The RHugin Package User's Guide

Kjell Konis

Contents

1 The RHugin package

Hugin [Hugin Expert A/S, 2011b] is a software tool for working with Bayesian networks and decision graphs. The core of Hugin is the Hugin Decision Engine® (HDE), the software library that performs the probability calculations. Hugin provides two interfaces for the HDE: a user-friendly GUI and a set of APIs (C, C++, Java, C#). The **RHugin** package provides an HDE API for the R environment for statistical computing [R Development Core Team, 2011]. R is an open source project that has become increasingly popular in the field of academic statistics (as well as in many others). In contrast to the official HDE APIs, the HDE API for R allows the HDE to be used in an interpreted environment. Hence it provides the same capacity for tinkering with Bayesian networks as the Hugin GUI, while also providing the precise control of the HDE permitted by the APIs. Since it is still the HDE that does all of the calculations, the performance penalty for this added flexibility is minimal.

This user's guide assumes that the reader is already somewhat proficient with R [R Development Core Team, 2011] and with Bayesian networks (see, for example, Jensen [2001]; an in-depth treatment of Bayesian networks can be found in Cowell et al. [1999]). Some experience with the Hugin GUI would also be helpful.

The user's guide is organized as follows. Section 2 discusses installing Hugin and the **RHugin** package. Section 3 demonstrates how to build and make inference from a simple Bayesian network. Finally, section 4 shows how to use the **RHugin** package to access the sensitivity analysis features available in the HDE.

2 Getting started

The **RHugin** package provides an interface linking the Hugin Decision Engine® (HDE) and R, hence it will not be terribly useful unless both the HDE and R are installed on the computer. The HDE is a component of *Hugin Developer* (for commercial use) and *Hugin Researcher* (for academic use). Both of these programs are available from Hugin Expert A/S

http://www.hugin.com

for Microsoft Windows®, SUN Solaris®, Linux (Red Hat Enterprise 4 and compatible distributions) and Apple Macintosh OS X® platforms. Hugin Expert A/S also provides a trial version called *Hugin Lite* — its capabilities are limited but sufficient for evaluating the **RHugin** package. You must have at least one of these three programs (i.e., Hugin Developer, Hugin Researcher, or Hugin Lite) installed on your computer in order to use the **RHugin** package.

2.1 Installing the RHugin package

The **RHugin** website on R-Forge [Theußl and Zeileis, 2008]

http://rhugin.r-forge.r-project.org

contains instructions for installing the **RHugin** package and, optionally, the **Rgraphviz** package. Once the **RHugin** package is installed, run the command

```
> library(RHugin)
```

to load it into your R session.

2.2 Getting help

All of the functions provided in the **RHugin** package are documented using R's built-in help system. For example, the following command will display the help file for the `hugin.domain` function.

```
> help(hugin.domain)
```

A complete list of functions can be obtained using the `ls` function.

```
> ls("package:RHugin")
```

Also, since the **RHugin** package is a wrapper for the Hugin Decision Engine®'s C API, the Hugin API Reference Manual [Hugin Expert A/S, 2011a] can also serve as a reference. The **RHugin** documentation lists which C API calls are used to implement each **RHugin** function.

3 Bayesian networks

The *Building Bayesian Networks* tutorial, available in the Tutorials section of the Hugin website,

http://www.hugin.com/developer/tutorials

shows how to build a simple Bayesian network using the Hugin GUI. This section works through the same example using the **RHugin** package. If you are not already familiar with this tutorial, we recommend you work through it using the Hugin GUI before proceeding.

The goal of this tutorial is to diagnose an apple tree that is losing its leaves. The two possible causes under consideration are (1) the tree is sick and (2) the tree is too dry. The relevant conditional probability tables are given in Table 1. Given that the tree is observed to be losing its leaves, the Bayesian network will be used to compute revised probabilities for the tree being sick and too dry.

3.1 Building the Bayesian network

The first step is to create a new (empty) domain using the `hugin.domain` function. We call our domain `apple`.

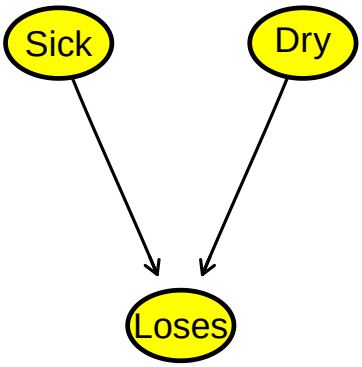
```
> apple <- hugin.domain()
```

We are now ready to add nodes and edges to the `apple` domain and use it for inference.

