AALBORG UNIVERSITY

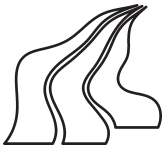
The gRbase Package for Graphical Modelling in R

by

Søren Højsgaard and Claus Dethlefsen

June 2004 R-2004-19

Department of Mathematical Sciences
Aalborg University
Fredrik Bajers Vej 7 G ▪ DK - 9220 Aalborg Øst ▪ Denmark
Phone: +45 96 35 80 80 ▪ Telefax: +45 98 15 81 29
URL: www.math.aau.dk/research/reports/reports.htm



ISSN 1399–2503 ▪ On-line version ISSN 1601–7811

The **gRbase*** Package for Graphical Modelling in R

Søren Højsgaard† and Claus Dethlefsen‡

18th June 2004

Abstract

We have developed a package, called **gRbase**, consisting of a number of classes and associated methods to support the analysis of data using graphical models. It is developed for the open source language, R, and is available for several platforms. The package is intended to be widely extendible and flexible so that package developers may implement further types of graphical models using the available methods.

gRbase contains methods for representing data, specification of models using a formal language, and is linked to **dynamicGraph**, an interactive graphical user interface for manipulating graphs. We show how these building blocks can be combined and integrated with inference engines in the special cases of hierarchical log–linear models (undirected models).

1 Introduction

Graphical models in their modern form have now been around for nearly a quarter of a century. In the present context, a graphical model is a class of statistical models that can be represented by a graph which can be used to identify conditional independence properties. For terminology and theoretical aspects of graphical models, see Lauritzen (1996). Some common examples of graphical models are Bayesian networks (directed graphical models),

*Version 0.1

†Biometry Research Unit, Danish Institute of Agricultural Sciences, Research Centre Foulum, DK–8830 Tjele, Denmark. E–mail: sorenh@agrsci.dk

‡Department of Mathematical Sciences, Aalborg University, DK–9220 Aalborg E, Denmark, E–mail: dethlef@math.aau.dk

log–linear models (undirected models), block-recursive graphical models, and BUGS models.

Various computer programs for inference in graphical models have evolved at different places around the world. A few examples are BUGS (Thomas, 1994), CoCo (Badsberg, 2001), Digram (Klein, Keiding, and Kreiner, 1995), and MIM (Edwards, 2000). Most such packages for graphical models are tailormade to analyse a particular class of models, they have their own command language and are not possible to extend since the code is not open source.

It is of interest to make some of these programs and ideas underlying them go into a general purpose statistical package. The gR initiative (Lauritzen, 2002) is a project launched in 2002 for making facilities in R (R Development Core Team, 2004) so graphical modelling becomes easily accessible and extendable. R is Free Software, Open Source, and runs on various platforms. This facilitates extensions in the form of R packages which may rely on the whole R system.

Recently, some of the existing programs have been made available in R. One example is the **mimR** package (Højsgaard, 2003) which integrates the functionality of the stand–alone program MIM into R. Other packages for graphical modelling have been developed as packages for R, such as **ggm** (Marchetti and Drton, 2003) and **deal** (Bøttcher and Dethlefsen, 2003).

The aim of gR is to provide users and package developers with a framework for graphical modelling in R. The work is organized in three levels. A *core group* works with defining data structures and standard methods, in particular developing the packages **gRbase**, **dynamicGraph** (see Section 6), and **gRaph** (not described here. It provides methods for representing and manipulating graphs efficiently). The *package developers* use the work from the core group to adapt or develop new packages for R that use a common user interface and data structures. Finally, the *group of endusers* use the developed packages in their work with data or model analysis.

In this paper we describe the elements of **gRbase** (available soon from cran.r-project.org) and illustrate how to combine them to create facilities for analysis of hierarchical log–linear models for discrete variables (undirected models). We have chosen to implement in the S4 class system of R.

gmData objects: A fundamental element of **gRbase** is a common class for representing data. No matter the actual representation of data, the important characteristics are contained in a graphical metadata object (gmData). It contains the abstraction of data into a meta data object including variable names and types etc. However, the actual data might not be present or may

be represented by a reference to data, such as a database file. Also, it may be possible to work without data, which may be valuable if the point of interest is in the model alone. Separating the specification of the variables from data has the benefit, that some properties of a model can be investigated without any reference to data, for example decomposability and collapsibility. The `gmData` class is described in Section 3.

