particular model to examine in depth
  - `pf`: to fully print this model
  - `h`: to print the table of arc use frequencies
6. `q`: Quit

We will cover the commands for such a session, roughly in that order. Type ? to get a menu, or refer to Table 1 in the Appendix.

### 3.2 Constraints

- P: Print all constraints
- C: Clear all constraints
- A  $i j$ : Add constraint  $i$  earlier than  $j$
- D  $i j$ : Delete constraint  $i < j$

You may provide CaMML with temporal constraints that variable  $i$  is earlier than variable  $j$ . Then CaMML will ignore all models inconsistent with those constraints, and search only the state space of models consistent with that (partial) temporal ordering.

This is not nearly as powerful as the constraint-specifications available in the outer level. Ideally we should allow those constraints to cross over.

## 3.3 Gibbs Sampling

### 3.3.1 `g`: Do Gibbs sampling starting from current

You may think of `g` as `go!`. The Gibbs sampling procedure is the routine where CaMML actually constructs the causal model for your data. For more details, see Chapter 4.

### 3.3.2 `T`: Change search temperature, cap cost

### 3.3.3 `w` `W`: Search for $W$ times normal steps ( $W$ is float)

This setting is useful or speeding up a search. For example, if I am running a dataset with 200 variables (!), I might just want to verify that I'm getting sensible models at all, before waiting 2 days for a run. In that case, I can set `w` to `.0001`.

## 3.4 Model Input

Specific models may be read in from files. A file may be read in to the current model using `i`, or to the true model using `t`. Either way, CaMML prompts you for a filename which must have a model in CaMML's format.

If you want to start your Gibbs sampling from a particular model, you should read that model in using `i`. You may then examine or modify it before starting the sampling run.

If you know the true model, read it in using `t`. If a true model is specified in this way, CaMML will provide an additional metric for comparing other models, because they can now be measured against the true model.

Why not use this to pass our rodo models in?

### 3.4.1 Model Format

CaMML's model format is similar to its data format in that you must first specify the number of variables and the number of states for each variable. CaMML will check that your model has the same number of variables as the data file, and that each variable has the same number of states. If the check fails you are returned to the main menu. If your model passes these initial checks but is wrong in other ways, it will likely crash CaMML.

After those first two lines, each variable in order lists its parents and, optionally, its conditional probability table (CPT). Table 3.1 explains a small sample model file.

File sample	Explanation
3 16	3 variables, 16 cases
2 2 2	Each variable has 2 states (0 and 1).
-1	Variable 1 has no parents. The -1 signals that no CPT follows.
1 -1	Variable 2 has variable 1 as a parent. Again, no CPT.
1 0 .2 .7	V3 has V1 as a parent. 0 signals CPT follows.

Table 3.1: Model file format

You must signal the end of the parents' list with either a `-1` or a `0`. A `-1` says, "End of the parents list for this variable, and no CPT info follows." CaMML then moves on to the parents list for the next variable, if any.<sup>1</sup>

A `0` signals the end of the parents list and the beginning of the CPT for the current variable,  $V$ . The CPT for  $V$  will be read as  $c$  groups of  $(s - 1)$  states, where  $s$  is the number of states in  $V$ , and  $c$  is the number of combinations of states of the parents of  $V$ . You can think of this as  $c$  rows of  $s - 1$  entries per row, even though you need not format the file in that way (though you can, and for large hand-generated CPTs, it would be a good idea). Because all probabilities in a row of a CPT must add up to 1, you need (and may) only specify  $s - 1$  probabilities. For example, if  $V$  is a binary variable, you need only specify 1 probability per row.

<sup>1</sup>Any negative number will do. For sanity, stick with `-1`.

Finally, the line breaks are not strictly necessary. But it's not like you need to conserve them, and they are necessary for human readability.

## 3.5 Working with Models

### 3.5.1 Selecting models

#### Current, Kept, Final

The list of *kept* models is that list of all the high-probability models found by the Gibbs sampling. The models counted for Gibbs sampling are already simplified representative models rather than actual individual models visited. For more details, see 4 and the papers.

