

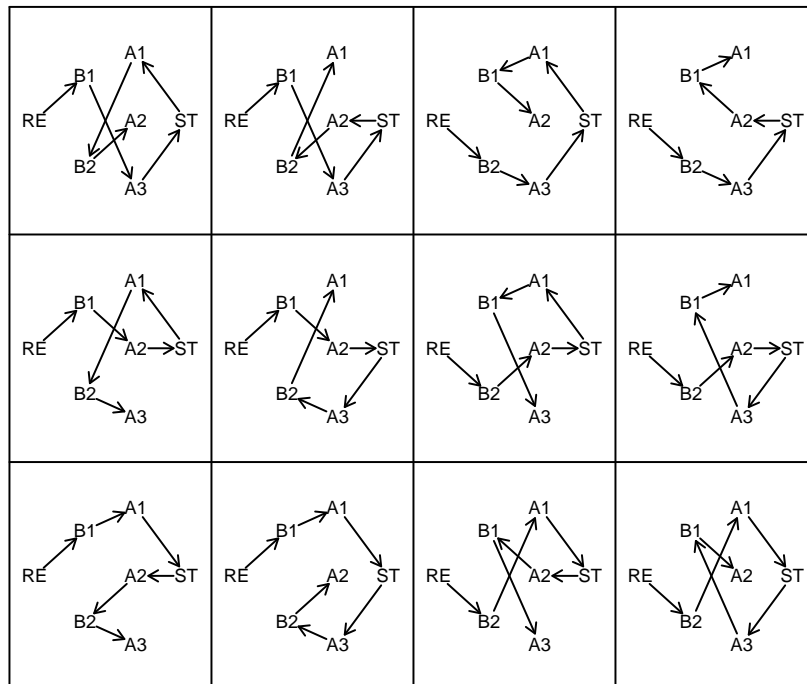
# rebastaba Project

J.-B. Denis\*, M.-L. Delignette-Müller and R. Pouillot

09\_06\_24

rebastaba means *réseaux bayésiens traités par statistique bayésienne*, that is *Bayesian networks tackled with Bayesian statistics*. It is freely available from [9].

The aim of this note is to give the reader the non technical but nevertheless compulsory keys to understand for what the rebastaba project [/rbsb/] is intended and which way it is conceived. Any kind of reaction to this project is welcome.



\*corresponding author: Jean-Baptiste.Denis@Jouy.Inra.Fr

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Aims</b>	<b>5</b>
2.1	for who . . . . .	5
2.2	basic objects . . . . .	6
2.3	basic actions . . . . .	6
2.4	principle: safety first . . . . .	6
2.5	future . . . . .	6
<b>3</b>	<b>What does mean Bayesian networks in ReBaStaBa context</b>	<b>7</b>
3.1	framework . . . . .	7
3.2	node types . . . . .	7
3.3	probability distribution types . . . . .	9
3.4	Bayesian network types . . . . .	9
3.5	restrictions on parentship . . . . .	9
<b>4</b>	<b>Main ReBaStaBa features</b>	<b>10</b>
4.1	Different kinds of nodes . . . . .	10
4.2	Studying the paper of a particular node . . . . .	11
4.3	Defining nodes and /bn/ from a text file . . . . .	11
4.4	Some statistics . . . . .	11
<b>5</b>	<b>General points</b>	<b>11</b>
5.1	how to start . . . . .	11
5.2	Bayesian networks and their structure . . . . .	12
5.3	domains for random variables . . . . .	13
5.4	distinguishing nodes and variables . . . . .	13
5.5	assembling nodes to form a /bn/ . . . . .	14
5.6	playing with a /bn/ . . . . .	14
5.7	organization . . . . .	17
<b>6</b>	<b>More details</b>	<b>20</b>
6.1	defining a node . . . . .	20
6.2	repeated nodes . . . . .	23
6.3	multivariate nodes . . . . .	23
6.4	data based distributions . . . . .	24

6.5	Parentship . . . . .	25
6.6	introducing pieces of code into a /bn/ . . . . .	25
6.7	providing consistent simulating functions . . . . .	26
6.8	how /rbsb/ incorporates a new node into a /bn/ . . . . .	26
6.9	Links with other packages . . . . .	27
6.10	/rbsb/ constants . . . . .	28
6.11	miscellaneous . . . . .	28
6.12	statistics . . . . .	28
<b>7</b>	<b>Toy example</b>	<b>28</b>
<b>8</b>	<b>Developments</b>	<b>29</b>
8.1	documentation . . . . .	30
8.2	construction and manipulation of objects . . . . .	30
8.3	displaying objects . . . . .	32
8.4	exploiting objects . . . . .	33
8.5	interfacing with other applications . . . . .	33
8.6	computing . . . . .	34
8.7	programming . . . . .	34
<b>9</b>	<b>Terminology</b>	<b>35</b>
<b>10</b>	<b>Remerciements</b>	<b>36</b>

# 1 Introduction

To answer some commitments taken towards several collaborators, calculations and learning involving Bayesian networks [bn/] had to be performed. Among the tools that we know and dispose, a conclusion is that Bugs<sup>1</sup> represents the core of what is needed. Bugs can deal with a great variety of bn/, allowing a large spectrum of distributions and mixing discrete and continuous variables<sup>2</sup>. If Bugs is perfect for simple models, it becomes quite difficult<sup>3</sup> when the number of variables is numerous and/or when multinomial variables are involved<sup>4</sup>. So we would be pleased to be able to deal networks with tools and structures in R<sup>5</sup> deriving automatically models according to Bugs code syntax. This was the first incentive for starting /rbsb/. Furthermore, along its use in different projects many other aims were defined and partly satisfied...

Among them, tools to simulate from defined bn/, to manipulate associated directed graphs, to extract subbn/... were derived. A great advantage is to have at hand the power of programing through /R/, including the access to other packages devoted to bn/ like are **deal** package and **grappa** pseudo-package.

Some /R/ packages related to bn/ or connected with Bugs softwares already exist. Among them:

**R2WinBugs** is a collection of functions to call WinBugs (then under MSWindows only) from /R/, replacing clicking by call functions. It does not comprise analyses of the bn/.

**BRugs** is the equivalent of R2WinBugs for OpenBugs. As far as we understood<sup>6</sup>, an interface under /R/ to replace clicking by coding but without adding new features. Seems similar in spirit to R2WinBugs, even if much more integrated.

**RJags** a transposition of Jags into /R/ environment.

**BayesMix** is an interface for Jags but for the limited class of bn/ describing a finite mixtures of univariate distributions from a statistical point of view.

**BayesCount** the same but for count data (includes some zero-inflated models).

**deal** is mainly orientated to the learning of the bn/ structure. To do so efficiently, it restricts the bn/ distributions to mutinormality for continuous variables and multinomiality for discrete variables.

**bnlearn** also orientated to the purpose of learning : a package to be studied...

**grappa** is a suite of functions (like a package but not registrated as such) in /R/ for calculating marginal and conditional probability distributions on collections of discrete variables when all marginal and conditional probabilities defining the bn/ are numerically specified. It does part of the job of the Hugin system.

---

<sup>1</sup>By Bugs it is meant OpenBugs (WinBugs, LinBugs) or Jags.

<sup>2</sup>Continuous variables are important for parsimony purpose.

<sup>3</sup>specifically to be sure that you are running the model to which you dreamt.

<sup>4</sup>Because, one has to consistently introduce many conditional probability tables.

<sup>5</sup>similar to those of package *deal*

<sup>6</sup>For the moment, only available under MSWindows.

For a complete view of /R/ linked development, have a glance to the page of Jong Hee Park on the CRAN<sup>7</sup>.

/Rbsb/ is aimed on the manipulation and the analysis of general /bn/s with connections with more specialized systems, particularly Bugs softwares. To do so safely and easily, /R/ objects and associated methods (functions) are provided. This first step is no more that putting up the scenary. It is being done, keeping in mind that the proposed tools must be restricted to specific calculations not already available somewhereelse.

For the moment, the /rbsb/ project is finishing the phase of establishing the main structures, and polishing the definition of models. Its present main possibilities are the representation of the graph, the study of the parentship and simulation. Limited possibilities of translation into Bugs code are/were available. Access to grappa algorithms is implemented.

## 2 Aims

### 2.1 for who

An important point is to decide *for who* this programming project is intended. Honestly, we must answer firstly for us. It gives us the possibility to manipulate /bn/s and a good way to deepen our understanding of what they are. Nevertheless, from the beginning, we made the necessary efforts to make /rbsb/ accessible to other people, implementing systematic checks of the objects, providing extensive documentation and examples, adding the possibility to define the /bn/ by means of structured text files. In this spirit, different levels of use are distinguished:

**Exploiting:** manipulating /bn/ already constructed, getting more outputs (printing, plotting,...) from them with f3-functions<sup>8</sup>. Of course, even for this basic level two prerequisites remain: to know something about /bn/ theory and to have a minimal familiarity with R.

**Constructing:** elaborating new /bn/ with the help of prepared f2-functions<sup>9</sup> to create, modify them and generate bugs code...,

**Programming:** programming on /bn/ components with deeper f1-functions<sup>10</sup>,

**Improving:** proposing additionnal new functions or improving already existing functions. This means modifying /rbsb/ code.

Besides to /rbsb/, specific functions were added convenient general purpose functions (f0-functions<sup>11</sup>).

Another important characteristics of /rbsb/ must be underlined. It is adapted for *tap-tap* addicted not for *klik-klik* addicted. There is no friendly *GUI*, only /R/ functions, even if for all functions a maximum of default arguments are provided. The only exception is `modify4gn` and its future relatives to position into  $\mathbf{R}^3$  the nodes of a graph, to add/remove arcs, to add/remove nodes...