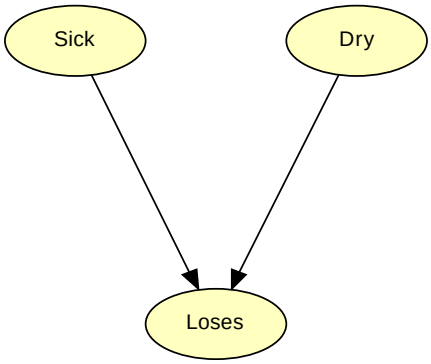
3.1.1 Specifying the structure of the Bayesian network

Next, we use the `add.node` function to add nodes for the variables *Sick*, *Dry* and *Loses*. Each node has two states, namely *yes* and *no*. These states can be specified when the node is created using the optional `states` argument.

```
> add.node(apple, "Sick", states = c("yes", "no"))
> add.node(apple, "Dry", states = c("yes", "no"))
> add.node(apple, "Loses", states = c("yes", "no"))
```



(a) The `apple` domain rendered by the **RHugin** plot method.



(b) The `apple` domain viewed in the Hugin GUI.

Figure 1: The `apple` domain viewed in R and in the Hugin GUI.

Sick = yes	0.1
Sick = no	0.9

(a) Sick

Dry = yes	0.1
Dry = no	0.9

(b) Dry

	Dry = yes		*Dry* = no	
	Sick = yes	*Sick* = no	*Sick* = yes	*Sick* = no
Loses = yes	0.95	0.85	0.90	0.02
Loses = no	0.05	0.15	0.10	0.98

(c) Dry

Table 1: The conditional probability tables for the *Building a Bayesian Network* tutorial.

Note the syntax of the `add.node` function; particularly that the first argument is the domain that `add.node` will act on. This paradigm holds for most of the functions in the **RHugin** package. Further, it is not necessary to assign the output of `add.node`. The functions in the **RHugin** package automatically keep the domain up-to-date.

Next, we use the `add.edge` function to add directed edges from *Sick* to *Loses* and from *Dry* to *Loses*.

```
> add.edge(apple, "Loses", "Dry")
> add.edge(apple, "Loses", "Sick")
```

The structure of the Bayesian network is now complete. We can take a look at the `apple` network using the `plot` method

```
> plot(apple)
```

which uses the **Rgraphviz** [Gentry et al., 2011] package to layout and render the network. The plot is shown in Figure 1a. Figure 1b shows the same network displayed in the Hugin GUI.

3.1.2 Specifying the conditional probability tables in the Bayesian network

After the structure of the network has been specified, the conditional probability tables (CPTs) can be filled in. The contents of a node's CPT depend on the states of the node and on the states of its parents. Since the arrangement of the CPT can get quite complicated, it is recommended that users start with a copy of the CPT obtained from the Hugin Decision Engine® (HDE) which will have the proper arrangement. A copy of the CPT can be obtained using the `get.table` function.

```
> sick.table <- get.table(apple, "Sick")
```

The conditional probability table is stored in R as a data frame containing a factor variable listing the states of the node and a numeric variable `Freq` giving the probability (or frequency) of each state. Initially, all of these values are set to 1. Note that the values in the `Freq` column do not need to sum to one, the HDE automatically normalizes them as necessary.

```
> sick.table
```

```
  Sick Freq
1  yes    1
2   no    1
```

Proceed by changing the probabilities in the `Freq` column of `sick.table` to match those given in Table 1a, then use the `set.table` function to set the CPT for the node *Sick* in the `apple` domain.

```
> sick.table[["Freq"]] <- c(0.1, 0.9)
> set.table(apple, "Sick", sick.table)
```

Lastly, just to be safe, use the `get.table` function to verify that the conditional probability table for the node *Sick* has been properly set in the `apple` domain.

```
> get.table(apple, "Sick")
```

```
  Sick Freq
1  yes  0.1
2   no  0.9
```

Repeat these steps to set the conditional probability table for the node *Dry*.

```
> dry.table <- get.table(apple, "Dry")
> dry.table[["Freq"]] <- c(0.1, 0.9)
> set.table(apple, "Dry", dry.table)
```

The data frame representing the conditional probability table for the node *Loses* is slightly more complicated. It contains three factor variables: the first for *Loses* itself, a second for the parent *Dry*, and a third for the parent *Sick*, so that all 8 cells shown in Table 1c are present.

```
> loses.table <- get.table(apple, "Loses")
> loses.table
```

```
  Loses Dry Sick Freq
1   yes yes  yes    1
2    no yes  yes    1
3   yes  no  yes    1
4    no  no  yes    1
5   yes yes   no    1
6    no yes   no    1
```

```
7   yes  no   no    1
8    no  no   no    1
```