`gModel` **objects:** A `gModel` object links a model to a `gmData` object. The model may be specified by either a formula or using an interactive graphical user interface. When defining a `gModel` object, no fitting is done. This is an important difference between model in **gRbase** and e.g. linear models in R. The idea is that a model may be interesting to analyse without any data attached. The `gModel` class is described in Section 4.

Depending on the particular `gModel`, one may choose to fit the model, *i.e.* combine the model and data using a specified *engine*. There may be several engines available, depending on the methodology. For example models may be analysed from either a Bayesian or Frequentist perspective. The result of the fitting process is an object that can be post-processed and provide the results from the analysis. Inference is described in Section 5.

Some features of **gRbase** will be illustrated in the present paper on the basis of the `rats` dataset in the **gRbase** package. The `rats` dataset is from a hypothetical drug trial, where the weight losses of male and female rats under three different drug treatments have been measured after one and two weeks. See Edwards (2000) for more details. The dataset is provided in the **gRbase** package. We will also refer to the dataset `HairEyeColor` (Snee, 1974), included in R.

2 A small sample session

Before describing the core elements of **gRbase**, we present a sample session intended to give the reader a feel for how an enduser will use **gRbase**.

First, data are created as a `gmData` object from an existing `table` object.

```
> library(gRbase)

> data(HairEyeColor)
> gmd.hec <- as(HairEyeColor, "gmData")
> gmd.hec
```

```
Description:
  varNames numberLevels latent varTypes
1     Hair            4  FALSE Discrete
2      Eye            4  FALSE Discrete
3      Sex            2  FALSE Discrete
To see the values of the factors use the 'valueLabels' function
To see the data use the 'observations' function
```

Then, the model with sex independent of hair- and eye-color is defined, fitted with the `loglm`-engine and finally the output is analysed using the `anova` procedure to test the model against the saturated model.

```
> sex.indp <- new("hllm", ~Hair:Eye +
+     Sex, gmd.hec)
> sex.indp <- fit(sex.indp, engine = "loglm")
> anova(getFit(sex.indp))

Call:
loglm(formula = loglm.formula, data = rawdata)

Statistics:
                      X^2 df   P(> X^2)
Likelihood Ratio 29.34982 15 0.01449443
Pearson          28.99286 15 0.01611871
```

The enduser would likely display the models of interest using **dynamicGraph** (see Section 6), and could interact with the graph to investigate properties of the model or change the model. Also, the enduser could change *engine* and for example use the same model, but analysed with a Gibbs sampling algorithm, providing another type of output.

3 `gmData` class – graphical meta data

A `gmData` object by default contains information about variable names, variable types, their labels, their levels (for the discrete variables), and whether the variables are latent or not. Besides, a `gmData` object may contain data or a reference to data, but need not do so.

3.1 Creating a gmData object manually

An object of class gmData may be created by the initialize method using the new command. When an object is printed, only the summary of the variables are printed. Data and value labels are not displayed, but may be accessed separately.

```
> library(gRbase)

> gmd.rats.nodata <- new("gmData", varNames = c("Sex",
+     "Drug", "W1", "W2"), varTypes = c("Discrete",
+     "Discrete", "Continuous", "Continuous"),
+     numberLevels = c(2, 3, NA, NA), valueLabels = list(Sex = c("M",
+         "F"), Drug = c("D1", "D2", "D3")))
> gmd.rats.nodata

Description:
  varNames numberLevels latent   varTypes
1      Sex            2  FALSE   Discrete
2     Drug            3  FALSE   Discrete
3       W1           NA  FALSE Continuous
4       W2           NA  FALSE Continuous
To see the values of the factors use the 'valueLabels' function
To see the data use the 'observations' function

> observations(gmd.rats.nodata)

NULL

> valueLabels(gmd.rats.nodata)

$Sex
[1] "M" "F"

$Drug
[1] "D1" "D2" "D3"
```

The variable types must be from a vector predefined types which may be inspected by the command validVarTypes(). The available types may be extended by the package developers as demonstrated below. The types of the variables are important for the way they are displayed using the package **dynamicGraph**. The type is also important when the models are fitted to data.

```
> oldtypes <- validVarTypes()
> validVartypes <- function() c(oldtypes, "MyVarType")
> validVartypes()

[1] "Discrete"   "Ordinal"    "Continuous" "MyVarType"
```