---

<sup>7</sup>From France : <http://cran.cict.fr/web/views/Bayesian.html>

<sup>8</sup>f3 functions are in the `rebastaba.f3.r` file.

<sup>9</sup>f2 functions are in the `rebastaba.f2.r` file.

<sup>10</sup>f1 functions are in the `rebastaba.f1.r` file.

<sup>11</sup>stored in the `rebastaba.f0.r` file.

## 2.2 basic objects

The first idea is the construction of `/R/` objects representing `/bn/s` and associated structures. The basic objects are specified by means of new S4 classes. The central object is `bn` for bayesian networks. See §5.7.1 for other objects. Using 4th generation of S-objects, this is defined through `setClass` instructions to be found in file `rebastaba.1a.r`, the types of the slots are defined and automatically checked within calculations by `/R/`: this is a strong advantage.

## 2.3 basic actions

`/R` offers a collection of tools to investigate the properties of `/bn/s`. Main actions are:

- creating `bn` objects representing `/bn/` (this is probably the most complicated point for the standard user),
- exploring and displaying the structure of the graph (*e.g.* the genealogy),
- manipulating `/bn/s`, for instance to get sub-`/bn/` or to modify interactively an initial `/bn/`.
- simulating data sets from the joint probability distribution, without or with conditioning about some nodes<sup>12</sup>,
- performing some statistical descriptions<sup>13</sup> of the generated data sets.

These actions are performed for univariate, repeated and multivariate nodes, distinguishing categorical and numerical variables...

## 2.4 principle: safety first

There is no doubt that `/rbsb/` cannot be more than a prototype. Probably much more time will be spent programming on it and with it than computing with it. So the principle is to facilitate the safety of programmation rather than the speed of computation introducing systematic checks. Check of the objects is done when entering each function by means of some functions (`check3rbsb` is the main one). This implies repeated useless checkings but avoid some painfull research of errors. If necessary, to prevent the checkings, one can replace these functions with a function doing nothing as does `nocheck`. Another important advantage of such a strategy is the safety provided when changing object definitions.

## 2.5 future

Some additional tools to explore the `/bn/` properties will be progressively added, some ideas are listed in §8.

A major aim remains to produce from `/bn/` objects some Bugs code. Albeit incomplete, something was started. The next future could be the extension to most of the available distributions

---

<sup>12</sup>By itself `/rbsb/` cannot conditioned on non ancestor nodes (that is nodes with parents). To do so, it must appeal to other software where *ad hoc* algorithms are implemented.

<sup>13</sup>Of course, using `/R/` facilities and only when the specificity of `/bn/` is involved.

in Bugs model description. For the moment, it is not clear that the translation the other way round, from Bugs to /rbsb/, should deserve attention<sup>14</sup>.

*Grappa* algorithms were<sup>15</sup> introduced for /bn/s comprising only numeric multinomial<sup>16</sup> *categoric* nodes. In a next version a connexion with learning algorithms could be included either with *deal* or *bnlearn* package.

### 3 What does mean Bayesian networks in ReBaStaBa context

#### 3.1 framework

/Bn/s are strictly understood as a convenient<sup>17</sup> way to define the joint probability distribution for a set of random variables using repeatedly conditional probabilities and the Bayes' theorem. More precisely, let  $\{A, B, C, \dots, Z\}$  the set of random variables considered in that order, the joint probability is defined by

$$[A, B, C, \dots, Z] = [A] \cdot [B | A] \cdot [C | A, B] \dots [Z | A, B, C, \dots, Y] \quad (1)$$

where  $[X_1, X_2, \dots, X_p | Y_1, \dots, Y_q]$  stands for the joint probability of  $\{X_1, X_2, \dots, X_p\}$  conditionally to  $\{Y_1, \dots, Y_q\}$ . Equation (1) applies to any joint probability, most often some simplifications occurs for specific /bn/. As an example

$$[A, B, C, D] = [A] \cdot [B | A] \cdot [C | A] \cdot [D | B, C]$$

is associated to the directed acyclic graph<sup>18</sup> shown in Figure 1. In /rbsb/ this graphes are associated to /gn/ objects, which are not limited by the acyclic property and arcs between nodes can be decorated in several ways...

#### 3.2 node types

Then each node represents a random variable. But there is many types of random variables to associate them consistent distributions and parentships when building the /bn/ up. In /rbsb/ this is done through different traits:

**dimension** The dimension of a node is the number of scalar random variables comprised within the node. Most of them are unidimensional. Of course, it is theoretically possible to break down a multidimensional variable  $(X_1, X_2, \dots, X_p)$  into a sequence of conditional variables  $\{(X_1), (X_2 | X_1), \dots, (X_p | X_1, X_2, \dots, X_{p-1})\}$  but this is not always convenient because some symetry can be lost, because the order to choose is not obvious, because we are used to some standard multivariate distributions. In /rbsb/ three types of dimensions are distinguished *univariate*, *repeated* and *multivariate*. Univariate nodes are scalar and the unique variable will get as name the name of the node. Repeated nodes are vectorial

---

<sup>14</sup>And this will ask a tremendous effort of parsing!

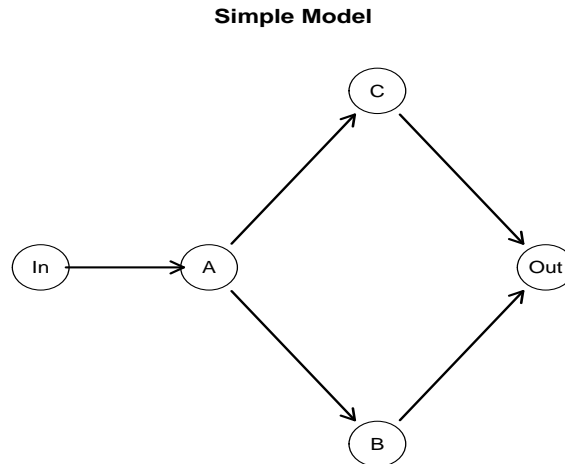
<sup>15</sup>very easily

<sup>16</sup>One can imagine conditional probability tables based on continuous covariables.

<sup>17</sup>easy, interpretable, parsimonious, partially graphically representable...

<sup>18</sup>So called *dag*.

Figure 1: A simple model comprising one input, one output linked through three hidden variables.



(including scalars) and, as a standard choice, the variables will get for names the name of the node followed by their sequence number between brackets; e.g. the repeated node  $X$  of dimension three will have as variables:  $X[-1-]$ ,  $X[-2-]$  and  $X[-3-]$ ; an important property of these variables which partly justify the node qualification is that they are conditionally independent from the parent nodes, independent but not compulsorily identically distributed, also they must have the same nature (see below). Multivariate nodes are vectorial (including scalars) and the variables will get as names the name of the node followed by any other name between brackets. E.g. the multivariate node  $Y$  of dimension two can have as variables:  $Y[one]$ ,  $Y[two]$ , standard names are  $Y[\sim 1 \sim]$ ,  $Y[\sim 2 \sim]$ ; no constraint about their joint distribution is supposed to stand.

**nature** The nature of a node (more precisely of a node variable since the natures can be different in the case of multivariate nodes) is the kind of value they can get. The main three natures are *continuous*, *integer* and *categoric*. Of course there are some links between the nature and the distribution : a normal is continuous, a binomial is integer, a numcat (see below for a definition) is categoric.

**domains** The probability distribution bears by itself the domain where the random variable can take values. Nevertheless in practical situations, it is important to specify according to the application the variation domain of a variable. For instance we can admit that the wheat yield follows a normal distribution not that it can take negative values. In /rbsb/ three domains are defined for each variable : the possible domain (pod) (when values are drawn outside it they are systematically replaced with a  $\mathbf{NA}$ ); the representation domain (red) (when summaries and graphical representation are performed only this domain is retained; the common domain (cod) that is the domain where the expert thinks a priori that the variable have to be found (this is a good way to detect bad behavior of a numerical model). The (pod) must always be given by the user ; other domains can be automatically provided from it.

### 3.3 probability distribution types

To each node is associated a probability distribution, of course there is some relationships between the node type and its probability; most of them are checked by `/rbsb/`. Each distribution belongs to one of the following category:

**conti\_scalar** For scalar or repeated node of continuous univariate distributions. Their parameters are given in a scalar way (*normal, uniform, beta, ...*).

**discr\_scalar** For scalar or repeated node of discrete univariate distributions. Their parameters are given in a scalar way (*Bernoulli, binomial, Poisson, ...*). The main difference with the **conti\_scalar** category is that no second step transformation is allowed.

**conti\_vector** For multidimensional continuous distributions, not repeatable and the arguments can be vector of different dimensions (*multinormal, Dirichlet, ...*).

**discr\_vector** For multidimensional discrete distributions, not repeatable and the arguments can be vector of different dimensions (*multinomial, ...*).

**categoric** For multinomial of size one, associated to factor variables.

**data\_based** For empirical drawing from a `data.frame` (*empidata, popula*).

**misc** A miscellaneous category comprising the **Dirac** distribution, thought as a univariate deterministic node, and also **easyp** and **program**, two very flexible node types since defined through the use of some `/R/` code.

### 3.4 Bayesian network types

For resorting to external possibilities of `/rbsb/`, it is necessary to check if a given `/bn/` satisfies some properties. For instance a `/bn/` can be

**grappa** compatible, that is the computation of some probability tables can be done using Grappa [3] algorithms.

**bugs** compatible, that is the `/bn/` can be translated into a file defined the model in a way usable by Jags [6].

**deal** compatible, that is the `/bn/` can be translated into a `deal` [2] object.

**lin\_normal** compatible, that is the `bn` defines a linear multinormal distribution from a set of scalar nodes.

Not all of them are already implemented.