As before, enter the appropriate values from Table 1c in the `Freq` column of `loses.table`, then set the table in the `apple` domain.

```
> loses.table[["Freq"]] <- c(0.95, 0.05, 0.9, 0.1, 0.85, 0.15,
+     0.02, 0.98)
> set.table(apple, "Loses", loses.table)
```

Again, verify that the conditional probability table was properly set using the `get.table` function. The optional `class` argument can be used to retrieve the CPT in different formats. Using `class = "ftable"`, for example, returns the CPT as a flat contingency table (class `ftable` in R) arranged in a similar manner to Table 1c.

```
> get.table(apple, "Loses", class = "ftable")
```

```
      Dry   yes        no
      Sick  yes   no   yes   no
Loses
yes        0.95 0.85 0.90 0.02
no         0.05 0.15 0.10 0.98
```

3.1.3 Saving and loading Hugin domains

The `write.rhd` function allows a Hugin domain to be saved as either a Hugin NET file or a Hugin Knowledge Base (hkb) file.

```
> write.rhd(apple, file = "apple.net", type = "net")
```

Both of these formats are supported by the Hugin GUI. Further, Bayesian networks do not need to be built in R to be used for inference via the **RHugin** package: NET and hkb files created using the Hugin GUI as well as those created using the `write.rhd` function can be loaded using the `read.rhd` function.

```
> apple.copy <- read.rhd("apple.net")
```

3.2 Making inference

A Bayesian network must be compiled before it can be used to make inference. Compilation refers to the task of triangulating the domain and computing a junction tree.

```
> compile(apple)
```

Evidence is entered into a Hugin domain using the `set.finding` function. In this example, the evidence is that the apple tree is losing its leaves. Hence we enter the finding that the node *Loses* is in state *yes*.

```
> set.finding(apple, "Loses", "yes")
```

The `propogate` function invokes the probability propagation algorithm [Lauritzen and Spiegelhalter, 1988] to compute revised beliefs given the evidence in the Hugin domain.

```
> propagate(apple)
```

Finally, after evidence has been entered and propagated, the `get.beliefs` function can be used to retrieve the revised beliefs from a specified node.

```
> get.belief(apple, "Sick")

      yes        no
0.4939956 0.5060044

> get.belief(apple, "Dry")

      yes        no
0.4694323 0.5305677
```

We find that, given the apple tree is losing its leaves, the revised probability that the tree is sick is 0.4939956 and the revised probability that the tree is too dry is 0.4694323.

4 Sensitivity analysis

There is often a subset of the nodes in a Bayesian network that can be regarded as the *input* nodes and another subset that can be regarded as the *output* nodes. Sensitivity analysis refers to the investigation of the effect of inaccuracies in the CPTs of the input nodes on the network's output nodes. The Hugin Decision Engine® (HDE) provides the following mechanism for assessing the effect of a change in the conditional probability table of an input node on the probability of interest (the belief of a particular output node).

Let $\mathsf{P}(A = a|E)$ be the probability of interest: the probability that an output node A takes state a given evidence E in a Bayesian network D. Further, let $x = \mathsf{P}(b_i|\pi)$ be an element (corresponding to the state b_i) of the conditional probability table of a node B in D and let π be the combination of states for the parents of B. When varying $x = p(b_i|\pi)$, the other elements $p(b_j|\pi)$, $j \neq i$ in the conditional probability table must be co-varied so that the probabilities continue to sum to one. Each element of the conditional probability table can thus be viewed as a function $g_j(x)$ of the input probability x that is the subject of the sensitivity analysis. Assuming that the elements of the conditional probability table are co-varied so that their mutual proportional relationship is kept constant, we have

$$g_j(x) = \begin{cases} x & j = i \\ p(b_j|\pi) \cdot \frac{1-x}{1-p(b_i|\pi)} & j \neq i \end{cases}.$$

Given this assumption, Gaag and Renooij [2001] show that the probability of interest can be expressed as the ratio of two linear functions of x

$$\mathsf{P}(A = a|E)(x) = \frac{\alpha x + \beta}{\gamma x + \delta} \tag{1}$$

where α, β, γ and δ are constants that can be computed from D. The function in equation 1 is called the *sensitivity function* and the parameters $\{\alpha, \beta, \gamma, \delta\}$ can be computed using the `get.sensitivity` function in the **RHugin** package.

4.1 Example

Recall the `apple` domain. Suppose we wish to investigate the effect of changing the probability that the tree is too dry on the computed belief that the tree is sick, given that the tree is losing its leaves.