After CaMML is done with the Gibbs sampling, it has a list of kept models. It then performs a more precise analysis of this set of models, to determine if some are indistinguishable given the data. If so, those models are joined into the most representative (simplest) of the bunch. The resulting set is the set of final models. Usually this is the set you want to work with. All the final models will be significantly different from each other. Their message lengths may be close, but they will imply very different causal stories.

By default, the current model is the best of the final models. The model selection commands allow you to change this.

**m k [f] Restore kept model k**

*k* > 0: model *k*

*k* = 0: lowest datacost model

*k* = -1: arc-probability based model

*k* = -2: true model

Makes the *kept* model *k* the current model. You may then analyze or modify this model.

**n k [f] Restore final cleaned and joined model k.**

Makes the *final* model *k* the current model. You may then analyze or modify this model. It works just like **m**, but operates on the final cleaned and joined models.

**m-3 Remove all kept models other than true**

### 3.5.2 Printing models

I'll talk first in Section 3.5.2 about printing general info about the model, including model structure. Then in Section 3.5.2 I will talk about printing the node details like the CPT (conditional probability table).

#### Printing model and structure info: p, pm, pn

The three print commands **p**, **pm**, **pn** all use the same format. All of them can be followed with the optional **f** which adds to the standard report a representation of the arcs in the model. The difference between them is that **p** prints only the current model, while **pm** prints the standard information for all *kept* models, and **pn** prints that info for all *final* models. For example, the output for **pf** might be as shown in Table 3.2. The most important parts are the posterior probabilities. You want them to be high (in this simple case, they were exceptionally so). The two variants are the posterior for the Statistical Equivalence Class (SEQ) and the MML equivalence class (MML). For details on what everything means, see Chapter 4.

If we ask for a *full* report with **f** option (**pf**, **pmf**, **pnf**), we get the final lines beginning with **V** (for 'variable'), which depict the structure of the model. If we do not specify **f**, the report omits the graph structure.

Each structure line begins with a variable such as **V 1**, and gives the parents of that variable. An actual picture can be made from various utilities like **dot**. The graph for the current example is shown in Fig. 3.1.

There are a few flags which can appear on the **V** lines. They are explained in Table 3.3.

```

.....: Current model.
      6 arcs, -LogLike 229.71, RawCost 282.65
Same M 7  Becomes N 1
Rel Prior 24.274 Posteriors: SEQ: 2.11 MML: 7.93
Perms      608
Child      <-      Parent(s)
V VisitAsia <-
V TB       <-
V Smoking  <- Bronchitis~
V Cancer   <-
V TBorCancer <- TB          Cancer
V XRay     <- TBorCancer
V Bronchitis <-
V Dyspnea  <- TBorCancer Bronchitis~

```

Table 3.2: Output of pf on AsiaCases.100.cas

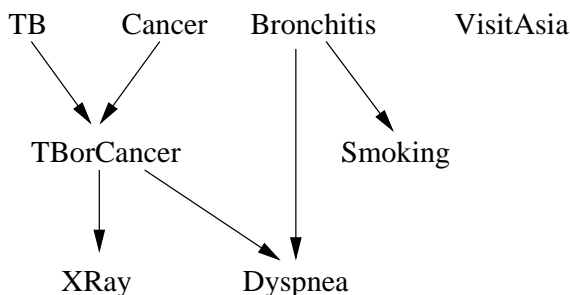


Figure 3.1: Graph of the “current model” discussed in section 3.5.2.

### Printing Node Detail pd, pc

The command `pd n` will print details about node  $n$ , including the number of dads, the `RawCost`, and the *count table* of the number of times the node took a particular value, for all combinations of its parent values. An example of the output of the `pd` command is in Table 3.4. The command `pc n` is like `pd` but prints the *conditional probability table* (CPT). The CPT is estimated from the count table according to the following:

$$\Pr_{i,j} = \frac{N_{i,j} + 0.5}{t_i + 0.5s} \quad (3.1)$$

Where  $\Pr_{i,j}$  is the conditional probability for that entry (row and column) in the CPT,  $N_{i,j}$  is the number of cases for that entry in the CPT,  $t_i$  is the total number of cases for row  $i$ , and  $s$  is the number of states (columns) for the current variable. Therefore, in the common case where there are zero cases for that CPT entry, CaMML assigns a small nonzero entry. For a 4-state variable where the parent configuration never appeared, each entry for that row would be  $\Pr_{i,j} = 0.5/(0.5s) = 1/s$ , which is to say, completely undetermined by the evidence at hand.