### 3.5 restrictions on parentship

Through the use of **program** links, every kind of `/bn/` can theoretically be defined. Nevertheless, the use of type of probabilities implies some constraints in the direct parentships between the nodes. The main ones are

??? là je ne sais plus ce que je voulais dire ??? Mais il faudrait au moins exposer comment les parentés se déclinent entre noeuds et variables intro noeuds, ce que l'on peut percevoir par `print(bn,«nv»)`...

## 4 Main ReBaStaBa features

### 4.1 Different kinds of nodes

In /rbsb/, nodes can have very different types according to several aspects.

#### A first distinction is the dimension of a node:

- A node can comprise a unique variable, it is said univariate (then unidimensional),
- A node can comprise a collection of variables independent conditionally to their parents (then multidimensional); the simple example is a set of i.i.d variables but this type also comprise variables having distinct parameter values; these nodes are called repeated nodes; the nature is identical for all the variables of such nodes; the number of repetitions can be one; when generated by /rbsb/ the name of variables for repeated nodes is the number sequence starting from 1 surrounded with hyphens<sup>19</sup>,
- A node can also be multivariate, as are multinormal, Dirichlet and so on probability distributions; the natures can be different between the variables of such a node; the number of variables can be one.

When the node is univariate, the name of the variable is generally the node name itself, when it is multidimensional the variable names are always a composition of node name and additional variable names. For instance `STATURE[height]` and `STATURE[weight]` for the node «stature» with two variables «height» and «weight». A consequence of this classification is that only univariate distributions can be repeated for /rbsb/.

The way to find the dimension of a node is to use the number of variable names and for that reason, the unnamed variables have in fact, internally, the name «» not `character(0)`.

#### A second distinction is the so-called nature of the variables :

- `conti`: for continuous variables,
- `integ`: for integer variables,
- `cateo`: for ordered categoric variables (not used for the moment),
- `categ`: for ordinary categoric variables,
- `unkno`: for other types of nature (not used for the moment).

The nature of a variable determines some of the properties and also the way they are dealt by /rbsb/. For instance Pearson correlations are computed between two continuous variables but frequency tables are proposed for categorical variables.

---

<sup>19</sup>But this is not a constraint.

**A third distinction is the probability distribution associated to the node** The variety of implemented distribution is quite small but at the end all distributions available in `/R/` might be included. To see which are present, just look at the result of `help81type("argu")` [see Table 2]. In fact, every kind of distribution can already be used since a self-programing possibility is offered through the node type `program`. It is also worth mentioning that empirical (draws from a `data.frame`) are implemented either at random (`empidata`) or in a systematic way (`popula`).

## 4.2 Studying the paper of a particular node

To study the role of a given node, `/rbsb/` provides the possibility to extract the sub-bn pruning all ascendance of the node and putting it some given, for instance uniform, probability distribution. This allows, simulating with the created `/bn/`, to see the «pure» effect of this node on its descendance.

## 4.3 Defining nodes and `/bn/` from a text file

The standard way to use `/rbsb/` is to write some `/R/` scripts. But to help new users of `/rbsb/` who are also new<sup>20</sup> users of `/R/`, an additional way of introducing `/bn/` with structured text files was added. This affords also the possibility to make computations with `/bn/` resulting from programs possibly not written with `/R/`<sup>21</sup>, an application we have in mind is the study of `/bn/` having a random structure.

A drawback to be eliminated<sup>22</sup> of this external definition is that `EASYPROMMING CODES` (see below) `CANNOT COMPRISE SPACE CHARACTERS` since they are the separator between list items... So easy to be stuck this way, believe us!

## 4.4 Some statistics

Presently `/rbsb/` comprises 31 coding files for about 20200 lines of which more of 9000 are main comment lines. Fourteen S4 classes are defined as well as 224 functions. A great care is given to the checking of consistency: `'check3rbsb'` is called 124 times, `'check4tyle'` 66 times, `'erreur'` 388 times and `'rapport'` 22 times, all of them being checking functions. This slows down the computations but quicks up the programming work a lot. The non vital ones can be cancelled by a simple call to `'nocheck'`.

# 5 General points

## 5.1 how to start

Everybody has her/his own way to start using a new software. For `/rbsb/`, the suggestion would be

---

<sup>20</sup>and it soon occurred that it was also very convenient for the other too!

<sup>21</sup>But of course, they can be written with `/R/` as well!

<sup>22</sup>But we are lacking of a good trick to do it.

1. to source<sup>23</sup> the demo scripts (as `rebastaba.a.r`, see §5.7.2 to know their names), they exemplify the basic calculations and give some idea about the possibilities;
2. to read extensively this document, possibly skipping the more technical sections (for instance §6);
3. to run all scripts called by `rebastaba.tst.r` with the variables `rbsb.batch` and `rbsb.file` at `FALSE` to see the outputs onto the screen;
4. to try some own use starting with functions `rgn`, `rcatbn` and `rnorbn`, pseudo-randomly generating bayesian networks;
5. to try more personal `/bn/` with the program `rebastaba.b.r` proposing some standard process of a bn read from a text file generated by the user (of course examples are proposed where you can change the values at first).
6. finally to start working with `/rbsb/`, if you are convinced of its utility for you.

Never hesitate in giving us reactions, they could help us in improving the documentation.

## 5.2 Bayesian networks and their structure

Bayesian networks can be introduced in many different ways. Here we adopt the view (remind §3) of stochastic modelling, so nodes are considered as random variables.

*A `/bn/` is a parsimonious and efficient way to define a complete joint probability distribution on a set of random variables through marginal and conditional probability distributions associated to each node.*

When deterministic relationships are introduced, they are under the Dirac distribution<sup>24</sup>.

The basic components of a network are the nodes, and if the definition of a node includes the complete specification of the probability distribution through the parents, then a set of nodes can<sup>25</sup> defines a `/bn/`. This is the object structure adopted in `deal`. Here, we prefer to have a network structure<sup>26</sup> not resting on node objects<sup>27</sup>. The same reasoning applies for the graph structure<sup>28</sup>.

The first point is to define the set of nodes (including their possible variables) this is made with the object `/nom/` common to `/bn/`, `/gn/` and `/dn/` objects.

By network structure, we mean the set of arcs joining the nodes. There are, at minimum, four basic (and equivalent) ways to define the structure (that is the arcs between nodes) of a network:

1. Associate to each node the set of its parents (done this way for `/bn/`),
2. Associate to each node the set of its children,

---

<sup>23</sup>Of course, to do so, you need to put all `/rbsb/` files in your working directory, or make them accessible in some other way. For the moment, `/rbsb/` is not given the state of R-package.

<sup>24</sup>Of course `easyp` and `program` can do this also.

<sup>25</sup>If consistent.

<sup>26</sup>It is the classe `bn`.

<sup>27</sup>There is no more node object in `/rbsb/` even if the concept is very present.

<sup>28</sup>It is the class `gn`.

3. Define the set of arcs by the pairs : parent -> child (object /arc/ used for /gn/),
4. Define the parentship matrix (object /pam<sup>29</sup>).

According to what must be done, but also to the number of nodes and arcs, one of these ways can be more convenient than the others. Functions to go from one form to another are provided. ???ci-dessous il y a répétition sur la définition des domaines ???

### 5.3 domains for random variables

When exploring a data set associated to a /bn/, it is convenient to bear in mind magnitudes on the natural domain of each variable<sup>30</sup>. This is why to each node[variable] three domains are associated : **possible**, **representation** and **common** domains. The possible domain indicates which are the acceptable values for the variable, outside this domain means a NA value when simulating. The representation domain is used to determine the window to apply to the variable when drawing some graphical representation, outside this domain the observations are not pictured. The common domain states where the major values are expected (expert credibility interval) ; in diagram its limits are drawn with dashed lines giving a reference for the interpretation.

When the variable is numeric, a domain is defined by a couple of two numbers<sup>31</sup> : lowest and highest values; when the variable is categoric, a domain is defined by the set of possible values.

### 5.4 distinguishing nodes and variables

Remind first what was explained about the dimension of the nodes in §4.1.

As most operations concern the nodes, it is important to provide an easy and safe access to the nodes of a /bn/: this means the way of identifying them. Similarly to what is proposed for elements of /R/ vectors, two ways of identification are simultaneously available: by sequential identification (*iin*, for internal numbering) and names (a character string). The acronym *ion* is used to designate *iin* or *name*. One way or the other can be indifferently used in most cases, but the user is advised to prefer the name one because it does not vary when pruning a /bn/, or joining two /bn/s. Indeed, there is a context in which **only names** can be used, it is when links between nodes are defined<sup>32</sup>, more specifically when relating a node to other nodes of the /bn/ in *easyprograming* syntax. Parents are designated by their node *names* tagged with double curly braces. For instance "100\***{V4}**/**{six}**" will be interpreted as 100\*X/Y where X is the node with **name**="V4" and Y is the node with **name** ="**six**". Of course, this can be extended for variables and another example could be "**sqrt**(**{V5[a]}**)\*(**{V5[b]}**+**{V5[c]}**)<sup>2</sup>"

Of course, the nature (continuous, integer,...) and the domain (categories, segment,...) have to be applied to the variables not to the nodes. Meanwhile the links of a /bn/ are established at the level of the nodes, even if they can involve a subset of variables of the parent nodes, this explains what the parents are indicated by variables (see below). Further details are given in §6.3.

---

<sup>29</sup>See the definition in §9

<sup>30</sup>This could be generalized at the bivariate/multivariate level.