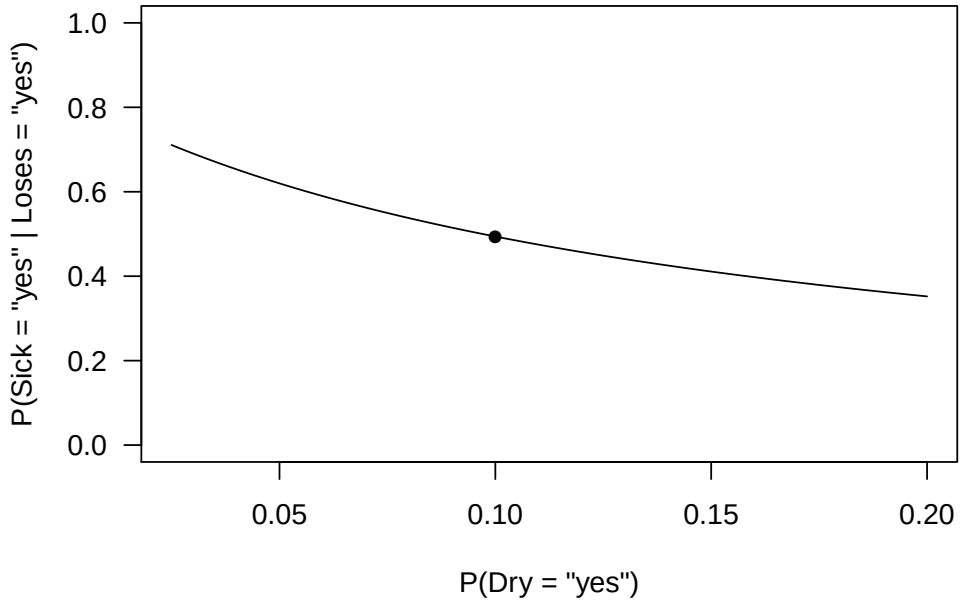


Figure 2: A plot of the sensitivity of the computed belief $\mathsf{P}(Sick = yes|Loses = yes)$ to changes in the input probability $\mathsf{P}(Dry = yes)$. The dot indicates the computed belief for the actual input probability $\mathsf{P}(Dry = yes) = 0.1$

In the framework given above, (1) A is the node *Sick* and a is the state *yes*, (2) the evidence is that the node *Loses* has state *yes*, and (3) B is the node *Dry* and b_i is the state *yes*. First, use the `get.sensitivity` function to compute the parameters of the sensitivity function.

```
> params <- get.sensitivity(apple, "Sick", "yes", Dry = "yes")
> params
```

```
             alpha      beta    gamma     delta
Dry:yes 0.02729258 0.4912664 4.104803 0.5895197
```

Define the sensitivity function

```
> sensitivity.func <- function(x, alpha, beta, gamma, delta) (alpha *
+     x + beta)/(gamma * x + delta)
```

and plot the sensitivity curve for $x = \mathsf{P}(Dry = yes)$ ranging from 0.025 to 0.2.

```
> x <- seq(0.025, 0.2, by = 0.001)
> y <- sensitivity.func(x, params[1], params[2], params[3], params[4])
> plot(x, y)
```

The sensitivity plot is shown in Figure 2.

References

Robert G. Cowell, A.Philip Dawid, Steffen L. Lauritzen, and David J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, New York, 1999. ISBN 0-387-98767-3.

Linda C. van der Gaag and Silja Renooij. Analysing sensitivity data from probabilistic networks. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 530–537, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-800-1.

Jeff Gentry, Li Long, Robert Gentleman, Seth Falcon, Florian Hahne, Deepayan Sarkar, and Kasper Hansen. *Rgraphviz: Provides plotting capabilities for R graph objects*, 2011. R package version 1.30.1.

Hugin Expert A/S. *Hugin API Reference Manual*, 2011a. URL http://www.hugin.com/developer/documentation/api-manuals.

Hugin Expert A/S. *Hugin Expert*, 2011b. URL http://www.hugin.com.

Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer, New York, 2001. ISBN 0-387-95259-4.

S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988. ISSN 00359246. URL http://www.jstor.org/stable/2345762.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL http://www.R-project.org/. ISBN 3-900051-07-0.

Stefan Theußl and Achim Zeileis. Collaborative software development using R-Forge. Report 81, Department of Statistics and Mathematics, Wirtschaftsuniversität Wien, Research Report Series, December 2008. URL http://epub.wu-wien.ac.at/dyn/openURL?id=oai:epub.wu-wien.ac.at:epub-wu-01_e50.