If you select  $n = -1$ , so `pc -1`, you will get the CPTs for all nodes. You probably don’t want to do this unless you are logging to a file! An example of the `pc` command is in Table 3.5:

~	The direction of this arc is uncertain.
?	Dubious arc.

Table 3.3: Flags appearing in the V lines.

```

Fun: pd 2
Details of node 2 with 2 dads, Rawcost = 10.166520
  4 5 : 0 1
  0 0 = 0 7
  0 1 = 0 0
  1 0 = 2 0
  1 1 = 0 91

```

Table 3.4: Output of the pd command.

```

CPT of node 2 with 2 dads, Rawcost = 10.17
  4 5 | 0 1
-----+-----
  0 0 | 0.06250 0.93750
  0 1 | 0.50000 0.50000
  1 0 | 0.83333 0.16667
  1 1 | 0.00543 0.99457

```

Table 3.5: Output of the pc command.

### 3.5.3 Modifying models

- fu: Fill in all links  $i$  to  $j$  where  $i > j$
- fd: Fill in all links  $i$  to  $j$  where  $i < j$
- c: Clear all links, giving null model
- a  $i j$ : Add arc from  $i$  to  $j$
- d  $i j$ : Delete arc from  $i$  to  $j$
- r  $i j$ : Reverse arc from  $i$  to  $j$
- s: Simplify current model by deleting insignif arcs
- jba: Reduce (join) kept models using bitmask  $ba$  ( $a$  = relative gain,  $b$  = any order)

All of these commands are self-explanatory except  $s$  and  $j$ . The  $s$ : (*simplify*) command removes dubious (insignificant) arcs from the current model. These arcs are marked with a '?' on the pf printout.<sup>2</sup>

The  $j$ : (*join*) command allows you to forcibly join the kept models according to your desired method. For those of us who do not think in bitmasks, here is what you enter:

- 0: Join by absolute gain.
- 1: Join by relative gain.
- 2: Join by absolute gain, in any order.
- 3: Join by relative gain, in any order.

Note, if you have used  $k$  to cripple the clean & join, then attempting to use  $j$  will tell you that clean & join have been disabled.

### 3.5.4 Analyzing models

#### h: Print table of arc use frequencies

As CaMML samples the model space, it counts how often each individual arc appears. That data is used to generate the arc frequency table. Here is a sample arc-frequency table for Asia.

Table of percent frequency of arcs from Vrow to Vcol									
	based on 200000 sample steps, total weight								
	1934								
1	*	51.1	18.1	11.2	11.0	17.8	15.2	14.5	
2	38.5	*	17.1	18.1	79.2	23.1	34.5	26.9	
3	0.3	0.2	*	2.2	0.9	0.8	16.3	3.4	
4	2.3	10.1	51.1	*	74.0	22.6	30.2	16.9	
5	1.6	16.4	25.4	25.9	*	80.3	17.6	56.8	
6	0.4	1.9	12.0	2.0	4.3	*	17.8	15.3	
7	0.3	0.3	26.5	2.9	0.3	4.5	*	65.0	

<sup>2</sup>OK, I lied. That was pretty self-explanatory too.

8            0.1   0.2   8.6   0.7   0.9   1.8   34.9   \*

The table tells us that an arc from V2 to V1 (2 → 1) appeared about 38.5% of the time, and that an arc from V1 to V2 (1 → 2) appeared about 51.1% of the time. If an arc appears about 50% in both directions, it will be flagged as direction uncertain.