<sup>31</sup>It is a range.

<sup>32</sup>The reason is that the *iin* can be modified for a manipulation of the /bn/s, the *name* is the stable identification.

Table 1: `rbsb.nat_pro` contents

	categoric	ordered	numeric
<code>conti</code>	FALSE	TRUE	TRUE
<code>integ</code>	FALSE	TRUE	TRUE
<code>cateo</code>	TRUE	TRUE	FALSE
<code>categ</code>	TRUE	FALSE	FALSE
<code>unkno</code>	FALSE	FALSE	FALSE

## 5.5 assembling nodes to form a `/bn/`

The basic way to obtain a `/bn/` is to start a zero node `/bn/` and adding successively as many nodes as wanted<sup>33</sup>. For instance:

```
B1 <- zero2bn("mon réseau");
B1 <- and4bn(B1,rbsb.alk_examples[[1]]);
B1 <- and4bn(B1,new8alk("B",ltype="normal",
                      lpara=list(mu="{nor1}",sigma=1),
                      lpod=list(c(-10,10)))
           );
print(B1);
```

defines a `/bn/` comprising two nodes named "nor1" and "B", "nor1" being the parent of "B". "B" is normally distributed and "nor1" a prepared node also follows a normal distribution with expectation the value of node "nor1" and standard variation one.

Much more elaborated cases are proposed in the example and demonstration files<sup>34</sup>.

For the sake of the shortness of the terminology, **link** stands for conditional probability distribution (including deterministic links). It must be understood as the way a node is connected to the remaining part of the `/bn/`. Links are defined when creating a new node, they can be modified afterwards. In `new8alk` function the link to create is specified by different arguments according to the type of link (see Table 2). Precise and uptodate information can be found using `help8ltype()` function. Some clues are given in §6.1.

The indication of the parent nodes for a new node can be extracted from the definition of the parameters for standard links or given by the user hands for more flexible links. Remind that `node[variable]` names can be used to identify the parents. The reader is referred to the examples and to the documentation of the functions for a more detailed comprehension.

## 5.6 playing with a `/bn/`

Once a `/bn/` has been defined it can serve different purposes:

- to print the `/bn/` specification (`print.bn`).
- to generate the associated DAG (`bn2gn`) and to give it a convenient disposition (`modify4gn`).

<sup>33</sup>Pseudo-random automatic generation is also possible.

<sup>34</sup>files `rebastaba.e??r`

Table 2: `help8ltype('argu')` specifies which arguments are associated to the different types of link.

```

=====
Arguments/Properties according to the different 'ltype's
=====
<YES> for 'must be provided by the user'
<yes> for 'can be provided by the user but there is a default value'
< no> for 'necessary but generated by rebastaba'
< NO> for 'necessary but fixed by rebastaba'
< - > for 'irrelevant'

```

	family	rep?	bugs?	lpara	ldim	lnat	lvar	lparent	lpod	lred	lcod	ltransfo	leasyp	lval	ldif	lfunc
normal	c_s	TRUE	TRUE	YES	yes	NO	no	no	YES	yes	yes	yes	-	-	-	-
uniform	c_s	TRUE	TRUE	YES	yes	NO	NO	no	YES	yes	yes	yes	-	-	-	-
Bernoulli	d_s	TRUE	TRUE	YES	yes	NO	NO	no	NO	NO	NO	NO	-	-	-	-
binomial	d_s	TRUE	TRUE	YES	yes	NO	NO	no	YES	yes	yes	-	-	-	-	-
Dyrac	mis	TRUE	TRUE	YES	yes	yes	NO	no	YES	yes	yes	yes	-	-	-	-
multinomial	d_v	FALSE	FALSE	YES	no	no	yes	no	YES	yes	yes	-	-	-	-	-
numcat	cat	FALSE	FALSE	YES	-	NO	yes	yes	YES	yes	yes	-	-	-	-	-
easyp	mis	FALSE	FALSE	-	yes	YES	yes	no	YES	yes	yes	yes	YES	-	-	-
empidata	dab	FALSE	FALSE	-	no	YES	YES	yes	YES	yes	yes	-	-	YES	yes	-
popula	dab	FALSE	FALSE	-	no	YES	YES	yes	YES	yes	yes	-	-	YES	yes	-
program	mis	FALSE	FALSE	-	no	YES	YES	yes	YES	yes	yes	-	-	-	-	YES

```

( ldes ltype ) are always YES
( lcomp ) is always no
Probability families are:
c_s : conti_scalar
d_s : discr_scalar
c_v : conti_vector
d_v : discr_vector
cat : categoric
dab : data_based
mis : miscellaneous

```

Table 3: `help8ltype('normal')` specifies the meaning of the arguments for the type of link `normal` and the default values.

```

=====
ltype = normal
=====
Classical univariate normal distribution.
(*)This distribution can be repeated(*)
(*)This distribution can be translated into Bugs(*)
(*)(lpara):   there are 2 parameters:(*)
               (*)mu:expectation(*)
               (*)sigma:standard deviation(*)
(*)(lnat):    nature(s) of the variable(s) node(*)
               (*)default: conti(*)
(*)(lvar):    names for the variables of the node(*)
               (*)default: [''](*)
(*)(lparent): names of the node parents(*)
               (*)default: no parents(*)
(*)(lpod):    possible domain(s) of the variable(s) node(*)
               (*)default: (*)
(*)(lred):    representation domain(s) of the variable(s) node(*)
               (*)default: lpod value(*)
(*)(lcod):    common domain(s) of the variable(s) node(*)
               (*)default: lred value(*)
(*)(ltransfo)$ some transformation can be asked for the generated values(*)
               (*)default: [[3]](*)

```

- to generate a data set from the whole construction (`bn2dn`).
- to compute other marginal, conditional and joint probability distributions for some nodes in case of a grappa compatible `/bn/` (`grappa4basic`).
- to compute empirical posterior distributions for available data sets (`bn2bugs` and relatives).

It is also of interest to study the properties of conditional independence. A way to do so is to construct sub `/bn/s` focusing on a subset of nodes conditioned by the values (or distributions) of another subset of nodes. It already exists some functions but these ideas have to be experimented and refined.

## 5.7 organization

### 5.7.1 main objects

The objects and their structures are all defined in `rebastaba.1a.r` file. The object list tends to increase but it is still reasonable. `/bn/`, `/gn/` and `/dn/` are the main objects and are described below. `/arc/` and `/pam/` objects were already mentioned before (see Table 4 and Figure 2).

**bn** stands for Bayesian network and is devoted to the description of the joint probability distribution of a set of random variables.

**gn** stands for graph on a set of nodes. It is specialized around the properties of the graph structure and its representation. Notice that cycles are allowed.

**dn** stands for data set, most often simulated from a `/bn/`.

At the beginning of `/rbsb/`, there were some redundancy between these objects. After some experimentation, it was decided to specialize them as reported above. The consequence is a more strict universe of objects and a more expansive management of them... but in hope of a safer tool at the end. Probably the best way to know which constraints are imposed upon the objects is to look at the code of `check3rbsb` function, systematically called to check the validity of transmitted arguments. When an inconsistency is discovered, either by `check3rbsb` or other function, a message is issued giving insight about the error. It is always preceded by the signature of `/rbsb/` and the printing of faulting objects; no indication is given about the location of the error, the user must call `traceback()`, basic function of `/R/`, to know where it occurred.

### 5.7.2 distribution files

The complete set of `/rbsb/` is given by a set of files all starting with the prefix `rebastaba`<sup>35</sup>. Usual suffixes are used to specify the type of file: `r` for `/R/` code, `dat` for data input, `txt` for text output and `pdf` for graphics outputs, and some others. The number of characters for the component between the prefix and the suffix indicates the role of the file: 0 for compiling the functions, 1 for specific programs, 2 constants and functions, 3 tests of the functions, 4 inputs for some demos, 5 for miscellenaous, 6 for documentation and 7 for demo outputs. Among them are:

---

<sup>35</sup>In fact `version` is empty for the version under development.

Figure 2: The different /rbsb/ objects and their relationships. **bn** for bayesian network; **gn** for graph of a network; **dn** for data set simulated from a /bn/; **arc** for arcs; **pam** for parentship matrix; **gen** for genealogy; **alk** for asked link; **pos** for positions; **pagr** for parameters of the graph; **pt** for probability table; **deal**, **grappa** and **bugs** represent external applications. Dashed line indicates that the relationship is to be made. Heads of the arrows indicate when /rbsb/ functions exist to go from an object to another object.