3.2 Creating a gmData object from a data frame or a table

Typically one will create a gmData object (with data) from a data frame

```
> data(rats)
> class(rats)

[1] "data.frame"

> gmd.rats <- as(rats, "gmData")
> gmd.rats

Description:
  varNames numberLevels latent   varTypes
1      Sex            2  FALSE   Discrete
2     Drug            3  FALSE   Discrete
3       W1           NA  FALSE Continuous
4       W2           NA  FALSE Continuous
To see the values of the factors use the 'valueLabels' function
To see the data use the 'observations' function
```

Also, data from a table can be converted into a gmData object,

```
> data(HairEyeColor)
> class(HairEyeColor)

[1] "table"

> gmd.hec <- as(HairEyeColor, "gmData")
> gmd.hec
```

```
Description:
  varNames numberLevels latent varTypes
1     Hair            4  FALSE Discrete
2      Eye            4  FALSE Discrete
3      Sex            2  FALSE Discrete
To see the values of the factors use the 'valueLabels' function
To see the data use the 'observations' function
```

It is also possible to write conversion methods for other data types, if needed.

3.3 Editing gmData objects

The information contained in a gmData object may be accessed or modified by the methods: varTypes, varNames, numberLevels, latent, valueLabels, and observations.

```
> observations(gmd.rats.nodata) <- rats
> valueLabels(gmd.rats.nodata)$Sex <- c("Male", "Female")
> valueLabels(gmd.rats.nodata)

$Sex
[1] "Male"   "Female"

$Drug
[1] "D1" "D2" "D3"
```

It is possible to extend the variable description of a gmData object with more information, here exemplified by a adding a "short name".

```
> nVar <- nrow(description(gmd.rats.nodata))
> description(gmd.rats.nodata)$shortName <- letters[1:nVar]
> gmd.rats.nodata

Description:
  varNames numberLevels latent   varTypes shortName
1      Sex            2  FALSE   Discrete         a
2     Drug            3  FALSE   Discrete         b
3       W1           NA  FALSE Continuous         c
4       W2           NA  FALSE Continuous         d
To see the values of the factors use the 'valueLabels' function
To see the data use the 'observations' function
```

4 gModel class – graphical models

The general class `gModel` contains a formula object and a `gmData` object. Implementations of different specific graphical model classes can inherit from this class and provide methods for parsing the formula. Here, we illustrate by implementation of a class for hierarchical log–linear models, `hllm`.

For a hierarchical log–linear model, we use the following formula language. The right hand side of the formula is a list of the generators separated by '+'. A generator is specified by variable names with separated by ':'. Commonly used models have short hand notations: saturated model (`~.^.`), main effects (`~.^1`), all kth order interactions (`~.^k`). By an optional argument, `marginal`, it is possible to specify a subset of the variables from the `gmData` object.

The saturated model

```
> m1 <- new("hllm", ~.^., gmd.hec)
> Formula(m1)

~Hair:Eye:Sex
```

The model where sex is independent of hair- and eye-color

```
> m2 <- new("hllm", ~Hair:Eye + Sex, gmd.hec)
```

The model with all main effects

```
> m3 <- new("hllm", ~.^1, gmd.hec)
> Formula(m3)

~Hair + Eye + Sex
```

The saturated model in the hair-eye marginal

```
> m4 <- new("hllm", ~.^., gmd.hec, marginal = c("Hair", "Eye"))
> Formula(m4)

~Hair:Eye
```

Also, the `gModel` class will have associated methods for making inference, which will be treated in Section 5.

5 Inference

In **gRbase** we intend to exploit already existing software by letting these packages do the actual calculations, much like the approach taken in **mimR** which uses the the `MIM` stand alone program as an "inference engine". Hierarchical log–linear models can (when taking a frequentist perspective) be fitted by e.g. `loglm`, `CoCo`, and `MIM`. The default inference engine is `loglm` which is a part of R. In the future, we envision changing the default inference engine to `CoCo`, because `CoCo` facilitates working with discrete and continuous variables in a frequentist setting. However, graphical models can also be analyzed in a Bayesian setting, and in this connection one can envision to use `BUGS` or `JAGS` as inference engine, when these are available in R.