### 1: Print rough distributions of model posteriors

This command shows roughly how many models fell into each broad level of significance for posterior probabilities, and conversely, the number of models you would need to join together to get increasingly high posterior probabilities. For example, on a long unconstrained run over AsiaCases.100.cas, we have:

```
::::: Distribution of model posteriors based on    21000 thousand steps
Numbers of skeletons of various posterior probabilities
>50.0 pc   >25.0 pc   >10.0 pc   > 5.0 pc   > 2.5 pc   > 1.0 pc   > 0.5 pc
          0            0            0            0            7            5            28
Numbers of skeletons needed to reach post-prob:
  10 pc   20 pc   30 pc   40 pc   50 pc   60 pc   70 pc   80 pc   90 pc
      3        6       11    22    38    62   106   197   408
```

## 3.6 Misc

- >: Save results of sampling to file
- <: Restore sampling results from file
- ?: Print this menu
- q: QUIT

## Chapter 4

# How does CaMML Work?

### 4.1 Gibbs sampling

The Gibbs sampling step is where CaMML searches over many models and scores them. In theory, it does this by computing the prior probabilities and likelihoods for all possible models, collapsing them into indistinguishable subsets, and then choosing the most likely subset. In practice, that would take forever.

CaMML uses a Metropolis algorithm to sample the state space of all possible models (limited by any specified constraints). However, rather than just accumulate a simple count of the number of visits to each model, CaMML acts intelligently. Although CaMML walks in real model space, it does not count models. Rather, for every model CaMML visits on its Metropolitan walk, it computes a representative simplification, and *counts* only those representative structures. The reason is that there are many trivial variations which would otherwise be counted as separate models, even though they are statistically indistinguishable. All of the *final* models should be significantly different from each other.

Before the Metropolis search begins counting, CaMML goes through a simulated annealing phase where it does basically the same thing, but with a greater (and steadily decreasing) chance of moving “downhill”. The goal is to escape local optimums that would otherwise trap it, before the real counting begins. Towards the end of the simulated annealing phase, CaMML re-estimates a few useful parameters (like the arc prior) which will figure in the Metropolis search proper.

Knowing how the search works, we can talk about one of the “research only” settings available: crippling.

#### 4.1.1 k *ba*: Cripple using bitmask *ba*: $a = \text{simplify}$ , $b = \text{P(arc)}$

Although the normal user would never want to use this feature, researchers are not normal users. In order to compare CaMML with other algorithms, you can cripple it in various ways. For those of us not used to thinking in terms of masking bits, here’s what you actually enter:

- 0: Cripple nothing (reset to default).
- 1: Cripple simplify only.
- 2: Cripple P(arc) encoding only.
- 3: Cripple both.

If you cripple the simplification step, then CaMML will count each model independently. This will drastically reduce the prior probability for each model, and thus increase each model’s message length. Worse than that, it makes it much less likely that you could find the best structure, for the simple reason that now in order to find it, you must visit **that precise structure**, not just any trivial extension of it. For example, in the classic *Alarm* network, Chris Wallace reports that a crippled CaMML will essentially never find the best structure, though an uncrippled CaMML will do so very quickly.

CaMML also normally uses an adaptive code to represent the presence or absence of arcs. In a sparse network, that means it takes just a fraction of a bit to say that an arc is missing, since you expect arcs to be missing unless told otherwise. Remember, the MML principle is based on having the most efficient code for each hypothesis. If you cripple the P(arc) feature, CaMML is forced to spend 1 bit to represent the presence or absence of *each* arc. That makes the true model more expensive to code, and therefore biases CaMML towards more complicated models, and impairs CaMML’s performance.

## 4.2 What are the scores?

When you print a model, you get some cryptic summary information like:

```
:::::::::::::::::: Current model.
6 arcs, -LogLike 2492.78, RawCost 2726.34
Same M 1   Becomes N 1
Rel Prior 1.000 Posteriors: SEQ: 97.78 MML: 97.78
Perms          4
```

Obviously it starts by telling us the number of arcs in the current model (6 in this case). The next two numbers on the first line represent the costs for this model.

`-LogLike` is the negative log of the likelihood of the data given the model. It is a measure of how well the model fits the data. In particular, it is the *message length* of the data, given the model, which is the second part of an MML message, and shorter is better.

`RawCost` is the total cost (message length) of specifying the model *and* the data given the model, but *without* regard to the fact that this model may be equivalent to other models. It is the `RawCost` which governs the Gibbs sampling process. The user will be more interested in the posterior probabilities, which take heed of which models are equivalent. (For example, two models having almost identical `RawCost` may nevertheless have vastly different posteriors.)