### main ReBaStaBa objects

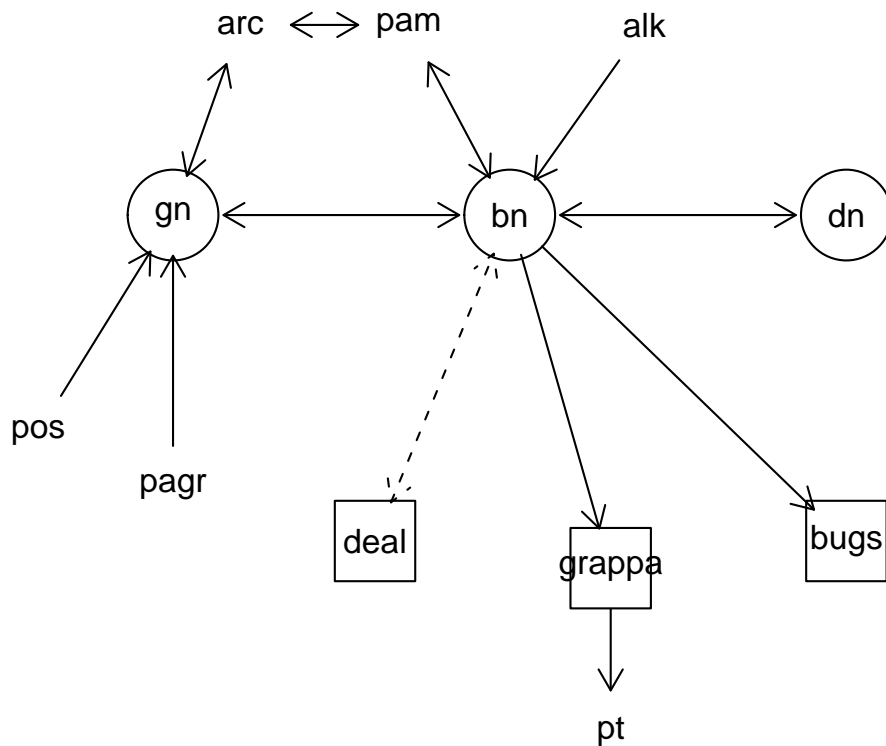


Table 4: Comparative main properties of **bn**, **gn** and **dn** objects.

concept	bn	gn	dn
elements	node & variables	nodes	variables
structure	parents	arcs	-
variable properties	yes	no	yes
node properties	yes	yes	no
link properties	yes	no	-
positioning	no	yes	no
statistical properties	no	no	yes

1. `rebastaba.r`, the `/R/` script launching `/rbsb/` objects in the working directory. It needs the variable `chemin` giving the directory where can be found the `/rbsb/` files.
2. `rebastaba.?.r`, the `/R/` scripts defining some demos. Especially intended for the user.
3. `rebastaba.la.r`, the `/R/` script defining the classes and constants. It also comprises the function `affaire` which gathers and relates minor improvements to be done in spare time.
4. `rebastaba.f?.r`, the `/R/` scripts defining the functions of different levels (`f3`: standard functions, `f2`: advanced functions, `f1`: deep function and `f0`: useful but not specific `/rbsb/` functions). The idea is that `{f0,f1}`, `{f0,f1,f2}` and `{f0,f1,f2,f3}` are consistent sets of functions. All functions are fortified with extended comments either after the function name in a standardized manner to be transformed in Rd files or in a free style within the code.
5. `rebastaba.et?.r`, the `/R/` scripts performing specific tests of the basic functions. These scripts are not specially intended for the end user.
6. `rebastaba.ed?.r`, the `/R/` scripts performing small (and commented) demonstrations (`d1`: the toy example (cf. §7), `d2`: my first `/bn/` with `/rbsb/`, `d3`: going from a `/bn/` to a `/gn/` and vice-versa,...). These scripts are specially intended for the end user.
7. `rebastaba.ee?.r`, the `/R/` scripts performing small (and commented) examples (`e1`: very simple multinormal `/bn/`, `e2`: very simple multinomial `/bn/`, `e3`: the Asian `/bn/`, `e4`: using grappa to compute conditional and marginal probability tables). These scripts are specially intended for the end user.
8. `rebastaba.ema.r`, the `/R/` script performing figures and outputs for this manual.
9. `rebastaba.tst.r`, which runs all `rebastaba.e??r`, files. Useful to know that the constant `rbsb.batch` specifies if the execution must be interactive (`FALSE`) or batch (`TRUE`); the constant `rbsb.file` specifies if outputs<sup>36</sup> must be stored in files (`TRUE`) or displayed onto the screen (`FALSE`).
10. `rebastaba.exy?.dat`, file text of data values for the empirical nodes used within script `rebastaba.exy.r`.
11. `rebastaba.conte.r`, the `/R/` script revising the creation and revision dates of `/rbsb/` functions. It produces `rebastaba.conte.pdf` file.
12. `rebastaba.manual.*`, files related to this document.
13. `rebastaba.Rdocu?.pdf`, the standard documentation of `/R/` functions for the three levels. `Rdocu1` contains all functions, `Rdocu2` contains only `{f0,f2,f3}` functions and `Rdocu3` contains only `{f0,f3}` functions, so any type of users have only one of these documents to consult.

---

<sup>36</sup>texts and graphics

### 5.7.3 implicit rules

As far as possible, consistency is intended when creating functions to lighten the need to examine the documentation. Here are some pseudo-rules:

- When a function makes some action with an object, the name will be `action8object`. Examples are `plot8gene`, `paths8gn`, `help8ltype`,... The first argument of such functions must always be the concerned object. When the function is overwriting some standard function, the 8 is replaced by a dot as usual: `print.bn`, `plot.gn`,...
- When a function creates a new object (or structure) `b` from another object (or structure) `a`, its name is `objecta2objectb`. Examples are `bn2gn`, `gn2bn`, `df2bn`,... The first argument is the object serving for the creation, *i.e.* `objecta`.
- When a function transforms or modifies an already existing object `b` from another object `a`, then its name is `objecta4objectb`. An example is `???`. The first argument is `objecta`, the second argument is `objectb`. Also when a function complements an object with some additional property, its name is `property4object`, and the first argument is the object. Examples are `position4gn`, `and4bn`,...
- Specialized functions start with a four/five character prefix. The prefixes are `geom` for geometry, `expr` for expressions, `form` for format, `help` for help. In that case the numeric separator is 3 as for `check3rbsb` and `form3title`.
- arguments, slots, keywords are usually singular and in lower cases.
- For `/rbsb/` functions manipulating nodes, the general rule is that inputs (arguments of the function) are given as *ion* (indifferently internal numbering or names) and output are obtained as *iin*<sup>37</sup>.

## 6 More details

### 6.1 defining a node

The definition of a new node is done through a collection of arguments the name of which starts with '1'. Not all are of use according to the node type. Rules are provided in the function `help8ltype()`. Before describing the main significances of some of them, it is important to know that the definition of a node can be in three ordered different states :

1. Such as defined by the user, for instance when writing a text file to define a `bn`. This correspond to the inputs of the function `new8alk`. For instance, `numerics` are given as `characters`.
2. Such as interpreted by `/rbsb/`, more or less completing some of the defaults but not related to a precise `/bn/`, the node defined *per se*. This corresponds to the output of `new8alk` and to the inputs of `complete8alk`. They are called uncompleted `/alk/s` and checked as such by `check3rbsb`. This levels of specification is sufficient for using nodes for `/gn/s`.

---

<sup>37</sup>The key function is `nv2ion` for such transformation, unfortunately not that simple so specialized versions exist like `nanv` and `nbnv`.

Table 5: Parameters associated to each available distribution. More details are given by `help81type(«distribution»)`.

distribution	paramètres
normal	mu sigma
Bernoulli	p
binomial	n p
multinomial	n p
Dyrac	k
categorical	p
easyp	-
program	-
empidata	-

- Such as linked to a given `/bn/` where the parentship can be checked and also where the simulating functions are elaborated. This corresponds to the output of `complete8alk`. They are called completed `/alk/s` and checked as such by `check3rbsb`. One of the important part played at this step is the replacement of multidimensional nodes by their collection of variables.

Here are the main fields described from the user viewpoint that is the simplest form (1), after creating the `alk` object (2) and after inclusion in a `/bn/` (3).

**ldes** (`/ds/` object) (1) Description of the `/alk/`. At least the name slot must be provided. Other characteristics of the node can be provided concerning the date of creation, the origin, the extended definition, the role as well as free comments...

**ltype** (scalar character) (1) Indicates the type of links : mandatory, of course this type must be predefined (type `help81type('argu')` to get the list). For instance "normal" indicates that the distribution of the node is normal. The available types of links are given by the `help81type()` function (See Table 5). The type of links are standard probability distributions plus three additional possibilities : "easyp" for easy programming, "program" for `/R/` standard programming and "empidata". What is expected in the other arguments depends on the `ltype` value (see Tables 2 and 3). For the moment, a reduced series of standard probability distributions is more or less implemented, it will be increased soon.

**lpara** (list of numerics and/or characters) (There are no more default values, they are replaced by elements in the liste `example.alk` that the user can easily modify.) When `ltype` is a standard distribution, `lpara` must provide a list with the corresponding parameters. The parameters can be simple functions of already defined variables of different nodes<sup>38</sup>. For instance `lpara=list(mu="{A}",sigma=0.49)` indicates that a normal distribution has got the value of node A as expectation and a standard deviation of 0.49; this implies also that A node is the unique<sup>39</sup> parent of this node. See Table 5 for the list of parameters.

<sup>38</sup>Even if theoretically possible with networks , parentship between variables belonging to the same node are not possible in `/rbsb/`. Something of this kind ought to be implemented for dynamical `/bn/`.

<sup>39</sup>except if other were introduced with `ltransfo` argument.

**lrep** (scalar numeric) (can be imposed for repetitions (1) or deduced<sup>40</sup> from **lvar** at (2)). It is zero for non-repeated nodes.

**lnat** (character) ((1) scalar may be vector for multidimensional node with different natured variables; extended at (2) when the node is repeated) This argument provides the nature(s) of the node variable(s); the natures must belong to the registered list **rbsb.nature** which comprises the unknown case (see §5.4); their properties are indicated in **rbsb.nat\_prot** (see Table 1).

**lvar** (character) ((1): only for multidimensional nodes; provided at (2) for repeated nodes) This argument provides the names of the variables when the node is multivariate. These names must not comprise the node name and the brackets. When empty string, the node is univariate and the variable name is ". For standard mutivariate distributions (like multinomial) the names are automatically generated and are some numbering; they must be provided by the user for **program** and **empidata ltypes**.