5.1 Model fitting

The `fit` procedure combines a model with an engine and produces an output object that may be further post-processed to yield the results used for inference. As the output objects may be very different, we have set the class name to be a concatenation of the model class and the engine. For example the output from fitting an `hllm` model with the `loglm` engine is of class `hllmloglm`. This way, it is simple to extend with other engines and methods for postprocessing the output.

The default engine for fitting objects of class `hllm` is the `loglm` procedure contained in the **MASS** package.

```
> m2.f <- fit(m2, engine = "loglm")
> m2.f

An object of class "hllmloglm"
Slot "fit":
Call:
loglm(formula = loglm.formula, data = rawdata)

Statistics:
                      X^2 df   P(> X^2)
Likelihood Ratio 29.34982 15 0.01449443
Pearson          28.99286 15 0.01611871

Slot "formula":
~Hair:Eye + Sex
```

```
Slot "gmData":
Description:
  varNames numberLevels latent varTypes
1     Hair            4  FALSE Discrete
2      Eye            4  FALSE Discrete
3      Sex            2  FALSE Discrete
To see the values of the factors use the 'valueLabels' function
To see the data use the 'observations' function
```

To illustrate the `fit` procedure with another engine, we also show how the engine `loglin` may be used. As `loglm` is a front-end to `loglin`, the output from `loglm` is based on the output from `loglin`.

```
> fit(m2, engine = "loglin")

2 iterations: deviation 0
[1] "list"
An object of class "hllmloglin"
Slot "fit":
$lrt
[1] 29.34982

$pearson
[1] 28.99286

$df
[1] 15

$margin
$margin[[1]]
[1] "Hair" "Eye"

$margin[[2]]
[1] "Sex"



Slot "formula":
~Hair:Eye + Sex
```

```
Slot "gmData":
Description:
  varNames numberLevels latent varTypes
1     Hair            4  FALSE Discrete
2      Eye            4  FALSE Discrete
3      Sex            2  FALSE Discrete
To see the values of the factors use the 'valueLabels' function
To see the data use the 'observations' function
```

5.2 Model summary

For each combination of model class and engine, methods for extracting information must be provided by the package developer.

A summary (including certain model properties) of a `gModel` can be achieved using the `summary()` function. Here, the summary function for the `loglm` output is used.

```
> summary(m2.f)

Formula:
~Hair:Eye + Sex
attr(,"variables")
list(Hair, Eye, Sex)
attr(,"factors")
     Sex Hair:Eye
Hair   0        2
Eye    0        2
Sex    1        0
attr(,"term.labels")
[1] "Sex"      "Hair:Eye"
attr(,"order")
[1] 1 2
attr(,"intercept")
[1] 1
attr(,"response")
[1] 0
attr(,".Environment")
<environment: 02F58138>

Statistics:
```

```
                     X^2 df   P(> X^2)
Likelihood Ratio 29.34982 15 0.01449443
Pearson          28.99286 15 0.01611871
```

5.3 Model editing

One important aspect of graphical modelling is the ability to interact with the model. Editing the model means *e.g.* that edges are added or removed and the resulting model is further investigated. Using **dynamicGraph**, it is possible to edit the model using a graphical user interface. The package developer needs to provide the methods `addEdge` and `dropEdge` for his model class.

```
> m5 <- addEdge(m2, "Hair", "Sex")
> Formula(m5)

~Hair:Eye + Hair:Sex

> m6 <- dropEdge(m5, "Hair", "Eye")
> Formula(m6)

~Hair:Sex + Eye
```

In addition, variables may be added or deleted from the model (and thus the associated `gmData` object) by the methods `dropVertex` and `addVertex`, which should also be provided by the package developer.

It is up to the package developer to define the body of these methods. The output should be an object similar to the input object. If for example the input object is a fitted object, the returned object should also be fitted with the same engine.