The `RawCost` is provided mostly for diagnostic purposes. Of course what it estimates is the total message length, and that is the heart of MML encoding and inference. More complex models might explain the data better, but will have a higher model cost. If they are too complex, they will have a higher *total* cost than a simpler model, even if that simpler model does not explain the data as well. As the name MML implies, the best model class will be the one with the *minimum* overall message length.

The `RawCost` displayed (at least in Discrete classic and rodo) is not the actual raw cost:

$$\ln(k!) + M \ln(P_a) + \left( \frac{k(k-1)}{2} - M \right) \ln(1 - P_a) + \text{datacost}$$

It is instead the over-estimate:

$$M \ln(P_a) - M \ln(1 - P_a) + \text{datacost}$$

The cost reduction  $\left[ \ln(k!) + \frac{k(k-1)}{2} \ln(1 - P_a) \right]$  is constant for any given arc probability  $P_a$ , and as we are only interested in differences of the MML score, CaMML does not compute it.<sup>1</sup>

The second line tells us something about the place of this model in the *kept* and *final* hierarchy. `Same M m` means that in the final cleaning and joining, the current model was judged to be same as model  $m$ . `Becomes N n` means that after the final filtering (clean and join), this model became the  $n^{\text{th}}$  final model. If three *kept* models 2, 4 and 7 both reported `Same M 4`, then they would all be collapsed to the same final model N, and so would all report something like `Same N 1`. When several models collapse in the final filtering, you may sometimes see `Part of N n` as well as `Same N n`.

The third line reports the relative prior of the model and two posterior probabilities. The relative prior is the prior probability of this model compared to the prior probability of the best MML TOM. In MML, the simplest models have the shortest encoding and therefore the highest priors.

The SEQ posterior is the estimated probability that the true model is in the statistical-equivalence class of this model. The MML posterior is the estimated probability that the true model is in the MML equivalence class of this model. The MML equivalence class will always be larger than (or equal to) the SEQ class, so the MML posterior will always be greater than (or equal to) the SEQ posterior.

Remember that the posterior probabilities are really estimates, because they are derived from a Gibbs sampling process over all the possible models (and all possible parameters of those models). If you have doubts about the reliability of the estimates, vary the length of the sampling process or the temperature.

`Perms` is the number of permutations of variable orderings consistent with generating the current model. For instance, if the model is  $A \rightarrow B \rightarrow C$ , then there is only one permutation,  $ABC$ . However, if it has a common-cause structure  $A \leftarrow B \rightarrow C$ , it has two permutations, namely  $BAC$  and  $BCA$ .

<sup>1</sup>This may lead to `RawCost` having strange (maybe even negative) values for nasty cases such as each variable having one state, and  $\text{arcProb} > 0.5$ . The metric is still valid though, as adding the cost reduction would return them to more sensible numbers, and we only care about the difference.

If you have specified a true model, you also get an `Expected -LogLikelihood` measure.

Finally, remember that the model shown is really the *representative* model for that MML equivalence class, in the same way that “0.5” may represent the range [0.25, 0.75).

## 4.3 Search Algorithm

I keep having to go back to the paper to figure out just how CaMML works, so I thought I would try to write down a step-by-step procedure here. However, Rodney noticed that while the paper describes Linear CaMML, Discrete CaMML has evolved quite a bit since then. Rodney kindly revised this section. So we will first present the algorithm in the paper (and used by Linear CaMML), and then introduce the more complex schemes used in Discrete CaMML.