**lparent** (character) (at (1) for some distributions but deduced at (2) for standard distributions) This argument provides the name of the parents for the node when those cannot be deduced from the other parameters. This is the case when **ltype** is "numcat", "empidata" or "program". This list of parents is provided under the complete responsibility of the user<sup>41</sup>; when **ltype** is **empidata**, these parents will be used for the restriction in the random drawing of the node variable(s) value at simulation time. Remind that parents are designated by the complete node[variable] names when not all the variables are necessary<sup>42</sup>.

**lpod** (list of numeric (for numeric variables) and character (for categoric variables)) ((1) except for the Bernoulli nodes where it is imposed to (0,1)) This argument provides the strict domain(s) of the node variable(s). Practically, it is the lower and upper bounds for numeric variables, or the set of possible values for categorical variables. When a value is generated outside this domain, it is replaced by a missing value (NA). Must be a list<sup>43</sup> to give possibly different domains to each variable of the node.

**lred** (can be fixed at (1) by the user if not copied from **lpod**; nevertheless cannot accept infinite values contrary to it) Similar to **lpod** but used as window for graphical representations (to avoid the outliers to mask the variation of the majority of the values).

**lcod** (can be fixed at (1) by the user if not copied from **lred**) Similar to **lred** but used to draw limits on graphical representations (to give some natural scale for the variable).

**ltransfo** (scalar character) (if exist must be defined at (1), standard rounding is implemented for standard continuous links) For most standard distributions and "easyp", **ltransfo** can be NULL or provides a character string to be parsed as /R/ instructions to perform some transformations with a shortcut for rounding. When the node is multivariate, it applies individually to all the variables of the node. For the other cases a new node needs to be created.

---

<sup>40</sup>This has to be checked.

<sup>41</sup>To the contrary, for standard distributions and **easyp**, the parents of the node are deduced by /rbsb/ from the provided definitions.

<sup>42</sup>with the exception of multivariate repeated nodes.

<sup>43</sup>even is the node is univariate

**ldf** (`data.frame`) This argument specific to data based nodes provides either the name of the file where is stored the data base<sup>44</sup> or the `/R/` (variable or function) name under which the `data.frame` is accessible. See the code of `read8df` for complete details. **ldf** is a list comprising at least `name`, `what` and `valu` components.

**lwin** (scalar numeric) This argument specific to data based nodes specifies how to condition the empirical drawing from the parent values with windows associates to each variable (see the function `draw8empidata` for complete details, also §6.4).

**lfunct** (function) This argument provides pieces of codes to define the link when **ltype** is "program", it provides an `/R/` function which will be introduced without control and must be consistent with the list of parents provided in `lparent`. See also §6.7.

**lcomp** (logical) Indicates if the link is already completed to pertain to a `/bn/`; this field is managed by `/rbsb/`.

## 6.2 repeated nodes

For standard distributions used in a repeated way, the parameter(s) must be given for each in one of the (consistent if more than one) following way:

- a single numerical value for all parameters and `lrep` or other parameter(s) indicating the dimension,
- a single `easyp` expression comprising one (or several consistent) repeated parents,
- a vector of numerical values for parameters (giving then the number of repetitions),
- a vector of `easyp` expressions comprising one or several repeated parents.

Take care also that if «RN» is a repeated node of dimension 3, you can use «c(1,2,5)» or «RN» or «c(«RN[1]», «2\*RN[2]», «5\*RN[3]»)» but **NOT** «c(1,2,5)\*RN»: that is cycling is not implemented.

## 6.3 multivariate nodes

The following conventions are adopted:

1. The name of a node is a single character string, for instance `A`, `Age` or `Size.arm`.
2. The name of a variable is the name of its node followed by its name within square brackets, for instance `A[today]`, `A[tomorrow]`.
3. A node can also be designated by its sequential number within the structure (`/nom/`).
4. Naming the variables is not compulsory, in that case they are attributed standard variable names.
5. When a variable name set is expected, giving a node name is equivalent to give the set of all variables belonging to this node.

The basic reference remains the function `nv2ion`.

---

<sup>44</sup>It will be read with `read.csv2(df@valu)`; This means that for `categ` variables, no space must exist between the semi-colon separators. See the file `rebastaba.dae2.dat` for an example.

## 6.4 data based distributions

`/Rbsb/` allows also to use so-called *empirical distribution* that is sampling among the values stored in some database. Presently, the type names chosen for this kind of distribution is *empidata* (random drawing) and *popula* (systematic drawing). Simulating with it is done through `draw8empidata` and `draw8popula` functions. For systematic drawing, the node is supposed to be without parents which is possible for random drawing. Here are some indications about random drawing, complete details can be found in the Rd documentation. The first key idea to get is that in case of drawing values for a node comprising parents, besides the data values from which the values are extracted (**E**), the drawn parent values (**X**) are compulsory to establish the relationship. The second key idea is that for some combinations of the parents no equivalent can be found<sup>45</sup> in **E**, in that case either a **NA** is drawn or the subset of possible values is enlarged: parameters `dif` and `strict` monitor this situation.

More precisely the arguments of `draw8empidata` are : `ny`, **E**, **X**, `nam`, `nat`, `dif`, `bnid`, `strict=FALSE`, `mnbo=10`. Briefly `ny` is the name of the node to simulate, **E** is the data.frame defining the empirical distribution, **X** is the set of already simulated nodes comprising the parents of `ny`, `nam` is the vector of the parent names, `nat` their natures, `dif`, `strict` & `mnbo` the way to select the observations in **E** from which the drawing will be done (subsetting), `mnbo` the minimum number of selected observations for a drawing to be considered (if not reached a **NA** is returned). When there are no parents, then a simple drawing is done in the set of rows of **E**. When there are parents, the same but a subsetting of **E** is firstly performed for the observations having the same values of parents as can be found in **X**<sup>46</sup>. One can imagine that the way of subsetting depends on the nature of the parent variables.

A specific care must be given to the names of variables in data coming from `data.frames`. When extracting variable(s) using such a way, the resulting node is always considered as multivariate (even if with only one variable), and this rule must be enforced: the variable names of the `data.frame` must not comprise '[' or ']' since they will be used as variable names of the node. Nevertheless the possible parent variable are taken without considering the node name. This obliges when the parent variable comes from another multivariate node to use some intermediate node just to change the variable name<sup>47</sup>. To give an example, if the `data.frame` used to define the node `STATURE` comprises the variables `SEX`, `HGT` and `WGT` and that the parent is `SEX`, then it must preexist a univariate node `SEX` and the result will be a bivariate node of components `STATURE[HGT]` and `STATURE[WGT]`.

Two other distributions can be assimilated to empirical distributions in `/rbsb/`: `multinomial` which refers to a multivariate node and `numcat` which refers to the standard discrete situation in most softwares dealing with `/bn/`. Two specialized functions were created for them, respectively `draw8multinom` and `draw8edcg`. The user needs not to know them but they can give ideas about the way the simulation is performed in `/rbsb/`<sup>48</sup>.

---

<sup>45</sup>Especially for continous parents.

<sup>46</sup>This requires that the parent be also in **E**.

<sup>47</sup>A way for better flexibility would be to introduce formal equalities to define the parents, e.g. `PARENT[SEX]=CHILD[VAR]` but this is a lot of work for few cases... Don't forget that always exist the possibility to use direct `/R/` programing where everything can be done.

<sup>48</sup>The master function being `simulate8bn`.

## 6.5 Parenthood

There are some rules to satisfy when defining the parent of a new node due to the complexity borne by the multidimensional nodes.

- Repeatedness of a repeatable node can be generated (or reinforced<sup>49</sup>) only for standard scalar distributions having scalar parameters. To give some examples:
  1. `lpara <- list(mu='{{A}}',sigma=1)` is right when the node A is repeated
  2. `lpara <- list(mu=c('{{A}}','{{B}}'),sigma=1)` is right only when the nodes A and B are not repeated.
  3. `lpara <- list(mu=c('{{A[-2-]}}','{{B[prems]}}'),sigma=1)` is right since only one variable of the multidimensional nodes A and B are considered.
- Parenthood is only given through **lpara** and **leasyp**, no more with **ltransfo**. If necessary intermediate nodes have to be added or a program link to be implemented.
- Within `/rbsb/` inheritance of dimensionality due to parenthood is established in two steps: for uncompleted links scalar and repeated nodes are not distinguished since the characteristics of parent nodes are not known; the complete analysis is done when completing the node within a `/bn/`. Nevertheless if repeatedness is directly provided (by repeated parameters or a `ldim` greater than one for **scalar** nodes), it is taken into account.

## 6.6 introducing pieces of code into a `/bn/`

`/Rbsb/` has to adapt its algorithms to the specification of the `/bn/`. At a given level to obtain complete generality, there no more possibility that to ask the user to give some elements of code. This can be done at three different levels:

### 6.6.1 indicating variable names

The most straightforward example is when specifying a parameter distribution by the parent name. For instance `lpara=list(mu='{{A}}',sigma=0.49)` states the value of parameter `mu`, for this distribution, must be taken as the value of the node `A`<sup>50</sup>, indicating at the same time that `A` is a parent of this node. Another example is when defining the parent(s) for a link with an empirical distribution. `lparent=c("dad","mom")` implies that the `draw8empidata` function will use these two parents to define the subset of values in which to draw values for this node.

### 6.6.2 writing simple expressions

The definition of the parameters from the parents can be extended by some simple `/R/` expressions. For instance `lpara=list(mu='sqrt({{A}}+{{B}})',sigma='abs({{A}})')` states the `mu` parameter must be given as the square root of the sum of the two nodes `A` and `B`, indicating at the same time that those nodes are the parents of the node being defined; also its standard deviation is the square out of node `A`... The reader noticed the use of the double curly bracketing<sup>51</sup> to tag the variable names.