6 Display and interaction with models

The package **dynamicGraph** (Badsberg, 2004) provides an implementation of an interactive graphical user interface for manipulating graphs, using `tcl/tk` (included in R). The enduser should be provided with a homogeneous user interface no matter what type graphical model he is analysing. The package developer can provide methods that creates the interface with **dynamicGraph**. As an example, the graph for the log–linear model with only main effects, may be displayed by the following command. The enduser can then interactively insert an edge in the graph to obtain the display in Figure 1.

```
> library(dynamicGraph)
> Z <- DynamicGraph(names = as.character(varNames(gmd.hec)),
+     types = as.character(varTypes(gmd.hec)),
+     object = m2)
```

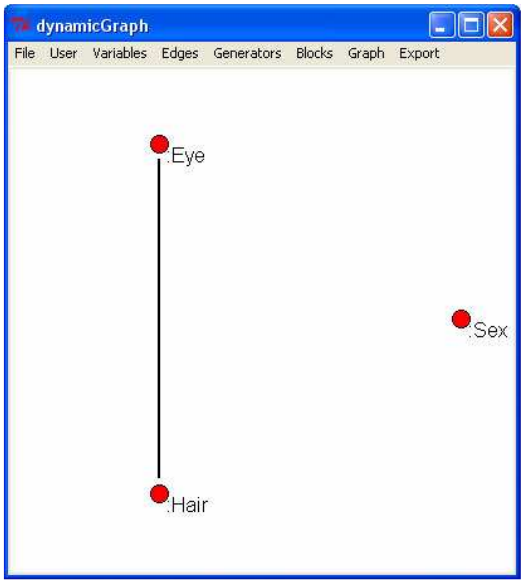


Figure 1: A display of the hierarchical log–linear model using **dynamicGraph**. The enduser can manipulate the graph using a point and click interface.

The package developer can link procedures to items in the user menu, thus providing functionality for easy access to frequently used procedures. The following piece of code will add an item to the User Menu with the label "Forward search". When selected, the procedure `Forwardsearch` will be called with the current object as argument.

```
> UserMenus <- list(MainUser = list(label = "Forward search",
+     command = function(object,
+         ...) Forwardsearch(object,
+         ...)))
> Z <- DynamicGraph(names = as.character(varNames(gmd.hec)),
+     types = as.character(varTypes(gmd.hec)),
+     object = m2, UserMenus = UserMenus)
```

7 Discussion

In this paper we have described **gRbase**, a package that provides tools for creating new packages for graphical models in R. We believe that creating a common package for graphical modelling in R will not only help package developers in mutual support, but will also benefit the group of endusers. It is the intention that **gRbase** can serve as a common platform for future developments of software for graphical modelling in R, such that new packages will have a common interface, will use the gmData objects as basic data objects etc. Packages should have similar user interfaces, both the command interface and graphical user interface and it should not be too difficult to switch between the different model classes.

The **dynamicGraph** package is a standalone package that enables an enduser to manipulate graphs using a point and click interface. The package developers also have the benefit that it is possible to attach a graphical user interface to existing or new packages. The **gRbase** package provides the structure for a common representation of data, no matter the data source. This way facilitates working with models without/before data collection. Also, **gRbase** defines how models should be structured in classes and which methods should be associated by default. The **gRaph** package is planned to include methods for graph computations, which will work as a library for package developers.

Acknowledgements

The members of the gR project are acknowledged for their inspiration.

References

Jens Henrik Badsberg. A guide to CoCo. *Journal of statistical software*, 6 (4), 2001.

Jens Henrik Badsberg. dynamicGraph: Interactive graphical tool for manipulating graphs. URL: http://cran.r-project.org, 2004.

Susanne Bøttcher and Claus Dethlefsen. deal: A package for learning Bayesian networks. *Journal of Statistical Software*, 8(20), 2003.

David Edwards. *Introduction to Graphical Modelling*. Springer-Verlag, 2nd edition, 2000.

Søren Højsgaard. mimR – a package for graphical modelling in R. In K. Hornik, F. Leisch, and A. Zeileis, editors, *Proceedings of the 3rd international workshop on distributed statistical computing*, 2003. ISSN 1609-395X.

John P. Klein, Niels Keiding, and Svend Kreiner. Graphical models for panel studies, illustrated on data from the Framingham heart study. *Statistics in Medicine*, 14:1265–1290, 1995.

Steffen L. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

Steffen L. Lauritzen. gRaphical models in R. *R News*, 2(2):39, 2002.

Giovanni Marchetti and Mathias Drton. ggm: an R package for Gaussian graphical models. URL: http://cran.r-project.org, 2003.

R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. URL `http://www.R-project.org`. ISBN 3-900051-00-3.

R.D. Snee. Graphical display of two-way contingency tables. *The American Statistician*, 28:9–12, 1974.

Andrew Thomas. BUGS: a statistical modelling package. RTA/BCS Modular Languages Newsletter, 1994.