### 4.3.1 Linear Search Algorithm

Here is a simplified version of sections 6–8 of [9]. An even simpler schema for this algorithm is now given in chapter (7?) of [4].

```
main () {
    Set hot = 3.0
    Set temperature = hot
    k3 = max(k^3, 10^3)

    // Standard Simulated Annealing search
    cook( 20*k3 )           // cooking
    for ( i = 1; i < 20; 1++) { // cooling
        temperature = hot - (i/20)
        cook( 2 * k3 )
    }

    // Long cook session to get good idea of arcProb
    temperature = 1.0
    cook( 40 * k3 )
}

// Sample N models keeping track of the average
cook( N ) {
    totalArcs = 0
    Repeat N times:
        Pick one of the three mutators: do it
        Calculate the change in LJP (dLJP) for this new TOM
        Accept the mutation iff either:
            1) dLJP is positive, or
            2) ( exp (dLJP) / Temp ) > uniform(0,1)
        If accepted, update LJP, C, and T, link matrix.
        totalArcs = totalArcs + tom.numSignificantArcs

    averageArcs = totalArcs / N
    arcProb = (averageArcs + 0.5) / (maxArcs + 1.0)
}
```

### 4.3.2 Discrete Search Algorithm

The discrete version of classic CaMML algorithm has numerous changes. Although they do may not change the spirit of the algorithm, there is a good chance they could have a large impact on performance.

I suspect there are several reasons for the code changes:

- Further development of the algorithm (Fill and Clear in anneal search).
- Fiddling with parameters
- Optimization for discrete data (Estimate arcProb from best model saves cleaning every model)
- etc.

### Anneal phase

```

anneal () {
  temperature = 1.0
  k3 = max(k^3,10^3)

  cook( 7 * k3 )

  // fill
  Repeat 10 times
    Clear all arcs
    Randomize Total Ordering
    Add all possible arcs to network(limited to 7 parents per node)
    cook( 7 * k3 )

  // clear
  Repeat 10 times
    Clear all arcs
    cook( 7 * k3 )

  // Anneal
  hot = 2.0
  temperature = hot
  for ( i = 1; i < 12; i++ ) {
    temperature = hot - (i / 12);
    cook( 10 * k^3 )
  }
  temperature = 1.0
  cook( 10 * k^3 )
}

// Perform N mutation steps searching for the best TOM.
// Recalculate arcProb based on best model
cook( int N ) {
  Repeat N times:
    Pick one of 4 mutations: do it. (fourth involves switching a parent of a node)
    Calculate the change in LJP (dLJP) for this new TOM
    Accept the mutation iff either:
      1) dLJP is positive, or
      2) ( exp (dLJP) / Temp ) > uniform(0,1)
    If accepted, update LJP, C, and T, link matrix.

  clean the best TOM found
  arcProb = (cleanBestTOM.numArcs + 0.5) / (maxArcs + 1.0)
}

```

### Sampling phase

```

Begin with the final model from Anneal phase.
Temperature = 1.8
k3 = max(k^3,10^3)

```

```

Repeat 200000 k3 times:
Pick one of the mutators (respecting priors): do it
Calculate dLJP
If (dLJP > 0) or (exp (dLJP) / Temp > uniform(0,1)):
  * accept change, update LJP, C, T, link matrix
  * update arc frequency table
  * compute the MML model M' for this TOM:
    - clean the TOM (remove insignif. effect arcs)
    - compute the skeleton S and likelihood L for the clean TOM
  * Increment the visit count for M':
    - h1 = hash S and L using symmetric random matrix 1
    - h2 = hash S and L using symmetric random matrix 2
    - increment visit count at indices h1 in H1 and h2 in H2
  * Calculate weight of current TOM relative to best TOM (best TOM has weight 1.0)
    - weight = exp((bestMML-currentMML)/temperature)
  * Add weight to visit count for M's DAG
    - h3 = hash of S using asymmetric randmatrix a
    - h4 = hash of S using asymmetric randmatrix b
    - Add weight to visit count at indices h3 in D1 and h4 in D2
    - Add weight to totalWeight
  * If the SMALLER visit count for M' is now in the top 50,
    replace one of the top 50 MML models with M' [??]

```

### Cleaning phase

```

Further clean the (up to 50) retained models
  * Use a more accurate Fisher Info, and merge any
    which become the same
  * Then test for MML equivalence using KL-based
    measure of expected change in msglen: if < 0,
    merge. (Requires reestimation of priors by
    inverted Bayes formula.)