---

<sup>49</sup>That is already present for one parameter and generated on another parameter.

<sup>50</sup>Notice the tagging by double curly braces.

<sup>51</sup>Bracketing strings are given in `rbsb.pth` matrix.

### 6.6.3 mind out

Easyprogramming works well in standard cases but be in difficulty in extreme situations. For instance `mu=1:3` is considered as the indication of three repetitions and `mu=«1:3»` not<sup>52</sup>! Also `k=«{A}»-«{A}»+10»` will retain node A as a parent, which is not! Always be conscious of such artefacts when you get a non understandable error.

## 6.7 providing consistent simulating functions

In case of `ltype == 'program'`, the user can introduce any /R/ function s/he wants. This means the writing of an /R/ function to be used as such within /rbsb/ code. Then some deeper understanding of /rbsb/ is needed at least at the level of inputs and output of the function! First let us give two illustrations:

```
without_parent <- function(X) {
  nbsimu <- nrow(X);
  res <- runif(nbsimu,10,20);
  res[res<15] <- res[res<15] - 10;
  res;
}
#
with_two_parents <- function(X) {
  # X is the already simulated matrix
  # It comprises the two parents «A» and «B»
  mu <- log(X[,«A»]); sd <- abs(X[,«B»]);
  round(rnorm(nrow(X),mu,sd),2);
}
```

The argument of this function is a matrix, possibly with 0 columns but with the number of rows equal to the number of simulations to perform (`nbsimu`). The existing columns correspond to the already simulated variables, they can be used with standard /R/ syntax to simulate the value for the considered node. In case of categoric node, it can be profitable to use the `draw8edcg` function contained in `rebastaba.f1.r` file, it solves the unpleasant problem of mixing into a simple index several categoric parent variables.

If the node is univariate the output can be a vector of length `nbsimu` or a matrix (`nbsimu,1`). When the node is multivariate, it must be a (`nbsimu,nv`) matrix where `nv` is the number of variables.

## 6.8 how /rbsb/ incorporates a new node into a /bn/

(In fact the new node can be the first node into a new created /bn/ with `zero2bn`.) The incorporation of a new node is accomplished by `and4bn`, the main steps are:

1. The user defines an /alk/ which is candidate to be integrated. This must comprise different fields according to it type, some are compulsory, other ones are optional. In fact

---

<sup>52</sup>In spite of the fact that `mu=«3»` works. This is due to the use of `as.numeric`.

the field are classified into four categories: **YES** (must be provided by the user), **yes** (can be provided by the user but default values can be provided), **no** (cannot be provided by the user but they are necessary and deduced by /rbsb/ itself, **NO** (necessary and fixed by /rbsb/) and '-' (non relevant for this type of node). The call to `help81type(«argu»)` displays all fields and their respective categories for all type of nodes.

Such a definition can be done with `new(«alk», ...)` or more commonly by reading a list from a file and converting it with `list2alk` function.

2. The node candidate is scrutinized by /rbsb/, checked and completed according to the already present nodes. This concerned mainly the parent nodes and the dimension of the node. The /alk/ object is said to be completed; `complete8alk` performs this completion. This is one of the most intricated actions of /rbsb/ package since a differential syntax analysis has to be done of the contents of the fields proposed by the user.

Here are the main steps of this function. The first one consists in finding the parents of the node (with `parent8alk`), this is done, irrespectively of the type of the node, by exploring the different slots conveying this information (`lparent`, `lpara`, `ltransfo`). Then the number of repetition is obtained (with `nbrep4alk`); it can be zero (for non repeatable nodes) or a positive number for repeatable nodes. Then the variable names are provided (`var4alk`); the user can always impose his/her wishes with `lvar` slot, if not standard names are provided (remind that when the node is univariate, there is no need for a variable name but it can be given; also that `length(lvar)` is used as the dimension [=number of variables] of the node).

3. The /bn/ object is completed with the new node. That is all necessary information for the /bn/ is stored in the perspective of this object. Notice that `bn2alk` allows back transformation.

## 6.9 Links with other packages

### 6.9.1 Bugs translation

The present Bugs translation is intended for the Jags [6] dialect. The translation comprises the writing of the model associated to the /bn/ in a file and an associated file containing the corresponding Jags script. For the moment, only the univariate normal with repetition were implemented (see `*.etp.r` for examples). But generic functions were prepared for most distributions available in Bugs. Presently, since the last modification of the /rbsb/ object structures **nothing is supposed to work**.

### 6.9.2 Grappa access

In case of /bn/ comprising only `numcat` variables, it is possible from /rbsb/ to perform the algorithms developped in Grappa [3] and to get marginal and conditional probability tables which are not included in the definition of the /bn/. This suposes the availability grappa, easily obtained from the internet pages of Peter Green (Department of Mathematics, University of Bristol, UK).

### 6.9.3 deal / bnlearn

To be considered.

## 6.10 /rbsb/ constants

For readability and ease of modification, some /rbsb/ constants are defined in file `*.la.r`, they are named `rbsb.*`, for instance:

- Parenthesis to define numerical constants and rounding transformation. To obtain them, display the `rbsb.pth` matrix.
- `rbsb.fatal` indicates if a stop has to be issued when an error is found
- `rbsb.nature` lists the different natures of random variables, `rbsb.nat_pro` their associated properties.
- `rbsb.parag`, where graphical parameters are stored (of course the user can modify them).
- `rbsb.graf`, the type of graphical files to produce.
- `rbsb.fig`, the numbering of graphical files.
- `rbsb.na_ra`, the ratio of NA value from which a warning is issued.
- `rbsb.nb_min`, the minimum number of values to provide statistical summary of /dn/.
- `rbsb.nd_itself`, the character string to code the node itself when defining a transformation.
- `rbsb.alk_examples`, a series of /alk/ examples.
- and also a series of null values associated to the main /rbsb/ objects, *e.g.* `nom.null`, `df.null`,... To have an overview, just look at the function `isvide` in charge of identifying them.

## 6.11 miscellaneous

- Notice that /bn/, /gn/ and /dn/ objects have some common slot types (for instance to define the series of node names: `@nom`). This allows to use generic functions on them.
- In /rbsb/, the objects `pam` and `arc` don't include the names of the nodes (see §5.4).

## 6.12 statistics

The /rbsb/ set has got the characteristics presented in Table 6.

## 7 Toy example

Here is proposed a reduced application showing some of the basic possibilities offered by /rbsb/ standard functions. In fact, it is a part of `rebastaba.a.r` program.

Table 6: Some approximate features about /rbsb/ programing

number of code files		31
number of code lines (including comment lines)		20277
number of comment lines		9098
number of functions		226
number of S4 objects		14
number of calls to <code>check3rbsb</code>		125
number of calls to <code>check4tyle</code>		66
number of calls to <code>erreur</code>		389
number of calls to <code>rapport</code>		22

```

# starting an empty /bn/
bb <- zero2bn("Toy Example");
# defining three nodes
#   of names 'A', 'B', and 'C'
na <- new8noralk("A");
nb <- new8noralk("B","A");
nc <- new8noralk("C","A");
# incorporating the nodes to the bn
bb <- and4bn(bb,na);
bb <- and4bn(bb,nb);
bb <- and4bn(bb,nc);
# printing the /bn/
print(bb);
# plotting the /bn/
plot(bb);
# simulating with the bn
print(bn2dn(bb));

```

It is worth noticing the way the relationships between the three nodes are established through the definition of the distribution parameters.

## 8 Developments

Here are given some indications about the development already<sup>53</sup> performed and planned. Three levels are distinguished for each topic but the border between them is not firmly established! Other pending minor improvements are listed by the `afaire` function included in `rebastaba.la.r` file. Also ??? indicates within the code itself points to think about or to finish. It can occur that some function are started but not finished, they are left to introduce the idea and to provoke possible reactions.

---

<sup>53</sup>obvious points are removed, only those interested to know are left.

## 8.1 documentation

### 8.1.1 fixed

- `help81type()` displays which types of link (including standard distributions) are available and the needed arguments with their possible default values.
- Rd documentation of existing functions (by means of standardized comments introduced within the function code and extrated by the help of a Perl script).
- This document (even if it will be fitted to the future versions).

### 8.1.2 started

- Standardize the keywords of the functions in the code (used to constitute the index of Rd manuals).

### 8.1.3 planned

- Include this document to the Rd documentation.
- Fill the `#CALLING` fields and exploit them, making the graph of the calls.
- Propose a complete `help.rebastaba` function if possible with html pages connected to the function documentations.

## 8.2 construction and manipulation of objects

### 8.2.1 fixed

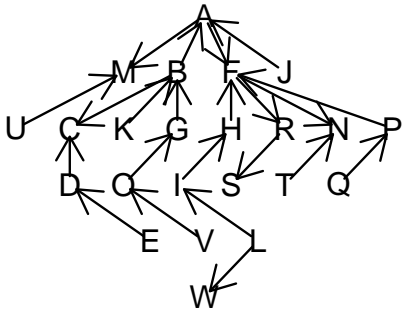
- Generating pseudo-random `/gn/` and `/bn/` (see Figure 3).
- Checking objects
- Checking domains and using them.
- General loop for Bugs dialect : repeated nodes (but to be fixed for the new objects)
- Introduce a description field allowing the user to integrate in the `/bn/` definition all relevant reference and comments. This is done through objects of class `«ds»`, as an extension of the previous `name` field; six components are proposed: `name`, `origine`, `time`, `definition`, `role` and `comment`.
- Extracting sub-`/bn/` from different viewpoints.