```

```

Posterior prob = sum(weights of represented TOMs) / totalWeight

```

# Bibliography

- [1] H. Dai, K. B. Korb, C. S. Wallace, and X. Wu. A study of causal discovery with weak links and small samples. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1304–1309, San Francisco USA, 1997. Morgan Kaufmann Publishers Inc.
- [2] R. J. Kennett, K. B. Korb, and A. E. Nicholson. Seabreeze prediction using Bayesian networks. In *PAKDD'01 – Proceedings of the Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 148–153, Hong Kong, 2001.
- [3] K. B. Korb and D. Boulton. An empirical study of minimality and causality. Technical report, School of Computer Science and Software Engineering, Monash University., 2003.
- [4] Kevin B. Korb and Ann E. Nicholson. *Bayesian Artificial Intelligence*. CRC Press, 2003.
- [5] J. R. Neil and K. B. Korb. The MML evolution of causal models. Technical Report 98/17, Dept of Computer Science, Monash University, Clayton, Victoria 3168, Australia, July 1998.
- [6] J. R. Neil and K. B. Korb. The evolution of causal models: a comparison of Bayesian metrics and structure priors. In N. Zhong and L. Zhou, editors, *Methodologies for knowledge discovery and data mining: third Pacific-Asia conference*, pages 432–437. Springer Verlag, 1999.
- [7] J. R. Neil and K. B. Korb. The evolution of causal models: a comparison of Bayesian metrics and structure priors. Technical Report 98/17, Dept of Computer Science, Monash University, Clayton, Victoria 3168, Australia, July 1999.
- [8] Julian R. Neil, Chris S. Wallace, and Kevin B. Korb. Learning Bayesian networks with restricted causal interactions. In *Uncertainty in Artificial Intelligence 1999*, pages 486–493, Stockholm, Sweden, 1999.
- [9] Chris S. Wallace and Kevin B. Korb. Learning linear causal models by MML sampling. In A. Gamerman, editor, *Causal Models and Intelligent Data Management*. Springer-Verlag, 1999.

<b>Constraint Commands</b>	
P	Print all constraints
C	Clear all constraints
A $i j$	Add constraint $i$ earlier than $j$
D $i j$	Delete constraint $i < j$
<b>Gibbs Sampling Commands</b>	
g	Do Gibbs sampling starting from current
k $ba$	Cripple using bitmask $ba$ ( $a$ = simplify, $b$ = P(arc))
w $W$	Search for $W$ times normal steps ( $W$ is float)
T	Change search temperature, cap cost
<b>Model Input Commands</b>	
i	Read in a specification and set as current model
t	Read in a specification of true model
<b>Model Selection Commands</b>	
m $k$ [ $f$ ]	Restore kept model $k$ (Current options: <i>available indices</i> or “-none-”) (0: lowest datacost, -1: arc-prob based, -2: true) Adding $f$ means show the arcs.
n $k$ [ $f$ ]	Like ‘mk’, but use final cleaned and joined models
m-3	Remove all kept models other than true
<b>Print Commands</b>	
p [ $f$ ]	Print the current model. $f$ prints full details (not CPT).
pm [ $f$ ]	Print all kept models. $f$ prints full details (not CPT).
pn [ $f$ ]	Print all final models. $f$ prints full details (not CPT).
pd $n$	Print Details (CPT ‘counts’) for node $n$
pc $n$	Print CPT for node $n$ . ( $n = -1$ , print for all nodes)
<b>Modify Commands</b>	
fu	Fill in all links $i$ to $j$ where $i > j$
fd	Fill in all links $i$ to $j$ where $i < j$
c	Clear all links, giving null model
a $i j$	Add arc from $i$ to $j$
d $i j$	Delete arc from $i$ to $j$
r $i j$	Reverse arc from $i$ to $j$
s	Simplify current model by deleting insignif arcs
j $ba$	Reduce kept models using bitmask $ba$ ( $a$ = relative gain, $b$ = any order)
<b>Analysis Commands</b>	
h	Print table of arc use frequencies
l	Print rough distributions of model posteriors
<b>Misc.</b>	
>	Save results of sampling to file
<	Restore sampling results from file
?	Print basic menu
??	Print extended menu
q	QUIT

Table 1: Inner CaMML menu combined and reorganized