### 8.2.2 started

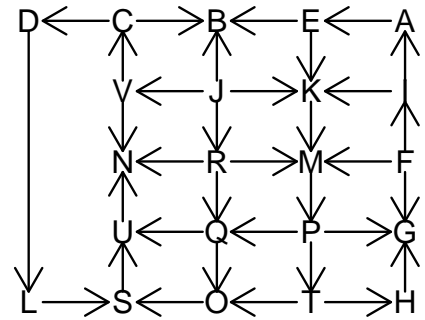
- discretize continuous variables

Figure 3: randomly generated gn objects. (**h**ierarchy, **s**quare, **r**ound, **t**otal).

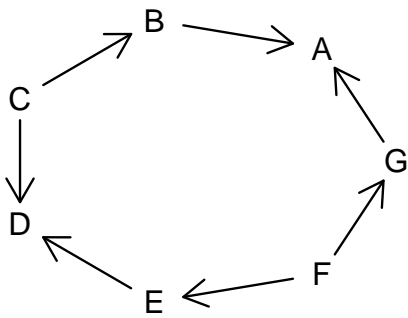
**(h)**



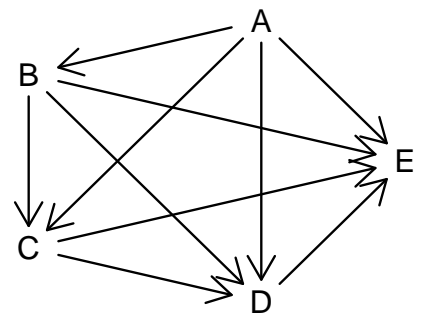
**(s)**



**(r)**



**(t)**



### 8.2.3 planned

- add most of probability distributions available in R.
- translate other probability distributions available in Bugs.
- provide standard /bn/s as the Asia bn as the result of a function, those resulting for deal examples,...
- introduce dynamical /bn/s as an orthogonal extension of the multivariate nodes
- function to suppress from the /bn/ those *one parent and one child* nodes since they do not participate to the branching structure of the network.
- generate the deal family models (distribution defined by parents and nature of the node). This is different of interfacing with deal.
- more generally think about classifying the bn according to their properties.
- add denomination to the arcs

## 8.3 displaying objects

### 8.3.1 fixed

- printing of most objects.
- plotting of most objects.

### 8.3.2 started

- Create derived function from `modify.gn` function, for instance a `gn` object could be generated *de novo*.
- Decorating diagrams with the properties of nodes and arcs.

### 8.3.3 planned

- provide a positioning keeping the arc directions (all ancestors at the top, all final nodes at the bottom and all arc from up to down). Or positions the point with respect to a subset of nodes (based on some kind of pseudo-distances). Breath First search. Depth First search.
- Think about introducing bent arcs in graphical representations of networks. This could be done by a chain of points, defining a broken line the angle of which could be smoothed in a way or another. Unfortunately, this important point for the clarity of the representation is not that easy.
- Allow more decorations of the nodes (color, geometrical surrounding), taking care of the dimensions of the node names...
- Allow more different styles of drawing for arcs

## 8.4 exploiting objects

### 8.4.1 fixed

- Simulating data sets from a /bn/
- Detecting cycles
- Enumerating non connected subsets of nodes

### 8.4.2 started

### 8.4.3 planned

- Exhibit the cycles
- provide tool about d-separability (conditionnal independence).
- Creating the implied /bn/ from a /bn/ when conditioning a subset of nodes with fixed values (or different distributions).
- Simulating data sets from a /bn/ and selecting the observations according to a criteria programmed by a function (kind of very rustic (or pedagogical) posterior computation).
- Describing all compatible orders of the nodes within a DAG.
- Statistically studying (from the simulation of the joint probability) relationships within nodes taking into account their natures.
- Introducing causality consequences.
- define and use objects to define a subset of simulated values according to their values.... Generalizing the `dif` argument used in `draw8empidata` function to get adaptative sub-setting.
- exhibiting the selection subsets as used by `draw8empidata`.
- extracting sub /bn/ comprising all the genealogy of a set of nodes provided by the user.
- allow the defintion of the domains to be the result of those of the parent for a child node.

## 8.5 interfacing with other applications

### 8.5.1 fixed

- Computing marginal and conditional probability tables using Grappa pseudo-package.

### 8.5.2 started

- Producing Bugs code

### 8.5.3 planned

- Getting empirical posterior distribution from Bugs outputs
- Use RJags instead of translating to Bugs dialect.
- Importing/exporting /bn/ from deal/bnlearn objects.

## 8.6 computing

### 8.6.1 fixed

- simulating from a /bn/ and computing some elementary descriptive statistics.

### 8.6.2 started

- Providing standard plots for a /bn/ taking into account the nature of the variables.
- Think about intermediate outputs for lengthy computations (especially simulations)

### 8.6.3 planned

- introduce and developp computation around the linear multnormal /bn/ giving the equivalent of what is available in grappa.

## 8.7 programming

### 8.7.1 fixed

- Let **ldf** to be a name file or a data.frame or a function.
- Replace the use of NULL by null objects (`isvide`).
- Implement the general multinomial case (something was started with function `draw8multinom`).

### 8.7.2 started

### 8.7.3 planned

- Checking the strict nesting between the three levels of functions (`f2`, `f1` & `f0`).
- Transform /rbsb/ into an /R/ package.

## 9 Terminology

For such a specialized matter, it is important to use a precise terminology. In that section, key-terms are listed with a definition, some are associated to the major fields of the networks.

**ancestor** A node without any parent, also called *root*.

**ascendants** The ascendants of a node are those nodes which are parent, or parent of one parent, or parent of one parent of one parent,... for the node.

**BN-bn** (Bayesian network) A construction specifying the joint probability of a set of random variables using conditional probabilities. In the context of /bn/, random uni- or multi-variables are named nodes.

**children** The children of a node are those nodes of which the node is a direct parent

**collider** a node having two parents and no child.

**DAG-dag** Directed Acyclic Graph. A necessary and sufficient condition for linked nodes to define a joint probability distribution, is that the graph built on the parentships be a dag.

**descendants** The descendants of a node are those nodes which are children, or children of one child, or children of one child of one child,... for the node.

**dn** Data from/for a Bayesian network.

**d-separation** Two subsets of nodes are d-separated by a third subset of nodes when they are conditionnaly independent for this third subset.

**final-node** A node without any child, also called leaf, sink

**fork** a node having got two children and a unique parent (divergent node).

**gn** Graph of a network. Usually, it is the graph of a Bayesian network but gn object are more general because they can admit cycles which are impossible for a /bn/. In principle, all operations not involving the distributions of the nodes are done on /gn/ and not on /bn/. **bn2gn(bn)** generates the corresponding /gn/ of a given /bn/.

**iin** (Internal integer). Internal sequential numbering of the nodes of an object

**ion** Identification of a node, either with *iin* or *node[variate]*.

**leaf** a node without child (a terminal node).

**node** One of the random uni- or multi-variables constituting the Bayesian network, also used for /gn/ objects.

**nd** Shortcut for node.

**nom** Object of all the node/variate names of an object.

**observational\_equivalence** Two /bn/ are said observationally equivalent when observations cannot be used to distinguish their graph structures.

**pam** Parentship matrix, it contains only zeros and one. If  $M$  is such a matrix, then its rows and columns are associated to the set of nodes and  $M_{ij} = 1$  means that node  $i$  is a parent of node  $j$ . It perfectly defines the graph and has the very interesting property that if there is  $q$  ways to go from node  $i$  to node  $j$  following exactly  $n$  arcs (possibly with repetitions of some of them) then  $N_{ij} = q^n$  where  $N = M^n$ .

**parents** In a /bn/ the distribution of each node is given conditionally to other nodes: the parents are these nodes. Of course, some nodes have no parents.

**root** Used by some authors to designate ancestor nodes.

**sink** Used by some authors to designate terminal nodes.

**variables** In /rbsb/, when no specified variables are univariate; in that case node and variable are identical. Multivariate nodes can comprise several variables.

## 10 Remerciements

To Pierre Pardon for his remarks, they lead to the improvement of the /rbsb/ manual.

To Laurence Mioche for her pertinent suggestions about the extensions of /rbsb/.

## References

- [1] Bishop Ch. M. (2006) Pattern recognition and machine learning (chapter 8). Springer, pp359-422.
- [2] Gammelgaard Susanne & Dethlefsen Claus (2007) deal: Learning Bayesian Networks with Mixed Variables. R package. <http://www.math.aau.dk/~dethlef/novo/deal>
- [3] Green Peter (2005) Grappa, a suite of functions in R for probability propagation in discrete graphical models. <http://www.stats.bris.ac.uk/~peter/Grappa/>
- [4] Pearl Judea (2000) Causality. Models, reasoning, and inference. Cambridge University Presse, 384pp.
- [5] J.-M. Marin & F. Rossi (avril 2004) Découvrez les réseaux bayésiens, Linux Magazine, 60, 56-65
- [6] Plummer Martyn (2007) Jags Version 0.99 manual. <http://www-fis.iarc.fr/~martyn/software/jags/>, 37pp.
- [7] Philippe Leray'home page (<http://asi.insa-rouen.fr/~pleray/>)
- [8] R Development Core Team (2007) R: A Language and Environment for Statistical Computing. <http://www.R-project.org>
- [9] rebastaba, last version available from <http://w3.jouy.inra.fr/unites/miaj/public/matrisq/jbdenis/outi>
- [10] Thomas Andrew (2007) OpenBugs. <http://mathstat.helsinki.fi/openbugs/>.
- [11] winbugs project (1996-2004). <http://www.mrc-bsu.cam.ac.uk/bugs/welcome.shtml>